# Table of Contents

## Welcome
- Getting started .............................................................................................................. 21
- Product architecture .......................................................................................................... 22
- Accessing Compute in Prisma Cloud Enterprise Edition ........................................... 22
- Accessing Compute in Prisma Cloud Compute Edition ............................................... 24
- Releases .............................................................................................................................. 26
- Support lifecycle .................................................................................................................. 27
- Definitions ............................................................................................................................ 27
- Prisma Cloud supportability policy ................................................................................... 27
- Third party dependencies .................................................................................................. 27
- Licensing .............................................................................................................................. 28
- Defender types .................................................................................................................. 28
- Workload fluctuation .......................................................................................................... 28
- Example scenarios .............................................................................................................. 29
- Prisma Cloud Enterprise Edition vs Compute Edition .................................................. 30
  - How is Compute delivered? ............................................................................................ 30
  - What are the similarities between editions? ................................................................. 31
  - When should you use Enterprise Edition? .................................................................... 31
  - When should you use Compute Edition? ...................................................................... 31
  - What advantages does Prisma Cloud Enterprise Edition offer over Compute Edition? .......................................................................................................................... 31
  - What are the differences between Prisma Cloud Enterprise Edition and Compute Edition? .......................................................................................................................... 31
  - How do Defender upgrades work? .............................................................................. 32
  - Can you migrate from Compute Edition to Enterprise Edition (SaaS)? ..................... 33
  - Summary .......................................................................................................................... 33
- Utilities and plugins .......................................................................................................... 35

## Install
- Getting started ...................................................................................................................... 37
  - Install guides .................................................................................................................... 39
  - Encryption ......................................................................................................................... 41
- System requirements ......................................................................................................... 42
- Prisma Cloud container images ....................................................................................... 48
  - Retrieving Prisma Cloud images using basic auth ...................................................... 48
  - Retrieving Prisma Cloud images using URL auth ....................................................... 49
- Onebox ................................................................................................................................. 50
  - Install Prisma Cloud ...................................................................................................... 50
  - Configure Console ......................................................................................................... 51
  - Uninstall ........................................................................................................................... 51
  - What’s next? ..................................................................................................................... 51
- Kubernetes .......................................................................................................................... 52
  - Cluster context ................................................................................................................. 52
  - Preflight checklist .......................................................................................................... 54
  - Install Prisma Cloud ...................................................................................................... 54
  - Install Defender .............................................................................................................. 56
  - Install Prisma Cloud on a CRI (non-Docker) cluster .................................................... 58
  - Install Prisma Cloud with Helm charts ................................................................------- 59
  - Alibaba Cloud Container Service for Kubernetes (ACK) .......................................... 60
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Elastic Kubernetes Service (EKS)</td>
<td>61</td>
</tr>
<tr>
<td>Azure Kubernetes Service (AKS)</td>
<td>62</td>
</tr>
<tr>
<td>Azure Container Service (ACS) with Kubernetes</td>
<td>63</td>
</tr>
<tr>
<td>DC/OS Kubernetes</td>
<td>65</td>
</tr>
<tr>
<td>Google Kubernetes Engine (GKE)</td>
<td>65</td>
</tr>
<tr>
<td>IBM Kubernetes Service (IKS)</td>
<td>66</td>
</tr>
<tr>
<td>Redeploying Defenders</td>
<td>67</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>67</td>
</tr>
<tr>
<td>Uninstall</td>
<td>68</td>
</tr>
<tr>
<td>OpenShift</td>
<td>69</td>
</tr>
<tr>
<td>Cluster context</td>
<td>69</td>
</tr>
<tr>
<td>Preflight checklist</td>
<td>71</td>
</tr>
<tr>
<td>Install Prisma Cloud</td>
<td>71</td>
</tr>
<tr>
<td>Control Defender deployments with NodeSelector</td>
<td>78</td>
</tr>
<tr>
<td>Install Prisma Cloud with Helm charts</td>
<td>80</td>
</tr>
<tr>
<td>OpenShift 4</td>
<td>81</td>
</tr>
<tr>
<td>Uninstall</td>
<td>81</td>
</tr>
<tr>
<td>Appendix: NFS PersistentVolume example</td>
<td>81</td>
</tr>
<tr>
<td>Appendix: Implementing SAML federation with a Prisma Cloud Console inside an OpenShift cluster</td>
<td>82</td>
</tr>
<tr>
<td>VMware Tanzu Enterprise PKS</td>
<td>84</td>
</tr>
<tr>
<td>Preflight checklist</td>
<td>84</td>
</tr>
<tr>
<td>Install Prisma Cloud</td>
<td>84</td>
</tr>
<tr>
<td>Docker Swarm</td>
<td>86</td>
</tr>
<tr>
<td>Install Prisma Cloud</td>
<td>86</td>
</tr>
<tr>
<td>Uninstall</td>
<td>92</td>
</tr>
<tr>
<td>Using a private registry</td>
<td>92</td>
</tr>
<tr>
<td>Amazon ECS</td>
<td>95</td>
</tr>
<tr>
<td>Key details</td>
<td>95</td>
</tr>
<tr>
<td>Download the Prisma Cloud software</td>
<td>95</td>
</tr>
<tr>
<td>Create a cluster</td>
<td>96</td>
</tr>
<tr>
<td>Create a security group</td>
<td>96</td>
</tr>
<tr>
<td>Create an EFS file system for Console</td>
<td>97</td>
</tr>
<tr>
<td>Set up a classic load balancer</td>
<td>98</td>
</tr>
<tr>
<td>Deploy Console</td>
<td>98</td>
</tr>
<tr>
<td>Deploy Defender</td>
<td>102</td>
</tr>
<tr>
<td>Create a Prisma Cloud Defender task definition</td>
<td>105</td>
</tr>
<tr>
<td>Using a private registry</td>
<td>106</td>
</tr>
<tr>
<td>DC/OS (Marathon/Mesos)</td>
<td>108</td>
</tr>
<tr>
<td>Deployment architecture</td>
<td>108</td>
</tr>
<tr>
<td>Install Console</td>
<td>108</td>
</tr>
<tr>
<td>Install Defender</td>
<td>110</td>
</tr>
<tr>
<td>Windows</td>
<td>112</td>
</tr>
<tr>
<td>Feature matrix</td>
<td>112</td>
</tr>
<tr>
<td>Deploying Defender on Windows</td>
<td>113</td>
</tr>
<tr>
<td>Registry scanning</td>
<td>113</td>
</tr>
<tr>
<td>Uninstalling Defender</td>
<td>113</td>
</tr>
<tr>
<td>Defender types</td>
<td>115</td>
</tr>
<tr>
<td>Container Defender (Linux and Windows)</td>
<td>117</td>
</tr>
<tr>
<td>Host Defender (Linux and Windows)</td>
<td>117</td>
</tr>
<tr>
<td>Serverless Defender</td>
<td>117</td>
</tr>
<tr>
<td>App Embedded Defender</td>
<td>117</td>
</tr>
<tr>
<td>PCF Defender</td>
<td>119</td>
</tr>
<tr>
<td>Defender capabilities</td>
<td>119</td>
</tr>
<tr>
<td>Connectivity</td>
<td>119</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Service account monitoring</td>
<td>197</td>
</tr>
<tr>
<td>Istio monitoring</td>
<td>199</td>
</tr>
<tr>
<td>Serverless Radar</td>
<td>201</td>
</tr>
<tr>
<td>Layout</td>
<td>201</td>
</tr>
<tr>
<td>Exploring the data</td>
<td>202</td>
</tr>
<tr>
<td>Icons and colors</td>
<td>203</td>
</tr>
<tr>
<td>Notes</td>
<td>205</td>
</tr>
<tr>
<td>Setting up Serverless Radar</td>
<td>206</td>
</tr>
<tr>
<td>What's next?</td>
<td>207</td>
</tr>
<tr>
<td>Prisma Cloud Rules Guide - Docker</td>
<td>208</td>
</tr>
<tr>
<td>Running Docker commands through Defender</td>
<td>208</td>
</tr>
<tr>
<td>About this reference environment</td>
<td>208</td>
</tr>
<tr>
<td>Defend access rules</td>
<td>208</td>
</tr>
<tr>
<td>Defender architecture</td>
<td>227</td>
</tr>
<tr>
<td>Defender design</td>
<td>227</td>
</tr>
<tr>
<td>Why not a kernel module?</td>
<td>227</td>
</tr>
<tr>
<td>Blocking rules</td>
<td>228</td>
</tr>
<tr>
<td>Firewalls</td>
<td>230</td>
</tr>
<tr>
<td>Telemetry</td>
<td>231</td>
</tr>
<tr>
<td>Disabling telemetry</td>
<td>231</td>
</tr>
<tr>
<td>Configure</td>
<td>233</td>
</tr>
<tr>
<td>Rule ordering and pattern matching</td>
<td>235</td>
</tr>
<tr>
<td>Pattern matching</td>
<td>235</td>
</tr>
<tr>
<td>Exemptions</td>
<td>236</td>
</tr>
<tr>
<td>Rule ordering</td>
<td>236</td>
</tr>
<tr>
<td>Disabling rules</td>
<td>237</td>
</tr>
<tr>
<td>Image names</td>
<td>238</td>
</tr>
<tr>
<td>Disaster recovery</td>
<td>239</td>
</tr>
<tr>
<td>Configuring automated backups</td>
<td>239</td>
</tr>
<tr>
<td>Making manual backups</td>
<td>240</td>
</tr>
<tr>
<td>Restoring backups from the Console UI</td>
<td>240</td>
</tr>
<tr>
<td>Restoring backups from twistcli</td>
<td>241</td>
</tr>
<tr>
<td>Downloading backup files</td>
<td>242</td>
</tr>
<tr>
<td>Custom feeds</td>
<td>243</td>
</tr>
<tr>
<td>Supplementing the IP reputation list</td>
<td>243</td>
</tr>
<tr>
<td>Supplementing the malware data feed</td>
<td>243</td>
</tr>
<tr>
<td>Globally allowing CVEs</td>
<td>244</td>
</tr>
<tr>
<td>Test Prisma Cloud's malware detection capabilities</td>
<td>244</td>
</tr>
<tr>
<td>Configuring Prisma Cloud's proxy settings</td>
<td>246</td>
</tr>
<tr>
<td>Settings</td>
<td>246</td>
</tr>
<tr>
<td>Configuring proxy settings</td>
<td>247</td>
</tr>
<tr>
<td>Custom certs for Console access</td>
<td>248</td>
</tr>
<tr>
<td>Configuration options</td>
<td>250</td>
</tr>
<tr>
<td>Securing access to Console with custom certs</td>
<td>250</td>
</tr>
<tr>
<td>Configure scanning</td>
<td>253</td>
</tr>
<tr>
<td>Configuring scan intervals</td>
<td>253</td>
</tr>
<tr>
<td>Last scan time</td>
<td>253</td>
</tr>
<tr>
<td>Scan performance</td>
<td>254</td>
</tr>
<tr>
<td>Scan JavaScript components in manifest but not on disk</td>
<td>254</td>
</tr>
<tr>
<td>Unrated vulnerabilities</td>
<td>255</td>
</tr>
<tr>
<td>Orchestration</td>
<td>255</td>
</tr>
<tr>
<td>User certificate validity period</td>
<td>256</td>
</tr>
<tr>
<td>Configuring the validity period of user certificates</td>
<td>256</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Expired user certificates</td>
<td>256</td>
</tr>
<tr>
<td>Generating new certificates</td>
<td>256</td>
</tr>
<tr>
<td>Enable HTTP access to Console</td>
<td>237</td>
</tr>
<tr>
<td>Set different paths for Defender and Console (with DaemonSets)</td>
<td>258</td>
</tr>
<tr>
<td>Authenticate to Console with certificates</td>
<td>259</td>
</tr>
<tr>
<td>Setting up your certs</td>
<td>259</td>
</tr>
<tr>
<td>What’s next?</td>
<td>260</td>
</tr>
<tr>
<td>Customize terminal output</td>
<td>261</td>
</tr>
<tr>
<td>Specifying a custom message</td>
<td>261</td>
</tr>
<tr>
<td>Output itemized list of compliance issues</td>
<td>262</td>
</tr>
<tr>
<td>Collections</td>
<td>264</td>
</tr>
<tr>
<td>Creating collections</td>
<td>264</td>
</tr>
<tr>
<td>Assigned collections</td>
<td>265</td>
</tr>
<tr>
<td>Assigning collections</td>
<td>267</td>
</tr>
<tr>
<td>Selecting a collection</td>
<td>268</td>
</tr>
<tr>
<td>Collections limitations</td>
<td>269</td>
</tr>
<tr>
<td>Tags</td>
<td>272</td>
</tr>
<tr>
<td>Creating tags</td>
<td>272</td>
</tr>
<tr>
<td>Logon settings</td>
<td>273</td>
</tr>
<tr>
<td>Setting Console’s token validity period</td>
<td>273</td>
</tr>
<tr>
<td>Setting Console’s token validity period</td>
<td>273</td>
</tr>
<tr>
<td>Single sign-on to the Prisma Cloud Support</td>
<td>273</td>
</tr>
<tr>
<td>Basic authentication to Console and API</td>
<td>274</td>
</tr>
<tr>
<td>Strict certificate validation in Defender</td>
<td>274</td>
</tr>
<tr>
<td>Strong passwords for local accounts</td>
<td>275</td>
</tr>
<tr>
<td>Reconfigure Prisma Cloud</td>
<td>276</td>
</tr>
<tr>
<td>Subject Alternative Names</td>
<td>277</td>
</tr>
<tr>
<td>Adding a SAN to Console’s certificate</td>
<td>277</td>
</tr>
<tr>
<td>Credentials store</td>
<td>278</td>
</tr>
<tr>
<td>AWS</td>
<td>279</td>
</tr>
<tr>
<td>Azure</td>
<td>282</td>
</tr>
<tr>
<td>Google Cloud Platform (GCP)</td>
<td>283</td>
</tr>
<tr>
<td>IBM Cloud</td>
<td>284</td>
</tr>
<tr>
<td>Kubeconfig</td>
<td>285</td>
</tr>
<tr>
<td>Authentication</td>
<td>287</td>
</tr>
<tr>
<td>Logging into Prisma Cloud</td>
<td>289</td>
</tr>
<tr>
<td>Login flow</td>
<td>289</td>
</tr>
<tr>
<td>Direct login URL</td>
<td>290</td>
</tr>
<tr>
<td>Integrate with Active Directory</td>
<td>291</td>
</tr>
<tr>
<td>Configuration options</td>
<td>291</td>
</tr>
<tr>
<td>Integrating Active Directory</td>
<td>292</td>
</tr>
<tr>
<td>Adding Active Directory group to Prisma Cloud</td>
<td>292</td>
</tr>
<tr>
<td>Verifying integration with Active Directory</td>
<td>293</td>
</tr>
<tr>
<td>Integrate with OpenLDAP</td>
<td>294</td>
</tr>
<tr>
<td>Integrating OpenLDAP</td>
<td>294</td>
</tr>
<tr>
<td>Verifying integration with OpenLDAP</td>
<td>295</td>
</tr>
<tr>
<td>Integrate Prisma Cloud with Open ID Connect</td>
<td>296</td>
</tr>
<tr>
<td>PingOne</td>
<td>296</td>
</tr>
<tr>
<td>Okta</td>
<td>296</td>
</tr>
<tr>
<td>Prisma Cloud to OIDC user identity mapping</td>
<td>297</td>
</tr>
<tr>
<td>Prisma Cloud to OIDC provider group mapping</td>
<td>298</td>
</tr>
<tr>
<td>Integrate with Okta via SAML 2.0 federation</td>
<td>300</td>
</tr>
<tr>
<td>Setting up Prisma Cloud in Okta</td>
<td>300</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Configuring Console</td>
<td>306</td>
</tr>
<tr>
<td>Granting access by group</td>
<td>306</td>
</tr>
<tr>
<td>Granting access by user</td>
<td>307</td>
</tr>
<tr>
<td>Integrate Google G Suite via SAML 2.0 federation</td>
<td>308</td>
</tr>
<tr>
<td>Setting up Google G Suite</td>
<td>308</td>
</tr>
<tr>
<td>Setting up Prisma Cloud</td>
<td>311</td>
</tr>
<tr>
<td>Integrate with Azure Active Directory via SAML 2.0 federation</td>
<td>312</td>
</tr>
<tr>
<td>Configure Azure Active Directory</td>
<td>312</td>
</tr>
<tr>
<td>Prisma Cloud User to AAD User Identity mapping</td>
<td>317</td>
</tr>
<tr>
<td>Prisma Cloud Groups to AAD Group mapping</td>
<td>317</td>
</tr>
<tr>
<td>Configure Prisma Cloud Console</td>
<td>321</td>
</tr>
<tr>
<td>Prisma Cloud User to AAD User Identity mapping</td>
<td>322</td>
</tr>
<tr>
<td>Integrate with PingFederate via SAML 2.0 federation</td>
<td>325</td>
</tr>
<tr>
<td>Configure PingFederate</td>
<td>325</td>
</tr>
<tr>
<td>Configure Prisma Cloud Console</td>
<td>333</td>
</tr>
<tr>
<td>User account name matching</td>
<td>336</td>
</tr>
<tr>
<td>Group name matching</td>
<td>336</td>
</tr>
<tr>
<td>Integrate with Windows Server 2016 &amp; 2012r2 Active Directory Federation Services via SAML 2.0 federation</td>
<td>338</td>
</tr>
<tr>
<td>Configure Active Directory Federation Services</td>
<td>338</td>
</tr>
<tr>
<td>Active Directory group membership within SAML response</td>
<td>348</td>
</tr>
<tr>
<td>Configure the Prisma Cloud Console</td>
<td>349</td>
</tr>
<tr>
<td>Integrate Prisma Cloud with GitHub</td>
<td>353</td>
</tr>
<tr>
<td>Configure Github as an OAuth provider</td>
<td>353</td>
</tr>
<tr>
<td>Integrate Prisma Cloud with GitHub</td>
<td>354</td>
</tr>
<tr>
<td>Prisma Cloud to GitHub user identity mappings</td>
<td>355</td>
</tr>
<tr>
<td>Integrate Prisma Cloud with OpenShift</td>
<td>359</td>
</tr>
<tr>
<td>Integrate Prisma Cloud with OpenShift</td>
<td>359</td>
</tr>
<tr>
<td>Prisma Cloud to OpenShift user identity mappings</td>
<td>360</td>
</tr>
<tr>
<td>Non-default UPN suffixes</td>
<td>363</td>
</tr>
<tr>
<td>Compute user roles</td>
<td>364</td>
</tr>
<tr>
<td>Summary of available roles</td>
<td>364</td>
</tr>
<tr>
<td>Roles</td>
<td>365</td>
</tr>
<tr>
<td>Assign roles</td>
<td>369</td>
</tr>
<tr>
<td>Assigning roles to Prisma Cloud users</td>
<td>369</td>
</tr>
<tr>
<td>Assigning roles to Prisma Cloud groups</td>
<td>369</td>
</tr>
<tr>
<td>Assigning roles to AD/OpenLDAP/SAML users</td>
<td>370</td>
</tr>
<tr>
<td>Assigning roles to AD/OpenLDAP/SAML groups</td>
<td>370</td>
</tr>
<tr>
<td>Use custom certificates for authorization</td>
<td>371</td>
</tr>
<tr>
<td>Setting up your custom certs</td>
<td>371</td>
</tr>
<tr>
<td>Explicit certificate trust list</td>
<td>372</td>
</tr>
<tr>
<td>Setting up</td>
<td>372</td>
</tr>
<tr>
<td>Credentials store</td>
<td>374</td>
</tr>
<tr>
<td>AWS</td>
<td>375</td>
</tr>
<tr>
<td>Azure</td>
<td>378</td>
</tr>
<tr>
<td>Google Cloud Platform (GCP)</td>
<td>379</td>
</tr>
<tr>
<td>IBM Cloud</td>
<td>380</td>
</tr>
<tr>
<td>Kubeconfig</td>
<td>381</td>
</tr>
<tr>
<td>Vulnerability management</td>
<td>383</td>
</tr>
<tr>
<td>Vulnerability Explorer</td>
<td>385</td>
</tr>
<tr>
<td>Roll-ups</td>
<td>385</td>
</tr>
<tr>
<td>Search tool</td>
<td>386</td>
</tr>
<tr>
<td>Top ten lists</td>
<td>386</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Risk factors</td>
<td>388</td>
</tr>
<tr>
<td>Risk trees</td>
<td>390</td>
</tr>
<tr>
<td>Recalculating statistics</td>
<td>391</td>
</tr>
<tr>
<td>Vulnerability management rules</td>
<td>392</td>
</tr>
<tr>
<td>Creating vulnerability rules</td>
<td>392</td>
</tr>
<tr>
<td>Severity-based actions</td>
<td>392</td>
</tr>
<tr>
<td>Scope</td>
<td>393</td>
</tr>
<tr>
<td>Vendor fixes</td>
<td>393</td>
</tr>
<tr>
<td>Rule exceptions</td>
<td>393</td>
</tr>
<tr>
<td>Custom terminal output</td>
<td>394</td>
</tr>
<tr>
<td>Grace period</td>
<td>395</td>
</tr>
<tr>
<td>Blocking based on vulnerability severity</td>
<td>395</td>
</tr>
<tr>
<td>Blocking specific CVEs</td>
<td>396</td>
</tr>
<tr>
<td>Ignoring specific CVEs</td>
<td>397</td>
</tr>
<tr>
<td>Search CVEs</td>
<td>398</td>
</tr>
<tr>
<td>Searching for a specific CVE</td>
<td>398</td>
</tr>
<tr>
<td>Allow a CVE</td>
<td>399</td>
</tr>
<tr>
<td>Scan reports</td>
<td>400</td>
</tr>
<tr>
<td>View image scan reports</td>
<td>400</td>
</tr>
<tr>
<td>Tagging vulnerabilities</td>
<td>402</td>
</tr>
<tr>
<td>Per-layer vulnerability analysis</td>
<td>403</td>
</tr>
<tr>
<td>Packages in use</td>
<td>406</td>
</tr>
<tr>
<td>Per-finding timestamps</td>
<td>407</td>
</tr>
<tr>
<td>Host and VM image scanning</td>
<td>408</td>
</tr>
<tr>
<td>Scan status</td>
<td>410</td>
</tr>
<tr>
<td>Package types</td>
<td>410</td>
</tr>
<tr>
<td>Customize image scanning</td>
<td>412</td>
</tr>
<tr>
<td>Configuring the severity of reported CVEs</td>
<td>412</td>
</tr>
<tr>
<td>Scanning custom components</td>
<td>412</td>
</tr>
<tr>
<td>Defining a custom vulnerability</td>
<td>412</td>
</tr>
<tr>
<td>Configure registry scans</td>
<td>414</td>
</tr>
<tr>
<td>Deployment patterns</td>
<td>414</td>
</tr>
<tr>
<td>Large-scale registries</td>
<td>414</td>
</tr>
<tr>
<td>Registry scan settings</td>
<td>415</td>
</tr>
<tr>
<td>Configure Prisma Cloud to scan a registry</td>
<td>417</td>
</tr>
<tr>
<td>Additional scan settings</td>
<td>418</td>
</tr>
<tr>
<td>CRI/containerd-only environments</td>
<td>418</td>
</tr>
<tr>
<td>Registry scanning</td>
<td>419</td>
</tr>
<tr>
<td>Registry scanning</td>
<td>419</td>
</tr>
<tr>
<td>Scan images in Alibaba Cloud Container Registry</td>
<td>419</td>
</tr>
<tr>
<td>Scan images on Amazon EC2 Container Registry (ECR)</td>
<td>421</td>
</tr>
<tr>
<td>Scan images in Azure Container Registry (ACR)</td>
<td>422</td>
</tr>
<tr>
<td>Scan images on Docker Registry v2 (including Docker Hub)</td>
<td>423</td>
</tr>
<tr>
<td>Scan images on Google Container Registry</td>
<td>424</td>
</tr>
<tr>
<td>Scan images on Harbor</td>
<td>427</td>
</tr>
<tr>
<td>Scan the IBM Cloud Container Registry</td>
<td>428</td>
</tr>
<tr>
<td>Scan images on Artifactory Docker Registry</td>
<td>429</td>
</tr>
<tr>
<td>Scan images on OpenShift's integrated Docker registry</td>
<td>433</td>
</tr>
<tr>
<td>Trigger registry scans with webhooks</td>
<td>434</td>
</tr>
<tr>
<td>Configure VM image scanning</td>
<td>438</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>438</td>
</tr>
<tr>
<td>Deployment</td>
<td>438</td>
</tr>
<tr>
<td>VM images scan settings</td>
<td>438</td>
</tr>
<tr>
<td>VM images rules</td>
<td>439</td>
</tr>
<tr>
<td>Additional scan settings</td>
<td>440</td>
</tr>
<tr>
<td>Compliance</td>
<td>461</td>
</tr>
<tr>
<td>------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Compliance Explorer</td>
<td>463</td>
</tr>
<tr>
<td>Manage compliance</td>
<td>464</td>
</tr>
<tr>
<td>Enforcement</td>
<td>464</td>
</tr>
<tr>
<td>Creating compliance rules</td>
<td>464</td>
</tr>
<tr>
<td>Reporting full results</td>
<td>465</td>
</tr>
<tr>
<td>CIS Benchmarks</td>
<td>467</td>
</tr>
<tr>
<td>Additional details about Prisma Cloud's implementation of the CIS benchmarks</td>
<td>467</td>
</tr>
<tr>
<td>Prisma Cloud Labs compliance checks</td>
<td>470</td>
</tr>
<tr>
<td>Prisma Cloud Labs Istio compliance checks</td>
<td>471</td>
</tr>
<tr>
<td>Serverless functions compliance checks</td>
<td>472</td>
</tr>
<tr>
<td>Types of issues</td>
<td>472</td>
</tr>
<tr>
<td>Compliance check details</td>
<td>473</td>
</tr>
<tr>
<td>Scanning serverless functions</td>
<td>474</td>
</tr>
<tr>
<td>Windows compliance checks</td>
<td>475</td>
</tr>
<tr>
<td>Custom compliance checks</td>
<td>477</td>
</tr>
<tr>
<td>Creating a new custom image check</td>
<td>477</td>
</tr>
<tr>
<td>Example scripts</td>
<td>478</td>
</tr>
<tr>
<td>Extensible compliance checks</td>
<td>480</td>
</tr>
<tr>
<td>Getting started</td>
<td>480</td>
</tr>
<tr>
<td>Importing an XCCDF Datastream</td>
<td>480</td>
</tr>
<tr>
<td>Setting your policy</td>
<td>481</td>
</tr>
<tr>
<td>Converting XCCDF checklists to datastream format</td>
<td>482</td>
</tr>
<tr>
<td>Trusted images</td>
<td>483</td>
</tr>
<tr>
<td>Feature overview</td>
<td>483</td>
</tr>
<tr>
<td>Trust indicators in the Console UI</td>
<td>484</td>
</tr>
<tr>
<td>Events Viewer</td>
<td>484</td>
</tr>
<tr>
<td>Establishing trust with rules</td>
<td>484</td>
</tr>
<tr>
<td>Creating trust groups manually</td>
<td>485</td>
</tr>
</tbody>
</table>
# Runtime defense

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime defense</td>
<td>501</td>
</tr>
<tr>
<td>Container Models</td>
<td>503</td>
</tr>
<tr>
<td>Capabilities</td>
<td>503</td>
</tr>
<tr>
<td>Learning mode</td>
<td>506</td>
</tr>
<tr>
<td>Active mode</td>
<td>508</td>
</tr>
<tr>
<td>Archived mode</td>
<td>508</td>
</tr>
<tr>
<td>Rules</td>
<td>508</td>
</tr>
<tr>
<td>Best practices</td>
<td>511</td>
</tr>
<tr>
<td>Runtime defense for processes</td>
<td>513</td>
</tr>
<tr>
<td>Process rules</td>
<td>513</td>
</tr>
<tr>
<td>Detection</td>
<td>513</td>
</tr>
<tr>
<td>Explicitly allowed and denied processes</td>
<td>514</td>
</tr>
<tr>
<td>Runtime defense for networking</td>
<td>515</td>
</tr>
<tr>
<td>Networking rules</td>
<td>515</td>
</tr>
<tr>
<td>Runtime defense for file systems</td>
<td>517</td>
</tr>
<tr>
<td>File system integrity defense</td>
<td>518</td>
</tr>
<tr>
<td>Runtime defense for hosts</td>
<td>520</td>
</tr>
<tr>
<td>Enabling host runtime protection</td>
<td>520</td>
</tr>
<tr>
<td>Host runtime policy</td>
<td>520</td>
</tr>
<tr>
<td>File integrity management (FIM)</td>
<td>522</td>
</tr>
<tr>
<td>Monitoring</td>
<td>523</td>
</tr>
<tr>
<td>Audits</td>
<td>526</td>
</tr>
<tr>
<td>Custom runtime rules</td>
<td>527</td>
</tr>
<tr>
<td>Rule library</td>
<td>527</td>
</tr>
<tr>
<td>Expression grammar</td>
<td>527</td>
</tr>
<tr>
<td>Activating custom rules</td>
<td>529</td>
</tr>
<tr>
<td>Limitations</td>
<td>530</td>
</tr>
<tr>
<td>Blocked containers</td>
<td>531</td>
</tr>
<tr>
<td>Blocking action</td>
<td>531</td>
</tr>
<tr>
<td>Viewing blocked containers</td>
<td>531</td>
</tr>
<tr>
<td>Blocked container artifacts</td>
<td>532</td>
</tr>
<tr>
<td>Import and export individual rules</td>
<td>533</td>
</tr>
<tr>
<td>Copying rules</td>
<td>533</td>
</tr>
<tr>
<td>Exporting rules</td>
<td>533</td>
</tr>
<tr>
<td>Importing rules</td>
<td>533</td>
</tr>
<tr>
<td>Discrete blocking</td>
<td>534</td>
</tr>
<tr>
<td>Creating discrete blocking rules</td>
<td>534</td>
</tr>
<tr>
<td>Runtime defense for AWS Fargate</td>
<td>536</td>
</tr>
<tr>
<td>Securing Fargate tasks</td>
<td>536</td>
</tr>
<tr>
<td>Sample task</td>
<td>536</td>
</tr>
<tr>
<td>WAAS for Fargate</td>
<td>540</td>
</tr>
<tr>
<td>Jenkins Fargate example</td>
<td>542</td>
</tr>
</tbody>
</table>
### Continuous integration

<table>
<thead>
<tr>
<th>Jenkins plugin</th>
<th>579</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build and scan flow</td>
<td>579</td>
</tr>
<tr>
<td>Installing the Prisma Cloud Jenkins plugin</td>
<td>579</td>
</tr>
<tr>
<td>Scan artifacts</td>
<td>581</td>
</tr>
<tr>
<td>Ignore image creation time</td>
<td>581</td>
</tr>
<tr>
<td>Post build cleanup</td>
<td>581</td>
</tr>
<tr>
<td>What's next?</td>
<td>581</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Jenkins Freestyle project</th>
<th>583</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting up a Freestyle project for container images</td>
<td>583</td>
</tr>
<tr>
<td>Setting up a Freestyle project for serverless functions</td>
<td>584</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Jenkins Maven project</th>
<th>586</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring Maven</td>
<td>586</td>
</tr>
<tr>
<td>Setting up a Maven project for container images</td>
<td>586</td>
</tr>
<tr>
<td>Setting up a Maven project for serverless functions</td>
<td>588</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Jenkins Pipeline project</th>
<th>590</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting up a Pipeline project for container images</td>
<td>590</td>
</tr>
<tr>
<td>Setting up a Pipeline project for serverless functions</td>
<td>594</td>
</tr>
</tbody>
</table>

Run Jenkins in a container | 596 |

### Access control

<table>
<thead>
<tr>
<th>Role-based access control for Docker Engine</th>
<th>565</th>
</tr>
</thead>
<tbody>
<tr>
<td>Securing remote access</td>
<td>565</td>
</tr>
<tr>
<td>Controlling access to resources</td>
<td>565</td>
</tr>
<tr>
<td>Setting Defender's listener type</td>
<td>567</td>
</tr>
<tr>
<td>Authentication and identity</td>
<td>567</td>
</tr>
<tr>
<td>Configuring Docker client variables</td>
<td>568</td>
</tr>
<tr>
<td>Creating access control rules</td>
<td>568</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>569</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Admission control with Open Policy Agent</th>
<th>571</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Policy Agent</td>
<td>571</td>
</tr>
<tr>
<td>Admission webhook</td>
<td>571</td>
</tr>
<tr>
<td>Configuring the webhook</td>
<td>571</td>
</tr>
<tr>
<td>Kubernetes webhook templates</td>
<td>572</td>
</tr>
<tr>
<td>Validating your setup</td>
<td>573</td>
</tr>
<tr>
<td>Creating custom admission rules</td>
<td>574</td>
</tr>
<tr>
<td>Examples</td>
<td>574</td>
</tr>
</tbody>
</table>

### Incident types

<table>
<thead>
<tr>
<th>Incident types</th>
<th>552</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incident types: Backdoor admin accounts</td>
<td>552</td>
</tr>
<tr>
<td>Backdoor SSH access</td>
<td>553</td>
</tr>
<tr>
<td>Brute force</td>
<td>553</td>
</tr>
<tr>
<td>Crypto miners</td>
<td>554</td>
</tr>
<tr>
<td>Data exfiltration</td>
<td>555</td>
</tr>
<tr>
<td>Hijacked processes</td>
<td>557</td>
</tr>
<tr>
<td>Kubernetes attacks</td>
<td>559</td>
</tr>
<tr>
<td>Lateral movement</td>
<td>559</td>
</tr>
<tr>
<td>Port scanning</td>
<td>560</td>
</tr>
<tr>
<td>Service violation</td>
<td>561</td>
</tr>
</tbody>
</table>

### Troubleshooting

<table>
<thead>
<tr>
<th>Troubleshooting</th>
<th>569</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring Docker client variables</td>
<td>568</td>
</tr>
<tr>
<td>Setting Defender's listener type</td>
<td>567</td>
</tr>
<tr>
<td>Controlling access to resources</td>
<td>565</td>
</tr>
<tr>
<td>Securing remote access</td>
<td>565</td>
</tr>
<tr>
<td>Setting up a Freestyle project for container images</td>
<td>583</td>
</tr>
<tr>
<td>Setting up a Freestyle project for serverless functions</td>
<td>584</td>
</tr>
<tr>
<td>Configuring Maven</td>
<td>586</td>
</tr>
<tr>
<td>Setting up a Maven project for container images</td>
<td>586</td>
</tr>
<tr>
<td>Setting up a Maven project for serverless functions</td>
<td>588</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Jenkins Pipeline project</th>
<th>590</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting up a Pipeline project for container images</td>
<td>590</td>
</tr>
<tr>
<td>Setting up a Pipeline project for serverless functions</td>
<td>594</td>
</tr>
</tbody>
</table>

Run Jenkins in a container | 596 |

### Incidents

<table>
<thead>
<tr>
<th>Incidents</th>
<th>546</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incident types</td>
<td>552</td>
</tr>
<tr>
<td>Viewing incidents</td>
<td>546</td>
</tr>
<tr>
<td>Forensics</td>
<td>548</td>
</tr>
</tbody>
</table>

### Data exfiltration

<table>
<thead>
<tr>
<th>Data exfiltration</th>
<th>555</th>
</tr>
</thead>
</table>

### Lateral movement

<table>
<thead>
<tr>
<th>Lateral movement</th>
<th>559</th>
</tr>
</thead>
</table>

### Crypto miners

| Crypto miners | 554 |

### Brute force

| Brute force | 553 |

### Backdoor SSH access

| Backdoor SSH access | 553 |

### Backdoor admin accounts

| Backdoor admin accounts | 552 |

### What's next?

| What's next? | 581 |

### Post build cleanup

| Post build cleanup | 581 |

### Ignore image creation time

| Ignore image creation time | 581 |

### Port scanning

| Port scanning | 560 |

### Service violation

| Service violation | 561 |

### Configuring Maven

| Configuring Maven | 586 |

### Setting up a Freestyle project for container images

| Setting up a Freestyle project for container images | 583 |

### Setting up a Freestyle project for serverless functions

| Setting up a Freestyle project for serverless functions | 584 |

### Configuring Maven

| Configuring Maven | 586 |

### Setting up a Maven project for container images

| Setting up a Maven project for container images | 586 |

### Setting up a Maven project for serverless functions

| Setting up a Maven project for serverless functions | 588 |

### Jenkins Pipeline project

| Jenkins Pipeline project | 590 |

### Setting up a Pipeline project for container images

| Setting up a Pipeline project for container images | 590 |

### Setting up a Pipeline project for serverless functions

| Setting up a Pipeline project for serverless functions | 594 |

### Run Jenkins in a container

<p>| Run Jenkins in a container | 596 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alerts</td>
<td>667</td>
</tr>
<tr>
<td>Jenkins pipeline on Kubernetes</td>
<td>596</td>
</tr>
<tr>
<td>Scripted Pipeline</td>
<td>597</td>
</tr>
<tr>
<td>CloudBees Core pipeline on Kubernetes</td>
<td>598</td>
</tr>
<tr>
<td>Integrating Prisma Cloud</td>
<td>601</td>
</tr>
<tr>
<td>Pipeline template</td>
<td>601</td>
</tr>
<tr>
<td>CI plugin policy</td>
<td>605</td>
</tr>
<tr>
<td>Vulnerability policy</td>
<td>606</td>
</tr>
<tr>
<td>Compliance policy</td>
<td>606</td>
</tr>
<tr>
<td>Firewalls</td>
<td>607</td>
</tr>
<tr>
<td>Web-Application and API Security (WAAS).</td>
<td>609</td>
</tr>
<tr>
<td>Architecture</td>
<td>609</td>
</tr>
<tr>
<td>Operation</td>
<td>611</td>
</tr>
<tr>
<td>Supported Protocols, Message Parsers and Decoders</td>
<td>612</td>
</tr>
<tr>
<td>Protection Capabilities</td>
<td>612</td>
</tr>
<tr>
<td>Deploying WAAS</td>
<td>618</td>
</tr>
<tr>
<td>Understanding WAAS rule resources and application scope</td>
<td>618</td>
</tr>
<tr>
<td>Deploying WAAS</td>
<td>621</td>
</tr>
<tr>
<td>WAAS Actions</td>
<td>632</td>
</tr>
<tr>
<td>WAAS protections</td>
<td>633</td>
</tr>
<tr>
<td>WAAS Access Controls</td>
<td>640</td>
</tr>
<tr>
<td>cURL Test Commands</td>
<td>644</td>
</tr>
<tr>
<td>Cloud Native Network Firewall (CNNF).</td>
<td>646</td>
</tr>
<tr>
<td>Architecture</td>
<td>646</td>
</tr>
<tr>
<td>Network objects</td>
<td>648</td>
</tr>
<tr>
<td>Enabling CNNF</td>
<td>648</td>
</tr>
<tr>
<td>Secrets</td>
<td>651</td>
</tr>
<tr>
<td>Secrets manager</td>
<td>653</td>
</tr>
<tr>
<td>Theory of operation</td>
<td>653</td>
</tr>
<tr>
<td>Capabilities</td>
<td>654</td>
</tr>
<tr>
<td>Best practices</td>
<td>654</td>
</tr>
<tr>
<td>Integrate with secrets stores</td>
<td>656</td>
</tr>
<tr>
<td>Refresh interval</td>
<td>656</td>
</tr>
<tr>
<td>Secrets Stores</td>
<td>657</td>
</tr>
<tr>
<td>Secrets Stores</td>
<td>657</td>
</tr>
<tr>
<td>AWS Secrets Manager</td>
<td>657</td>
</tr>
<tr>
<td>AWS Systems Manager Parameters Store</td>
<td>658</td>
</tr>
<tr>
<td>Azure Key Vault</td>
<td>659</td>
</tr>
<tr>
<td>CyberArk Enterprise Password Vault</td>
<td>660</td>
</tr>
<tr>
<td>HashiCorp Vault</td>
<td>661</td>
</tr>
<tr>
<td>Inject secrets into containers</td>
<td>662</td>
</tr>
<tr>
<td>Injecting secrets into containers</td>
<td>662</td>
</tr>
<tr>
<td>Injecting secrets: end-to-end example</td>
<td>664</td>
</tr>
<tr>
<td>Setting up Vault</td>
<td>664</td>
</tr>
<tr>
<td>Storing a secret in HashiCorp Vault</td>
<td>664</td>
</tr>
<tr>
<td>Integrating Prisma Cloud and Vault</td>
<td>665</td>
</tr>
<tr>
<td>Creating a rule in Console</td>
<td>665</td>
</tr>
<tr>
<td>Validating the secret is injected</td>
<td>665</td>
</tr>
</tbody>
</table>
Alert mechanism................................................................................................................................. 669
Triggers.................................................................................................................................................. 669
AWS Security Hub................................................................................................................................. 671
  Permissions........................................................................................................................................ 671
  Enabling AWS Security Hub............................................................................................................. 671
  Configuring alert frequency............................................................................................................... 671
  Sending alerts to Security Hub........................................................................................................ 671
  Create new alert profile.................................................................................................................. 671
  Configure the channel..................................................................................................................... 672
  Configure the triggers..................................................................................................................... 672
Cortex XSOAR alerts............................................................................................................................ 674
  Configuring alert frequency............................................................................................................... 674
  Send alerts to XSOAR..................................................................................................................... 674
  Create new alert profile.................................................................................................................. 674
  Configure the channel..................................................................................................................... 675
  Configure the triggers..................................................................................................................... 675
  Configure XSOAR............................................................................................................................ 675
Email alerts.......................................................................................................................................... 678
  Configuring alert frequency............................................................................................................... 678
  Sending email alerts......................................................................................................................... 678
  Create new alert profile.................................................................................................................. 679
  Configure the channel..................................................................................................................... 679
  Configure the triggers..................................................................................................................... 679
Google Cloud Pub/Sub........................................................................................................................ 681
  Configuring alert frequency............................................................................................................... 681
  Sending alerts to Google Cloud Pub/Sub...................................................................................... 681
  Create new alert profile.................................................................................................................. 682
  Configure the channel..................................................................................................................... 682
  Configure the triggers..................................................................................................................... 682
Google Cloud Security Command Center.......................................................................................... 684
  Configuring Google Cloud Security Command Center............................................................... 684
  Configuring alert frequency............................................................................................................... 687
  Sending alerts to Google Cloud SCC............................................................................................. 688
  Create new alert profile.................................................................................................................. 688
  Configure the channel..................................................................................................................... 689
  Configure the triggers..................................................................................................................... 689
IBM Cloud Security Advisor................................................................................................................ 691
  Configuring alert frequency............................................................................................................... 691
  Sending alerts to Security Advisor............................................................................................... 691
  Create new alert profile.................................................................................................................. 692
  Configure the channel..................................................................................................................... 692
  Configure the triggers..................................................................................................................... 692
JIRA Alerts.......................................................................................................................................... 694
  Intelligent issue routing.................................................................................................................... 695
  Configuring alert frequency............................................................................................................... 696
  Integrating Prisma Cloud with JIRA.............................................................................................. 696
  Create new alert profile.................................................................................................................. 697
  Configure the channel..................................................................................................................... 697
  Configure the triggers..................................................................................................................... 698
PagerDuty alerts................................................................................................................................ 699
  Configuring PagerDuty..................................................................................................................... 699
  Configuring alert frequency............................................................................................................... 700
  Sending alerts to PagerDuty............................................................................................................ 700
  Create new alert profile.................................................................................................................. 702
  Configure the channel..................................................................................................................... 702
# TABLE OF CONTENTS

**Kubernetes auditing** .................................................................................................................. 749  
**Rule library** ............................................................................................................................... 749  
**Expression grammar** ................................................................................................................ 749  
**Kubernetes audit events** ........................................................................................................... 749  
**Integrating with self-managed clusters** .................................................................................... 750  
**Integrating with GKE** ................................................................................................................ 752  
**CA bundle** ..................................................................................................................................... 753  
**Testing your setup** ....................................................................................................................... 753  

**Tools** ............................................................................................................................................. 755  

- **twistcli** .......................................................................................................................................... 757  
  - Installing twistcli ............................................................................................................................ 757  
  - Connectivity to Console .................................................................................................................. 757  
  - Functions ......................................................................................................................................... 757  
  - Capabilities ..................................................................................................................................... 758  
  - Install support .................................................................................................................................. 759  
  - Scan images with twistcli ................................................................................................................ 760  
    - Command ....................................................................................................................................... 760  
    - Scan results .................................................................................................................................... 762  
    - Dockerless scan ............................................................................................................................. 763  
    - Simple scan .................................................................................................................................... 764  
    - Scan with detailed report .............................................................................................................. 765  
    - Scan images built with Jenkins in an OpenShift environment .................................................. 768  
    - Scan images when the Docker socket isn't in the default location ............................................. 769  
    - Scan Podman/CRI images ........................................................................................................... 769  
  - CI/CD Automation ......................................................................................................................... 769  
  - Install Console with twistcli ......................................................................................................... 771  
  - Update the Intelligence Stream in offline environments ............................................................ 772  
  - Downloading updates from the IS ............................................................................................... 772  
  - Uploading IS updates to Console ............................................................................................... 773  

**Deployment patterns** .................................................................................................................. 775  

- **Projects** ........................................................................................................................................ 777  
  - Terminology .................................................................................................................................... 777  
  - When to use projects ...................................................................................................................... 777  
  - Architecture .................................................................................................................................... 778  
  - Provisioning flow ........................................................................................................................... 781  
  - Migration strategies ....................................................................................................................... 784  
  - Accessing the API .......................................................................................................................... 786  
  - Provisioning a project ..................................................................................................................... 786  
  - Decommissioning a project ........................................................................................................... 787  
  - Decommissioning disconnected projects .................................................................................... 787  
  - Deploying Defender DaemonSets for Projects (Console UI) ....................................................... 788  
  - Deploying Defender DaemonSets for Projects (twistcli) .............................................................. 788  
  - Deploying Defender DaemonSets for Projects (Prisma Cloud API) ............................................. 788  
  - Limitations ..................................................................................................................................... 788  
- **Best practices for DNS and certificate management** ............................................................... 790  
  - Map out your topology .................................................................................................................. 790  
  - Implement the topology ................................................................................................................. 791  
  - Updating the list of resolvable names for Console ..................................................................... 792  
  - **Caps** ........................................................................................................................................... 793
API.................................................................795
   Access the API.................................................................797
   Accessing the API using Basic authentication.................................797
   Accessing the API using token authentication.................................798
   Accessing the API using a client certificate.....................................798
   Set up a fresh Console...............................................................800
   Creating your first admin user......................................................800
   Setting up your license .....................................................................800
   Automate Defender install.............................................................802
   Install a stand-alone Defender.......................................................802
   Install a stand-alone Defender (no Docker).....................................803
   Install a Defender DaemonSet......................................................804
   Manage compliance policies with the API.......................................806
   Creating and editing policies...........................................................806
   Getting compliance results.............................................................807
   Getting compliance results.............................................................807
   Getting compliance results.............................................................807

Howto..................................................................................809
   Configure an AWS Classic Load Balancer for ECS..........................811
   Configure Prisma Cloud to use Nginx Ingress Controller in Kubernetes........813
   Configure the load balancer type for AWS EKS.................................814
   Provision a Network Load Balancer.................................................814
   Provision an internal load balancer..................................................815
   Deploy Defenders outside an OpenShift Cluster...............................817
   OpenShift and Prisma Cloud Console configuration........................817
   Deploy Prisma Cloud Defender Daemonset in Second OpenShift Cluster......819
   Deploy Prisma Cloud Defender on Windows Server 2016 w/ Containers Node...820
   Configure Prisma Cloud Console's listening ports.............................823
   Provision tenant projects in OpenShift.............................................824
   Configure Prisma Cloud to use Istio Ingress Gateway.......................827
   Setting up Istio.............................................................................827
   Setting up sidecar injection..............................................................829
   Installing Prisma Cloud Console.....................................................829
   Egress Controller for Prisma Cloud Intelligence Stream.....................830
   Creating Istio Ingress and VirtualService resources for Console and Defender traffic..................................................................................830
   Installing the Defender DaemonSet................................................832
   Configuring Prisma Cloud Projects through Istio Ingress Controllers........833
   Implementing SAML federation with a Prisma Cloud Console through Istio Ingress Controllers..................................................................................833
   Disable automatic learning...............................................................835
   Models and learning.........................................................................835
   Workflow....................................................................................835
   Exporting and importing rules from the Console UI............................836
   Exporting and importing rules programmatically................................836
   Performing a rolling upgrade of Defenders........................................838
   Debug data..................................................................................840
   Reviewing Console’s debug logs.....................................................840
   Reviewing Defender’s debug logs....................................................840
Download the software from the following link. You'll need your access token to enter the site.
https://docs.twistlock.com/docs

> Getting started
> Product architecture
> Releases
> Support lifecycle
> Licensing
> Prisma Cloud Enterprise Edition vs Compute Edition
> Utilities and plugins
Getting started

Welcome to the Prisma Cloud product documentation site. Start exploring how our technology can secure your environment.

Need help? Open a case on https://support.twistlock.com or email us at support@twistlock.com.

Preinstall check

Ensure your servers and environment meet the minimum system requirements.

Install the software

Download the latest Prisma Cloud release to your Prisma Cloud Console server or cluster controller. Then install Prisma Cloud using one of the dedicated guides.

Register your license key

Open a browser and navigate to the Prisma Cloud Console. Create an initial admin user, then enter your license key.

Your Prisma Cloud Console is available on https://<consoleServer>:8083

Install a test application

Use your own or check out the Sock Shop.

Explore Prisma Cloud’s core features

The following articles will get you started with Prisma Cloud’s core features:

• Scan and monitor Docker registries
• Review image scan reports
• Create compliance rules
• Create vulnerability rules
• Learn about runtime protection
• Set up a cloud native application firewall
• Set up a cloud native network firewall
Product architecture

Prisma Cloud offers a rich set of cloud workload protection capabilities. Collectively, these features are called Compute. Compute has a dedicated management interface, called Compute Console, that can be accessed in one of two ways, depending on the product you have.

- **Prisma Cloud Enterprise Edition** — Hosted by Palo Alto Networks. Prisma Cloud Enterprise Edition is a SaaS offering. It includes both the Cloud Security Posture Management (CSPM) and Cloud Workload Protection Platform (CWPP) modules. Access the Compute Console, which contains the CWPP module, from the Compute tab in the Prisma Cloud UI.

- **Prisma Cloud Compute Edition** - Hosted by you in your environment. Prisma Cloud Compute Edition is a self-hosted offering that’s deployed and managed by you. It includes the Cloud Workload Protection Platform (CWPP) module only. Download the Prisma Cloud Compute Edition software from the Palo Alto Networks Customer Support Portal. Compute Console is delivered as a container image, so you can run it on any host with a container runtime (e.g. Docker Engine).

The following table summarizes the differences between the two offerings:

<table>
<thead>
<tr>
<th>Capabilities</th>
<th>Prisma Cloud Enterprise Edition</th>
<th>Prisma Cloud Compute Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management interface</td>
<td>Hosted by Palo Alto Networks (SaaS).</td>
<td>Deployed and managed by you in your environment (self-hosted).</td>
</tr>
<tr>
<td>Modules</td>
<td>CSPM and CWPP.</td>
<td>CWPP only.</td>
</tr>
<tr>
<td>Security agents</td>
<td>Deployed and managed by you.</td>
<td>Deployed and managed by you.</td>
</tr>
</tbody>
</table>

Accessing Compute in Prisma Cloud Enterprise Edition

In Prisma Cloud, click the Compute tab to access Compute Console. Think of Prisma Cloud as the outer management interface, and Compute Console as the inner management interface.

To access the Compute Console UI, users must have the Prisma Cloud (outer management interface) System Admin role. Access is denied to users with any other role.

The following screenshot shows the Prisma Cloud UI, or the so-called outer management interface. It can be accessed directly from the Internet. The format of the URL is:

https://app<opt-num>.<opt-region>.prismacloud.io
The following screenshot shows Prisma Cloud with the Compute Console open. Compute Console is the so-called inner management interface. Compute Console’s GUI cannot be directly addressed in the browser. It can only be opened from within the Prisma Cloud UI. It's important to make the distinction between the inner and outer interfaces because a number of Compute components directly address the inner interface, namely:

- Defender, for Defender to Compute Console connectivity.
- twistcli
- Jenkins plugin
- Compute API
You can find the address of Compute Console in Prisma Cloud under **Compute > Manage > System > Downloads**. The address for Compute Console has the following format:

https://<region>.cloud.twistlock.com/<customer>

**Accessing Compute in Prisma Cloud Compute Edition**

In Compute Edition, Palo Alto Networks gives you the management interface to run in your environment. In this setup, you deploy Compute Console directly. There’s no outer or inner interface; there’s just a single interface, and it’s Compute Console. Compute Console’s address, whether an IP address or DNS name, is used for all interactions, namely:

- GUI access from a web browser.
- Defender to Compute Console connectivity.
- twistcli
• Jenkins plugin
• Compute API
Releases

Download links and release notes for official builds can be found at the following link. You'll need your access token to enter the site.

https://docs.twistlock.com/docs

In general, you should stay on the latest major release unless you require a feature or fix from a subsequent maintenance release. We recommend that you upgrade to new major releases as they become available. For more information, see the Prisma Cloud support lifecycle.

The bell icon in Console automatically notifies you when new recommended builds are available:

Prisma Cloud includes various open source components, which may change between releases. Before installing Prisma Cloud, review the components and licenses listed in twistlock-oss-licenses.pdf. This document is included with every download. Changes to components or licenses between releases are highlighted.
Support lifecycle

Because the container ecosystem is rapidly evolving, understanding supportability policies is an important part of keeping your environment supportable and secure. This article describes not only the support policy for Prisma Cloud itself, but also for other software you may integrate it with.

You can always find the most up to date information on available releases, along with download links, on the Releases page.

Definitions

- **Major Releases (X.Y.z)** --
  Include significant new features and changes. These are also known as 'milestones' and include significant new functionality; they are released approximately every four months and include all applicable fixes made in previous releases. These are versions such as 19.07 and 19.11.

- **Maintenance Releases (x.y.Z)** --
  Also known as 'updates', these are released to correct specific problems in previous releases. They incorporate all applicable defect corrections made in prior Maintenance Releases. These are versions such as 19.11.480.

- **End of Life (EOL)** --
  Versions that are no longer supported by Prisma Cloud. Updating to a later version is recommended.

- **Support** --
  Includes not only resolution of technical issues through interactive assistance, but also fixes delivered in maintenance releases to correct problems.

Prisma Cloud supportability policy

Prisma Cloud has an 'n-1' support policy that means the current release ('n') and the previous release ('n-1') receive support.

Note that in some cases, resolution of a problem in the n-1 version may require upgrading to a current build. Prisma Cloud will make commercially reasonable efforts to work with customers that require porting fixes back to the n-1 version but sometimes architectural changes are significant enough between versions that this is practically impossible without making the n-1 version essentially the same as the n version.

Third party dependencies

The container ecosystem is constantly evolving and there are sometimes requirements from third-party vendors that may impact Prisma Cloud's supportability options. For example, Prisma Cloud cannot effectively support third-party software that the vendor (or project) itself no longer supports. This is often most relevant in the case of the Docker Engine itself. Since Prisma Cloud often adds new features that take advantage of new capabilities in Docker, Prisma Cloud only provides commercial support for versions of Docker that Docker itself supports. This ensures that updates can be properly tested and supported throughout customers' environments.

For more information about Prisma Cloud's support for Docker, see System requirements.

Note that we take the same approach to other technologies that interface with Prisma Cloud, such as the host operating systems, orchestrators, registries, and so on. In all cases, Prisma Cloud will provide commercially reasonable assistance to try and resolve problems, but it may require upgrading components to vendor supported versions.
Licensing

You must procure a license for each resource that Prisma Cloud protects. Licenses are valid for one year, after which they can be renewed.

The Prisma Cloud license uses a metering system based on credits (these were called credits in previous releases). Prisma Cloud Compute protects your hosts, containers, and serverless functions using a security agent called Defender. The number of credits you consume directly correlates with the type and mix of Defenders you deploy.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Credits per resource</th>
<th>Defender type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hosts that don't run containers</td>
<td>1 credits</td>
<td>Host Defender</td>
</tr>
<tr>
<td>Hosts that run containers</td>
<td>7 credits</td>
<td>Container Defender</td>
</tr>
<tr>
<td>On-demand containers (such as AWS Fargate, Google Cloud Run)</td>
<td>1 credits</td>
<td>App-Embedded Defender</td>
</tr>
<tr>
<td>Serverless functions (such as AWS Lambda)</td>
<td>1 credits per 1 million invocations</td>
<td>Serverless Defender</td>
</tr>
<tr>
<td>Web Application and API Security (WAAS)</td>
<td>30 credits per protected web-application node (container/pod/host)</td>
<td>• Host Defender • Container Defender • App-Embedded Defender</td>
</tr>
<tr>
<td>Web Application and API Security (WAAS)</td>
<td>No credits consumed</td>
<td>Serverless Defender</td>
</tr>
</tbody>
</table>

Both Prisma Cloud Enterprise Edition (SaaS) and Prisma Cloud Compute Edition (self-hosted) are licensed with the same credits metering system.

Defender types

The type of Defender you deploy depends on the resource you're securing.

- **Host Defender** — Secures legacy hosts (Linux or Windows) that don't run containers.
- **Container Defender** — Secures hosts (Linux or Windows) that run containers. These types of hosts have a container runtime installed, such as Docker Engine or CRI-O. Container Defender protects both the underlying host and any containers it runs, and the license (7 credits) includes coverage for both. A container host consumes 7 credits whether it runs one container or a hundred containers.
- **Container Defender - App Embedded** — Secures containers which are run by a managed service, where the service provider maintains all infrastructure required to run the container, including the underlying host and container runtime. For this type of deployment, a Container App Embedded Defender is embedded into each container to be secured.
- **Serverless Defender** — Secures serverless functions. For this type of deployment, a Serverless Defender is embedded into each function to be secured.

Workload fluctuation

Prisma Cloud Compute Defenders are licensed on the honor system. License limits are not 'hard-enforced'. If you exceed your license count, Palo Alto Networks will notify you with a prominent banner at the top.
of the Prisma Cloud UI, but will neither disable any security functions nor prevent the deployment of additional Defenders. Protection is only disabled when your license expires.

Credit consumption is measured using a 30 day rolling average. To determine if you’re within your licensed coverage, the rolling average is compared to the number of credits in your license.

Prisma Cloud samples of the number of protected nodes hourly, then creates a daily average based on these samples. The preceding 30 daily averages are averaged to determine the credit consumption. If there is less than 30 days of data available, the average is calculated using the days available.

**Example:** Assume you’ve licensed 700 credits to cover 100 container hosts, and usage fluctuates from week to week:

Nov 1-7: Lower demand, uses 90 nodes (630 credits)  
Nov 8-15: Uses 100 nodes (700 credits)  
Nov 16-22: Uses 100 nodes (700 credits)  
Nov 23-30: High demand, uses 110 nodes (770 credits)

Even though you used 770 credits for a short period of time, you’re still properly licensed because the 30 day rolling average is 700:

\[
\frac{630 + 700 + 700 + 770}{4} = 700 \text{ credits}
\]

### Example scenarios

For hosts and containers, the number of credits you need to procure depends on the number of Defenders you intend to deploy.

**Example:** Assume you have a Kubernetes cluster with 100 nodes (hosts). You deploy a Container Defender to each node. You would procure a license with 700 credits:

100 container hosts * 7 credits per container host = 700 credits

Serverless functions are licensed based on the number of invocations, and averaged over the period of a month. Every 1 million serverless invocations in a one month period counts as 1 credits.

**Example:** Assume you have 5 functions and, in total, they are invoked 8 million times over the period month. You would procure a license with 8 credits.

8 million invocations / 1 million invocations per credits = 8 credits

**Example:** Assume you have a web application running over 4 NGINX containers. You would procure a license with 120 credits.

4 protected containers * 30 credits per container = 120 credits
Prisma Cloud Enterprise Edition vs Compute Edition

This article describes the key differences between Compute in Prisma Cloud Enterprise Edition and Prisma Cloud Compute Edition. Use this guide to determine which option is right for you.

Prisma Cloud Compute - Deployment Options

<table>
<thead>
<tr>
<th></th>
<th>Enterprise Edition*</th>
<th>Compute Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core feature set</td>
<td>Identical</td>
<td></td>
</tr>
<tr>
<td>What does it protect?</td>
<td>Hosts, containers, serverless</td>
<td></td>
</tr>
<tr>
<td>Where can it protect?</td>
<td>Any cloud, including your own datacenter</td>
<td></td>
</tr>
<tr>
<td>Licensing</td>
<td>Identical (by workload)</td>
<td></td>
</tr>
<tr>
<td>Pricing</td>
<td>Identical</td>
<td></td>
</tr>
<tr>
<td>Who runs the Console?</td>
<td>Palo Alto Networks</td>
<td>You do</td>
</tr>
<tr>
<td>Who runs the Defenders?</td>
<td>You do</td>
<td></td>
</tr>
</tbody>
</table>

Compute is delivered in one of two packages:

- **Prisma Cloud Enterprise Edition (SaaS)** — Single pane of glass for both CSPM (Cloud Security Posture Management) & CWPP (Cloud Workload Protection Platform). Compute (formerly Twistlock, a CWPP solution) is delivered as part of the larger Prisma Cloud system. Palo Alto Networks runs, manages, and updates Compute Console for you. You deploy and manage Defenders in your environment. You access the Compute Console from a tab within the Prisma Cloud user interface.

- **Prisma Cloud Compute Edition (self-hosted)** — Stand-alone, self-operated version of Compute (formerly Twistlock). Download the entire software suite, and run it in any environment. You deploy and manage both Console and Defenders.
What are the similarities between editions?

Both Enterprise Edition (SaaS) and Compute Edition (self-hosted) are built on the same source base. The Console container image we run for you in Enterprise Edition is the exact same container image we give to you in Compute Edition to run in your environment. We are committed to supporting and developing both versions without any feature divergence.

When should you use Enterprise Edition?

Prisma Cloud Enterprise Edition is a good choice when:

- You want a single platform that protects both the service plane (public cloud resource configuration) and the compute plane.
- You want convenience. We manage your Console for you. We update it for you. You get a 99.9% uptime SLA.

When should you use Compute Edition?

Prisma Cloud Compute Edition is a good choice when:

- You want full control over your data.
- You’re operating in an air-gapped environment.
- You want to implement enterprise-grade multi-tenancy with one Console per tenant. For multi-tenancy, Compute Edition offers a feature called Projects.

What advantages does Prisma Cloud Enterprise Edition offer over Compute Edition?

When the Prisma Cloud CSPM and CWPP tools work together, Palo Alto Networks can offer economies of scale by sharing data (so called “data overlays”). The Prisma Cloud CSPM tool has always offered the ability to integrate with third party scanners, such as Tenable, to supplement configuration assessments with host vulnerability data. Starting with the Nov 2019 release of Enterprise Edition, the CSPM tool can utilize the host vulnerability data Compute Defender collects as part of its regular scans. Customers that have already licensed one workload for a host can leverage that single workload for configuration assessments by the CSPM tool, host vulnerability scanning (via Compute Defender), and host runtime protection (via Compute Defender).

Customers can expect additional "data overlays" in future releases, including better ways to gauge security posture with combined dashboards.

What are the differences between Prisma Cloud Enterprise Edition and Compute Edition?

There are a handful of differences between Enterprise Edition and Compute Edition. Consider these differences when deciding which edition is right for you.

Projects:

There is no support for Compute Projects in the Prisma Cloud Enterprise Edition (PCEE). However, Enterprise Edition (EE) does offer alternatives that support Project’s primary use cases. The common use cases for Projects are:

- Isolation: Each team has a dedicated Console so that other teams can’t see each other’s data. Prisma Cloud EE supports isolation with multiple Prisma Cloud tenants, one per team, with one Compute Console per tenant. Within a single PCEE tenant, Compute Console also offers isolation to data access.
based on cloud account filtering. Contact Customer Success to create multiple tenants. Note that the license count shown in the Prisma Cloud UI is per tenant, not the aggregate across multiple tenants. If you want to control customer tenant deployments yourself, use Compute Edition.

- Centralized policies in scale Projects. A central Compute Console pushes the same set of policies to all sub-Console. This setup isn’t supported in Prisma Cloud EE. If you need it, use Compute Edition.

- Scale. Visibility from more than 5k Defenders in a single Compute Console. Compute supports 5K Defenders per Compute Console. Future releases will support more. If you need more Defender support, Prisma Cloud EE supports scale with multiple Prisma Cloud tenants, one per team, with one Compute Console per tenant. However, if you need to see all data in all tenants in a single Console, use Compute Edition.

Syslog:
- Prisma Cloud Enterprise Edition Consoles do not emit syslog events for customer consumption. Since we operate the Console service for you, we monitor Console on your behalf.
- Prisma Cloud Enterprise Edition Defenders still emit syslog events that you can ingest. Syslog messages from Defender cover runtime and firewall events. For more details, see the article on logging.

User management:
- In Prisma Cloud Enterprise Edition, user and group management, as well as auth, is handled by the outer Prisma Cloud app in Enterprise Edition.
- As such, Compute Console in SaaS mode disables AD, OpenLDAP, and SAML integration in the Compute tab.

RBAC:
- In Prisma Cloud Enterprise Edition, you can assign roles to users to control their level of access to Prisma Cloud. These roles are mapped to Compute roles internally.
- With this integration, users can scope what Prisma Cloud users can see in the Compute tab by cloud account.
- For the CI/CD use case (i.e. using the Jenkins plugin or twistcli to scan images in the CI/CD pipeline), there’s a new permission group called “Build and Deploy Security”.

Assigned Collections:
- Enforcing views of resource subsets for read-only users, as defined by filters (collections), is controlled by the cloud accounts assigned to users in Prisma Cloud. Manual collection assignment is not supported in PCEE.

How do Defender upgrades work?

Upgrades work a little differently in each edition.

- **Prisma Cloud Enterprise Edition (SaaS)** – When an upgrade is available, a button appears in the Compute UI. When you click it, your tenant’s version of Console is upgraded. The process takes about 10 seconds. After Console is upgraded, you must take action to manually upgrade all of your deployed Defenders.

  To minimize the manual effort required to maintain PCEE, the 20.04 release added support for automatically upgrading deployed Defenders. Starting in June 2020, Palo Alto Networks will start automatically upgrading PCEE Compute Consoles.

With the introduction of the Defender auto-upgrade feature in 20.04, Palo Alto Networks will start automatically upgrading PCEE Compute Consoles in June, 2020, to avoid manual efforts in Console management.

- **Prisma Cloud Compute Edition (self-hosted)** – You fully control the upgrade process. When an upgrade is available, customers are notified via the bell icon in Console. Clicking on it directs you to the latest
software download. Deploy the new version of Console first, then manually upgrade all of your deployed Defenders.

After fully upgrading to Compute 20.04 (both Console and Defenders), you'll no longer need to manually upgrade Defenders. For all subsequent upgrades, Compute Console will manage Defender upgrades for you. The Prisma Cloud Compute architecture now supports automatic in-place upgrades of Defender (no manual operator intervention or redeployment is required). Automatic Defender upgrades are triggered as soon as Console is upgraded. The auto-upgrade mechanism is enabled by default, but it can be disabled if you want more control over the upgrade process.

You must still manually upgrade App-Embedded Defenders.

Can you migrate from Compute Edition to Enterprise Edition (SaaS)?

We are working on an automated migration plan. In the meantime, if you have an Enterprise Edition license, you can still manually export policies from your self-hosted Console and import them into the SaaS Console. Then redeploy your Defenders and CI plugins to point to the SaaS Console, and start using the new SaaS Console. If you have any questions, contact the Customer Success team for assistance.

To be clear, no one will be forced to migrate from Compute Edition to SaaS. Compute Edition will always be available for customers that choose to download and run the software themselves, anywhere. Compute and SaaS literally run the exact same bits, so customers have the flexibility to decide which deployment option makes sense for them.

Summary

The following table summarizes the key differences between Enterprise Edition (SaaS) and Compute Edition (self-hosted). For gaps, we provide a date we intend to deliver a solution.

<table>
<thead>
<tr>
<th>Capability</th>
<th>Notes</th>
<th>Delivery Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projects</td>
<td>If you need Projects, use Compute Edition. Projects will not be ported to Prisma Cloud Enterprise Edition.</td>
<td>-</td>
</tr>
<tr>
<td>Syslog</td>
<td>Collecting customer requirements</td>
<td>TBD</td>
</tr>
<tr>
<td>User management</td>
<td>There’s no gap in functionality. No work to be done.</td>
<td>-</td>
</tr>
<tr>
<td>RBAC</td>
<td>Available via role mapping in Prisma Cloud Enterprise Edition. No work to be done.</td>
<td>-</td>
</tr>
<tr>
<td>Assigned collections</td>
<td>Available via role mapping in Prisma</td>
<td>-</td>
</tr>
<tr>
<td>Capability</td>
<td>Notes</td>
<td>Delivery Date</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>--------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Cloud Enterprise Edition. No work to be done.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defender auto upgrade</td>
<td>Available as an option. No work to be done.</td>
<td>-</td>
</tr>
<tr>
<td>Compute Edition to Enterprise Edition migration</td>
<td>In planning. This page will be updated when there's a date to share.</td>
<td>TBD</td>
</tr>
</tbody>
</table>
Utilities and plugins

All Prisma Cloud utilities and plugins can be downloaded directly from the Console UI. They are also bundled with the release tarball you download from the Customer Support Portal.

To download the utilities from Prisma Cloud Console, go to Manage > System > Downloads. From there, you can download:

- Jenkins plugin.
- Linux Container Defender image.
- twistcli for Linux, macOS, and Windows.
Install

Prisma Cloud can be deployed to almost any environment. The guides in this section show you how to deploy Prisma Cloud to a variety of on-prem and public cloud environments.

- Getting started
- System requirements
- Prisma Cloud container images
- Onebox
- Kubernetes
- OpenShift
- VMware Tanzu Enterprise PKS
- Docker Swarm
- Amazon ECS
- DC/OS (Marathon/Mesos)
- Windows
- Defender types
Getting started

Prisma Cloud software consists of two components: Console and Defender. Install Prisma Cloud in two steps. First, install Console. Then install Defender.

Console is Prisma Cloud’s management interface. It lets you define policy and monitor your environment. Console is delivered as a container image.

Defender protects your environment according to the policies set in Console. There are a number of Defender types, each designed to protect a specific resource type.

Install one Console per environment. Here, environment is loosely defined because the scope differs from organization to organization. Some will run a single instance of Console for their entire environment. Others will run an instance of Console for each of their prod, staging, and dev environments. Prisma Cloud supports virtually any topology.

The primary concern for most customers getting started with Prisma Cloud is securing their container environment. To do this, install Container Defender on every host that runs containers. Container orchestrators typically provide native capabilities for deploying an agent, such as Defender, to every node in the cluster. Prisma Cloud leverages these capabilities to install Defender. For example, Kubernetes and OpenShift, offer DaemonSets, which guarantee that an agent runs on every node in the cluster. Prisma Cloud Defender, therefore, is deployed in Kubernetes and OpenShift clusters as a DaemonSet.

In this section, you’ll find dedicated install guides for all popular container platforms. Each guide shows how to install Prisma Cloud for that given platform.

As you adopt other cloud-native technologies, Prisma Cloud can be extended to protect those environments too. Deploy the Defender type best suited for the job. For example, today you might use Amazon EKS (Kubernetes) clusters to run your apps. This part of your environment would be protected by Container Defender. Later you might adopt AWS Lambda functions. This part of your environment would be secured by Serverless Defender. Extending Prisma Cloud to protect other types of cloud-native technologies calls for deploying the right Defender type.
to Console via websocket to retrieve policies and send data. In Compute Edition (self-hosted), the Defender websocket connects to Console on port 8084 (configurable at install-time). The following diagram shows the key connections in Compute Edition.

![Diagram showing key connections in Compute Edition]

Install guides

Start your install with one of our dedicated guides.

<table>
<thead>
<tr>
<th>Install procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Onebox</strong></td>
<td>Simple, quick install of Prisma Cloud on a single, stand-alone host. Installs both Console and Defender onto a host. Suitable for evaluating Prisma Cloud in a small, self-contained environment. You can extend the environment by installing Defender on additional hosts.</td>
</tr>
</tbody>
</table>
| **Kubernetes**          | Prisma Cloud runs on any implementation of Kubernetes, whether you build the cluster from scratch or use a managed solution (also known as Kubernetes as a service). We’ve tested and validated the install on:  
  - Amazon Elastic Kubernetes Service (Amazon EKS)  
  - Azure Container Service with Kubernetes  
  - Azure Kubernetes Service (AKS)  
  - DC/OS Kubernetes  
  - Google Kubernetes Engine (GKE)  
  - IBM Kubernetes Service (IKS)  
  - Alibaba Cloud Container Service for Kubernetes  
  In some cases, there is a dedicated section for installing on a specific cloud provider’s managed solution. When there is no dedicated section, use the generic install method. |
| **OpenShift**           | Prisma Cloud offers native support for OpenShift.                                                                                           |
| **Pivotal Container Service** | Pivotal Container Service (PKS) is built on the latest stable OSS distribution of Kubernetes. Prisma Cloud always supports the latest version of Kubernetes, so installing Prisma Cloud on PKS is easy. Follow our dedicated PKS install guide, which mirrors the Kubernetes install flow. |
| **Docker Swarm**        | Prisma Cloud supports Docker Swarm using Swarm-native features. Deploy Console as a service and rely on Swarm to provide built-in high availability. Deploy Defender as a global service, which guarantees that Defender is automatically deployed to each worker node in the cluster. |
**Install procedure** | **Description**
---|---
**Amazon ECS** | To install Prisma Cloud, deploy Console to your cluster with a task definition. Then configure the launch configuration for cluster members to download and run Defenders, guaranteeing that every node is protected.

**DC/OS** | Prisma Cloud supports DC/OS or Mesos clusters that use either the Kubernetes or Marathon scheduler. For the Kubernetes scheduler, use our standard Kubernetes install procedure. For the Marathon scheduler, install Console using the `twistlock.sh` install script. Then deploy Defenders to the cluster as a Marathon application, which guarantees that each node in the cluster runs an instance of Defender.

**Windows** | Install Defender on Windows hosts running containers. Defender is installed using a PowerShell script. Note that while Defenders can run on both Windows and Linux hosts, Console can only run on Linux. Windows Defenders are designed to interoperate with the Linux-based Console to send data and retrieve policy.

**Encryption**

All network traffic is encrypted with TLS (https) for user to Console communication. Likewise, all Defender to Console communication is encrypted with TLS (WSS).

The Prisma Cloud database is not encrypted at rest, however all credentials and otherwise secure information is encrypted with AES 256 bit encryption. If you require data at rest to be encrypted, then underlying persistence storage `/var/lib/twistlock` can be mounted with one of the many options that support this.
System requirements

Before installing Prisma Cloud, verify that your environment meets the minimum requirements.

Hardware

**Metal:** Prisma Cloud has the following hardware requirements:

- **Console**
  - When fewer than 100 Defenders are connected, Console requires 1GB of RAM and 10GB of persistent storage.
  - When more than 100 Defenders are connected, Console requires 3GB of RAM and 50GB of persistent storage.
- **Defender** — 256MB of RAM and 8GB of host storage.

  Defender uses cgroups to cap resource usage at 512MB of RAM and 900 CPU shares; typical load is ~1-5% CPU and 30-70MB RAM

  Defender stores its data in /var. When allocating disk space for Defender, be sure the required space is available in /var.

  Defenders are designed to be portable containers that collect data. Any data that must be persisted is sent to Console for storage. Defenders themselves do not require persistent storage. Do not deploy persistent storage for Defenders because it can corrupt Defender files.

- **Registry scanning** — 2GB of RAM, 20GB of storage, and 2 CPU cores.
- **CI integration (Jenkins, twistcli)** — Required storage space depends on the size of the scanned images. The required disk space is 1.5 times the size of the largest image to be scanned, per executor. For example, if you have a Jenkins instance with two executors, and your largest container image is 500MB, then you need at least 1.5GB of storage space (500MB * 1.5 * 2).

**VMs:** Prisma Cloud has been tested on the following hypervisors:

- Microsoft Hyper-V
- VirtualBox
- VMware

**Cloud:** Prisma Cloud can run on nearly any cloud IaaS platform. Prisma Cloud has been tested on the following services:

- Amazon Web Services
- Google Compute Engine
- IBM Cloud
- Microsoft Azure
- Oracle Cloud

**Host operating systems**

Prisma Cloud is supported on the following host operating systems:
<table>
<thead>
<tr>
<th>Distro</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Linux 2</td>
<td>Latest LTS release v2</td>
</tr>
<tr>
<td>CentOS</td>
<td>CentOS 7, CentOS 8</td>
</tr>
<tr>
<td>CoreOS</td>
<td>CoreOS latest stable channel (CoreOS 2345.3.0)</td>
</tr>
<tr>
<td></td>
<td><strong>Host Defender</strong> isn't supported on CoreOS. CoreOS is specifically designed for running containers. Install <strong>Container Defender</strong> instead.</td>
</tr>
<tr>
<td>Debian</td>
<td>Debian 9 (Stretch), Debian 10 (Buster)</td>
</tr>
<tr>
<td>GCOOS</td>
<td>Container-Optimized OS on Google Cloud latest</td>
</tr>
<tr>
<td></td>
<td><strong>GCOOS is purposefully minimalistic. It doesn't support installing new packages or writing new bins. Hence, Prisma Cloud's vulnerability detection on GCOOS only covers Docker and Kubernetes package binary detection.</strong></td>
</tr>
<tr>
<td>Red Hat</td>
<td>Red Hat Enterprise Linux 7, Red Hat Enterprise Linux 8</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>Ubuntu Server 18.04 LTS, 16.04 LTS</td>
</tr>
<tr>
<td></td>
<td><strong>Ubuntu 14.04 LTS does not support system call monitoring. The required kernel option is not enabled in the default kernel.</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Host Defender isn't supported on Ubuntu 14.04.</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Console must be installed on a supported Linux operating system, either natively or through virtualization (such as Hyper-V). Defender is supported on Windows Server 2016 (vulnerability and compliance scanning), and Windows Server 2019 (vulnerability scanning, compliance scanning, and runtime defense for containers).</strong></td>
</tr>
<tr>
<td>VMware</td>
<td>Photon OS 3.0 latest release</td>
</tr>
</tbody>
</table>

**Kernel**

Prisma Cloud Defender requires the following kernel capabilities. More info about each capability can be found on the Linux capabilities man page.

- `CAP_NET_ADMIN`
- `CAP_SYS_ADMIN`
- `CAP_SYS_PTRACE`
• CAP_AUDIT_CONTROL
• CAP_MKNOD
• CAP_SETFCAP

When running on a Docker host, Prisma Cloud Defender uses the following files/folder on the host:

• /var/run/docker.sock—Required for accessing Docker runtime.
• /var/lib/twistlock—Required for storing Prisma Cloud data.
• /dev/log—Required for writing to syslog.

Docker Engine

Since Prisma Cloud often adds new features that take advantage of new Docker capabilities, Prisma Cloud provides commercial support only for the versions of Docker Engine that Docker itself supports. This ensures that updates can be properly tested and supported throughout customers’ environments. Prisma Cloud follows the same support lifecycle policy as Docker Enterprise Edition. For more information, see Docker’s Maintenance Lifecycle.

New versions of Docker Engine are supported shortly after they are released. Prisma Cloud supports the following and later versions. Only official mainstream Docker releases are supported.

• CE 19.03, 18.06
• EE 19.03, 18.03

For storage drivers, overlay2, overlay, and device mapper are supported. For more information, please refer to Docker’s guide to selecting a storage driver.

The versions of Docker Engine listed in this section apply to versions independently installed on a host. These versions might not be the same as the versions shipped as a part of an orchestrator, such as Red Hat OpenShift. In such cases, Prisma Cloud supports the version of Docker Engine that ships with any Prisma Cloud-supported version of the orchestrator.

Podman

Podman is a daemon-less container engine for developing, managing, and running OCI containers on Linux. The twistcli tool can use the preinstalled Podman binary to scan CRI images.

The following version of Podman are supported:

• Podman 1.6

Orchestrators

Prisma Cloud is supported on the following orchestrators. We support the following versions of official mainline vendor/project releases.

<table>
<thead>
<tr>
<th>Orchestrator</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker Swarm</td>
<td>CE 19.03 &amp; 18.06, EE 19.03 &amp; 18.03</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>1.17, 1.18, 1.19</td>
</tr>
<tr>
<td>OpenShift</td>
<td>3.11 - docker version only, 4.2, 4.3, 4.4, 4.5</td>
</tr>
<tr>
<td>Orchestrator</td>
<td>Version</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>VMware Tanzu Application Service - TAS (formerly Pivotal Cloud Foundry - PCF PAS)</td>
<td>N, N-1 support policy</td>
</tr>
<tr>
<td>EKS</td>
<td>1.14</td>
</tr>
</tbody>
</table>

**Container runtimes**

Prisma Cloud supports the following container runtimes:

<table>
<thead>
<tr>
<th>Container runtime</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker</td>
<td>See the Docker section</td>
</tr>
</tbody>
</table>
| cri-containerd    | Client version: 1.2.8  
Server version: 1.2.8                                                     |
| CRI-O             | OS 4.2 - crio version 1.14.12-10  
OS 4.3 - crio version 1.16.2-6  
OS 4.4 - crio version 1.17.4-18.dev.rhaos4.4.gitfb8131a.el8               |

**Istio**

Prisma Cloud supports Istio 1.0 - 1.6.

**File systems**

If you're deploying Prisma Cloud Console to AWS and you're using the EFS file system, the following minimum performance characteristics are required:

- **Performance mode**: General purpose
- **Throughput mode**: Provisioned. Provision 0.1 MiB/s per deployed Defender. For example, if you plan to deploy 10 Defenders, provision 1 MiB/s of throughput.

**Jenkins**

Prisma Cloud provides a Jenkins plugin that scans images for vulnerabilities after they are built.

The Prisma Cloud plugin supports the following Jenkins versions:

- 2.190, 2.204 and 2.222 (these versions support Podman <2)
- 2.235 (this version doesn't support Podman)

Prisma Cloud tests the latest (or near-latest) LTS releases of Jenkins. These versions are guaranteed to be compatible with Prisma Cloud. Other recent LTS versions should also work. However, if you're having issues with the Prisma Cloud plugin, we recommend that you upgrade to the version of Jenkins that Prisma Cloud has tested.
Shell
For Linux, Prisma Cloud depends on the Bash shell. For Windows, Prisma Cloud depends on PowerShell. The shell environment variable `DOCKER_CONTENT_TRUST` should be set to 0 or unset before running any commands that interact with the Prisma Cloud cloud registry, such as Defender installs or upgrades.

Browsers
Prisma Cloud supports the latest versions of Chrome, Safari, and Edge. For Microsoft Edge, we only support the new Chromium-based version (80.0.361 and later).

Image base layers
Prisma Cloud can protect containers built on nearly any base layer operating system. Comprehensive Common Vulnerabilities and Exposures (CVE) data is provided for the following base layers:

- Alpine
- Amazon Linux container image
- Amazon Linux 2
- BusyBox
- CentOS
- Debian
- Red Hat Enterprise Linux
- SUSE
- Ubuntu (LTS releases only)
- Windows Server

Serverless
Prisma Cloud can protect AWS Lambda functions at runtime. Prisma Cloud supports the following runtimes:

Serverless Runtime using Lambda Layers (including auto-protect)
- Node.js 10.x
- Python 2.7, 3.6, 3.7 and 3.8

Serverless Runtime using manually embedded Defenders
- C# (.NET Core 2.1, .NET Core 3.1)
- Java 8, Java 11
- Node.js 10.x
- Python 2.7, 3.6, 3.7 and 3.8

Prisma Cloud can also scan serverless functions for vulnerabilities and compliance benchmarks. Prisma Cloud supports the following runtimes for vulnerability and compliance scans in AWS Lambda, Google Cloud Functions, and Azure Functions:

Serverless Vulnerability and Compliance scanning
- C# (.NET Core 2.1, .NET Core 3.1)
- Java 8, Java 11
- Node.js 10.x
- Python 2.7, 3.6, 3.7 and 3.8
Default UID/GID

When installing Console with `twistlock.sh`, the Prisma Cloud data folder (`var/lib/twistlock`) owner and group are set to a UID and GID of 2674, and the Console process runs as user 2674 by default. To configure Console to run as root, open `twistlock.cfg` and set `RUN_CONSOLE_AS_ROOT` to true before running `twistlock.sh`. You must run Console as root if you want it to listen on port 443 or some other privileged port.

When installing Console in a Kubernetes or OpenShift cluster, the Console process runs as root by default. Defenders always run as root. Although Defenders run as root, they drop the capabilities they don’t need. For a list of capabilities that Defenders retain, see Defender Architecture.
Prisma Cloud container images

Prisma Cloud images can be retrieved from a cloud registry, and are available to all current customers. This option simplifies a lot of workflows, especially the install flow.

All builds, including private builds, are published to the registry. Private builds temporarily address specific customer issues. Unless you’ve been asked to use a private build by a Prisma Cloud representative during the course of a support case, you should only pull officially published builds.

You can optionally manage Prisma Cloud images in your own registry. You can push the Prisma Cloud images to your own private registry, and manage them from there as you see fit. The Console image is delivered as a .tar.gz file in the release tarball. The Defender image can be downloaded from Console, under Manage > System > Downloads, or from the Prisma Cloud API.

There are two different methods for accessing images in the cloud registry:

• Basic authorization.
• URL authorization.

The length of time that images are available on the cloud registry complies with our standard n-1 support lifecycle.

Retrieving Prisma Cloud images using basic auth

Authenticate using `docker login` or `podman login`, then retrieve the Prisma Cloud images using `docker pull` or `podman pull`. For basic authorization, the registry is accessible at `registry.twistlock.com`.

Image names contain a version string. The version string must be formatted as X_Y_Z, where X is the major version, Y is the minor version, and Z is the patch number. For example, 19.07.363 should be formatted as 19_07_363. For example:

`registry.twistlock.com/twistlock/defender:defender_19_07_363`.

Prerequisites:

• You have your Prisma Cloud access token.

STEP 1 | Authenticate with the registry.

```bash
$ docker (or podman) login registry.twistlock.com
Username: 
Password: 
```

Where Username can be any string, and Password must be your access token.

STEP 2 | Pull the Console image from the Prisma Cloud registry.

```bash
$ docker (or podman) pull registry.twistlock.com/twistlock/console:console_<VERSION>
```
STEP 3 | Pull the Defender image from the Prisma Cloud registry.

```bash
$ docker (or podman) pull registry.twistlock.com/twistlock/
defender:defender_<VERSION>
```

Retrieving Prisma Cloud images using URL auth

Retrieve Prisma Cloud images with a single command by embedding your access token into the registry URL. For URL authorization, the registry is accessible at `registry-auth.twistlock.com`.

By embedding your access token into the registry URL, you only need to run `docker pull` or `podman pull`. The `docker login` or `podman login` command isn’t required.

The format for the registry URL is: `registry-auth.twistlock.com/tw_<ACCESS-TOKEN>/<IMAGE>:<TAG>`

Image names contain a version string. The version string must be formatted as `X_Y_Z`, where `X` is the major version, `Y` is the minor version, and `Z` is the patch number. For example, `19.07.363` should be formatted as `19_07_363`. For example:

`registry.twistlock.com/twistlock/defender:defender_19_07_363`.

Prerequisites:

- You have a Prisma Cloud access token.
- The Docker or Podman client requires that repository names be lowercase. Therefore, all characters in your access token must be lowercase. To convert your access token to lowercase characters, use the following command:

```bash
$ echo <ACCESS-TOKEN> | tr ':upper:' ':lower:'
```

STEP 1 | Pull the Console image from the Prisma Cloud registry.

```bash
$ docker (or podman) pull \
registry-auth.twistlock.com/tw_<ACCESS-TOKEN>/twistlock/
console:console_<VERSION>
```

STEP 2 | Pull the Defender image from the Prisma Cloud registry.

```bash
$ docker (or podman) pull \
registry-auth.twistlock.com/tw_<ACCESS-TOKEN>/twistlock/
defender:defender_<VERSION>
```
Onebox

Onebox provides a quick, simple way to install both Console and Defender onto a single host. It provides a fully functional, self-contained environment that is suitable for evaluating Prisma Cloud.

Install Prisma Cloud

Install Onebox with the `twistlock.sh` install script.

**Prerequisites:**

- Your host meets the minimum [system requirements](#).
- You have a license key.
- Port 8083 is open. Port 8083 (HTTPS) serves the Console UI. You can configure alternative ports in `twistlock.cfg` before installing.
- Port 8084 is open. Console and Defender communicate with each other on this port.

**STEP 1 | Download** the latest Prisma Cloud release to the host where you want to install Onebox.

**STEP 2 | Extract the tarball.** All files must be in the same directory when you run the install.

```
$ mkdir twistlock
$ tar -xzvf prisma_cloud_compute_<VERSION>.tar.gz -C twistlock/
```

**STEP 3 | Configure Prisma Cloud for your environment.**

Open `twistlock.cfg` and review the default settings. The default settings are acceptable for most environments.

*If your Docker socket is in a custom location, update twistlock.cfg before continuing. By default, Prisma Cloud expects to find the Docker socket in `/var/run/docker.sock`. If it’s not located there on your host, open twistlock.cfg in an editor, find the `DOCKER_SOCKET` variable, and update the path.*

**STEP 4 | Install Prisma Cloud.**

```
$ sudo ./twistlock.sh -s onebox
```

- `-s` --
  
  Agree to EULA.

- `-z` --
  
  (Optional) Print additional debug messages. Useful for troubleshooting install issues.

- `onebox` --
  
  Install both Console and Defender on the same host, which is the recommended configuration. Specify `console` to install just Console.

**STEP 5 | Verify that Prisma Cloud is installed and running:**

```
$ docker ps --format "table ({{.ID}}\t{{.Status}}\t{{.Names}})"
CONTAINER ID        STATUS              NAMES
764ecb72207e        Up 5 minutes        twistlock_defender_<VERSION>
```
Configure Console

Create your first admin user and enter your license key.

**STEP 1** | Open Prisma Cloud Console. In a browser window, navigate to 'https://<CONSOLE>:8083', where <CONSOLE> is the IP address or DNS name of the host where Console runs.

**STEP 2** | Create your first admin user.
Consider using `admin` as the username. It's a convenient choice because `admin` is the default user for many of Prisma Cloud's utilities, including twistcli.

**STEP 3** | Enter your license key.

Uninstall

Use the `twistlock.sh` script to uninstall Prisma Cloud from your host. The script stops and removes all Prisma Cloud containers, removes all Prisma Cloud images, and deletes the `/var/lib/twistlock` directory, which contains your logs, certificates, and database.

**STEP 1** | Uninstall Prisma Cloud.

```
$ sudo ./twistlock.sh -u
```

**STEP 2** | Verify that all Prisma Cloud containers have been stopped and removed from your host.

```
$ docker ps -a
```

**STEP 3** | Verify that all Prisma Cloud images have been removed from your host.

```
$ docker images
```

What’s next?

Install Defender on each additional host you want to protect.
Kubernetes

This procedure is optimized to get Prisma Cloud installed in your Kubernetes cluster quickly. There are many ways to install Prisma Cloud, but we recommend that you start with this procedure first. You can tweak the install procedure after you have validated that this install method works.

Prisma Cloud is installed with a utility called `twistcli`, which is bundled along with the rest of the Prisma Cloud software. The `twistcli` utility generates YAML configuration files for Console and Defender. You then create the required objects in your cluster with `kubectl create`. This two step approach gives you full control over the objects created. You can inspect, customize, and manage the YAML configuration files in source control before deploying Console and Defender.

Prisma Cloud Console is created as a Deployment, which ensures a single copy of Console is always up and available. Prisma Cloud Defenders are deployed as a DeamonSet, which guarantees an instance of Defender runs on each worker node in the cluster.

In order to improve the availability of the Console service, the orchestrator should be free to run Console on any healthy node. If a node were to go down, the orchestrator should be able to simply reschedule Console somewhere else. To enable this capability, Console’s default YAML configuration files:

- **Deploy a persistent volume (PV), where Console can save its state.** No matter where Console runs, it must have access to its state. In order for PVs to work, every node in the cluster must have access to shared storage. Depending on your cloud provider, and whether Kubernetes is managed or unmanaged, setting up storage can range from easy to difficult. Google Cloud Kubernetes Engine (GKE), for example, offers it as an out-of-the box capability, so it requires zero configuration. If you build your cluster by hand, however, you might need to configure something like NFS.

- **Expose Console to the network using a load balancer.** Console must always be accessible. It serves a web interface, and it communicates policy with all deployed Defenders. A load balancer ensures that Console is reachable no matter where it runs in the cluster.

Cluster context

Prisma Cloud can segment your environment by cluster. For example, you might have three clusters: test, staging, and production. The cluster pivot in Prisma Cloud lets you inspect resources and administer security policy on a per-cluster basis.
Defenders in each DaemonSet are responsible for reporting which resources belong to which cluster. When deploying a Defender DaemonSet, Prisma Cloud tries to determine the cluster name through introspection. First, it tries to retrieve the cluster name from the cloud provider. As a fallback, it tries to retrieve the name from the corresponding kubeconfig file saved in the credentials store. Finally, you can override these mechanisms by manually specifying a cluster name when deploying your Defender DaemonSet.

Both the Prisma Cloud UI and twistcli tool accept an option for manually specifying a cluster name. Let Prisma Cloud automatically detect the name for provider-managed clusters. Manually specify names for self-managed clusters, such as those built with kops.

Radar lets you explore your environment cluster-by-cluster. You can also create stored filters (also known as collections) based on cluster names. Finally, you can scope policy by cluster. Vulnerability and compliance rules for container images and hosts can all be scoped by cluster name.

There are some things to consider when manually naming clusters:

- If you specify the same name for two or more clusters, they’re treated as a single cluster.
- For GCP, if you have clusters with the same name in different projects, they’re treated as a single cluster. Consider manually specifying a different name for each cluster.
• Manually specifying names isn’t supported in Manage > Defenders > Manage > DaemonSet. This page lets you deploy and manage DaemonSets directly from the Prisma Cloud UI. For this deployment flow, cluster names are retrieved from the cloud provider or the supplied kubeconfig only.

Preflight checklist

To ensure that your installation goes smoothly, work through the following checklist and validate that all requirements are met.

**General**
• You have a valid Prisma Cloud license key and access token.

**Cluster**
• You have provisioned a Kubernetes cluster that meets the minimum system requirements and runs a supported Kubernetes version.
• You have set up a Linux or macOS system as your cluster controller, and you can access the cluster with kubectl.
• The nodes in your cluster can reach Prisma Cloud’s cloud registry (registry-auth.twistlock.com).
• Your cluster can create PersistentVolumes and LoadBalancers from YAML configuration files.

**Runtimes**
• Prisma Cloud supports Docker Engine, CRI-O, and cri-containerd. For more information, see the system requirements

**Permissions**
• You can create and delete namespaces in your cluster.
• You can run kubectl create commands.

**Firewalls and ports**
Validate that the following ports are open.

**Prisma Cloud Console:**
• Incoming: 8083, 8084
• Outgoing: 443, 53

**Prisma Cloud Defenders:**
• Incoming: None
• Outgoing: 8084

**Install Prisma Cloud**

Use twistcli to install the Prisma Cloud Console and Defenders. The twistcli utility is included with every release. After completing this procedure, both Prisma Cloud Console and Prisma Cloud Defenders will be running in your Kubernetes cluster.

If you’re installing Prisma Cloud on Amazon Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS), or Azure Container Service with Kubernetes, a number of tweaks are required to the installation procedure. For more details, see the relevant sections in this article.
Download the Prisma Cloud software

Download the Prisma Cloud software to any system where you run `kubectl` to administer your cluster.

**STEP 1** | Go to Releases, and copy the link to current recommended release.

**STEP 2** | Download the release tarball to your cluster controller.

```bash
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```

**STEP 3** | Unpack the release tarball.

```bash
$ mkdir twistlock
$ tar xvzf twistlock_<VERSION>.tar.gz -C twistlock/
```

**Install Console**

Install Console, exposing the service using a load balancer.

*If you’re using NFSv4 for persistent storage in your cluster, we recommend that you use the nolock, noatime and bg mount options for your PersistentVolume. After generating the Console YAML file, add the following mount options to your PersistentVolume definition.*

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: twistlock-console
labels:
  app-volume: twistlock-console
annotations:
  volume.beta.kubernetes.io/mount-options: "nolock,noatime,bg"
```

**STEP 1** | On your cluster controller, navigate to the directory where you downloaded and extracted the Prisma Cloud release tarball.

**STEP 2** | Generate a YAML configuration file for Console, where `<PLATFORM>` can be `linux` or `osx`.

The following command saves `twistlock_console.yaml` to the current working directory. If needed, you can edit the generated YAML file to modify the default settings.

```bash
$ <PLATFORM>/twistcli console export kubernetes --service-type LoadBalancer
```

**STEP 3** | Deploy Console.

```bash
$ kubectl create -f twistlock_console.yaml
```

**STEP 4** | Wait for the service to come up completely.

```bash
$ kubectl get service -w -n twistlock
```

**Configure Console**

Create your first admin user and enter your license key.
STEP 1 | Get the public endpoint address for Console.

```bash
$ kubectl get service -o wide -n twistlock
```

STEP 2 | (Optional) Register a DNS entry for Console's external IP address. The rest of this procedure assumes the DNS name for Console is `yourconsole.example.com`.

STEP 3 | (Optional) Set up a custom cert to secure Console access.

STEP 4 | Open a browser window, and navigate to Console. By default, Console is served on HTTPS on port 8083. For example, go to `https://yourconsole.example.com:8083`.

STEP 5 | Create your first admin user.

STEP 6 | Enter your Prisma Cloud license key.

STEP 7 | Defender communicates with Console using TLS. Update the list of identifiers in Console's certificate that Defenders use to validate Console's identity.

1. Go to `Manage > Defenders > Names`.
2. In the `Subject Alternative Name` table, click `Add SAN`, then enter Console's IP address or domain name (e.g. `yourconsole.example.com`). Any Defenders deployed outside the cluster can use this name to connect to Console.
3. In the `Subject Alternative Name` table, click `Add SAN` again, then enter `twistlock-console`. Any Defenders deployed in the same cluster as Console can use Console's service name to connect. Note that the service name, `twistlock-console`, is not the same as the pod name, which is `twistlock-console-XXXX`.

Install Defender

Defender is installed as a DaemonSet, which ensures that an instance of Defender runs on every node in the cluster. Use `twistcli` to generate a YAML configuration file for the Defender DaemonSet, then deploy it using `kubectl`. You can use the same method to deploy Defender DaemonSets from both macOS and Linux kubectl-enabled cluster controllers.

The benefit of declarative object management, where you work directly with YAML configuration files, is that you get the full "source code" for the objects you create in your cluster. You can use a version control tool to manage and track modifications to config files so that you can delete and reliably recreate DaemonSets in your environment.

If you don't have kubectl access to your cluster, you can deploy Defender DaemonSets directly from the Console UI.

The following procedure shows you how to deploy Defender DaemonSets with twistcli using declarative object management. Alternatively, you can generate Defender DaemonSet install commands in the Console UI under `Manage > Defenders > Deploy > DaemonSet`. Install scripts work on Linux hosts only. For macOS and Windows hosts, use twistcli to generate Defender DaemonSet YAML configuration files, and then deploy it with kubectl, as described in the following procedure.

If you're using CRI-O or containerd, pass the `--cri` flag to twistcli (or enable the CRI option in the Console UI) when generating the Defender YAML or Helm chart.
You can run both Prisma Cloud Console and Defenders in the same Kubernetes namespace (e.g. twistlock). Be careful when running kubectl delete commands with the YAML file generated for Defender. This file contains the namespace declaration, so comment out the namespace section if you don’t want the namespace deleted.

For provider managed clusters, Prisma Cloud automatically gets the cluster name from the cloud provider. To override the the cloud provider’s cluster name, use the --cluster option. For self-managed clusters, such as those built with kops, you must manually specify a cluster name with the --cluster option.

STEP 1 | Determine the Console service’s external IP address.

```bash
$ kubectl get service -o wide -n twistlock
```

STEP 2 | Generate a `defender.yaml` file, where:

The following command connects to Console’s API (specified in `--address`) as user `ADMIN` (specified in `--user`), and generates a Defender DaemonSet YAML config file according to the configuration options passed to `twistcli`.

The `--cluster-address` option specifies the address Defender uses to connect to Console. For Defenders deployed in the cluster where Console runs, specify Prisma Cloud Console’s service name, `twistlock-console`. For Defenders deployed outside the cluster, specify either Console’s external IP address, exposed by the LoadBalancer, or better, Console’s DNS name, which you must manually set up separately.

The following command directs Defender to connect to Console using its service name. Use it for deploying a Defender DaemonSet inside a cluster.

```bash
$ <PLATFORM>/twistcli defender export kubernetes
--address https://yourconsole.example.com:8083
--user <ADMIN_USER>
--cluster-address twistlock-console
```

- `<PLATFORM>` can be linux or osx.
- `<ADMIN_USER>` is the name of the initial admin user you just created.

STEP 3 | (Optional) Schedule Defenders on your Kubernetes master nodes.

If you want to also schedule Defenders on your Kubernetes master nodes, change the DaemonSet’s toleration spec. Master nodes are tainted by design. Only pods that specifically match the taint can run there. Tolerations allow pods to be deployed on nodes to which taints have been applied. To schedule Defenders on your master nodes, add the following tolerations to your DaemonSet spec.

```yaml
tolerations:
- key: "node-role.kubernetes.io/master"
  operator: "Exists"
  effect: "NoSchedule"
```

STEP 4 | Deploy the Defender DaemonSet.

```bash
$ kubectl create -f defender.yaml
```

STEP 5 | Open a browser, navigate to Console, then go to Manage > Defenders > Manage to see a list of deployed Defenders.
Install Prisma Cloud on a CRI (non-Docker) cluster

Kubernetes lets you set up a cluster with the container runtime of your choice. Prisma Cloud supports Docker Engine, CRI-O, and cri-containerd.

**Deploying Console**

Irrespective of your cluster's underlying container runtime, you can install Console using the standard install procedure. Console doesn't interface with other containers, so it doesn't need to know which container runtime interface is being used.

**Deploying Defender DaemonSets**

When generating the YAML file to deploy the Defender DaemonSet, a toggle lets you select your runtime environment. Since Defenders need to have a view of other containers, this option is necessary to guide the communication. By default the toggle is off Prisma Cloud uses Docker Engine. When the toggle is on, Prisma Cloud generates the proper yaml for the CRI Kubernetes environment.

> If you use containerd on GKE, and you install Defender without the CRI switch, everything will appear to work properly, but you'll have no images or container scan reports in Monitor > Vulnerability and Monitor > Compliance pages and you'll have no runtime models in Monitor > Runtime. This happens because the Google Container Optimized Operating system (GCOOS) nodes have Docker Engine installed, but Kubernetes doesn’t use it. Defender thinks everything is OK because all of the integrations succeed, but the underlying runtime is actually different.

7. Deploy Defenders with SELinux Policy

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Defenders as privileged (required for AppArmor compatibility)</td>
<td>On</td>
</tr>
<tr>
<td>Nodes use Container Runtime Interface (CRI), not Docker</td>
<td>Off</td>
</tr>
<tr>
<td>Nodes runs inside containerized environment</td>
<td>Off</td>
</tr>
</tbody>
</table>

If you're deploying Defender DaemonSets with twistcli, use the --cri option to use to specify the runtime interface. By default (no flag), twistcli generates a configuration that uses Docker Engine. With the --cri flag, twistcli generates a configuration that uses CRI.

```
$ <PLATFORM>/twistcli defender export kubernetes \
   --cri \
   --address https://yourconsole.example.com:8083 \
   --user <ADMIN_USER> \
   --cluster-address yourconsole.example.com
```

When generating YAML from Console or twistcli, there is a simple change to the yaml file as seen below. In this abbreviated version DEFENDER_TYPE:daemonset will use the Docker interface.

```
...spec:
  template:
    metadata:
```
labels:
  app: twistlock-defender
spec:
  serviceAccountName: twistlock-service
  restartPolicy: Always
  containers:
  - name: twistlock-defender-19-03-321
    image: registry-auth.twistlock.com/tw_<token>/twistlock/defender:defender_19_03_321
    volumeMounts:
      - name: host-root
        mountPath: "/host"
      - name: data-folder
        mountPath: "/var/lib/twistlock"
    env:
      - name: WS_ADDRESS
        value: wss://yourconsole.example.com:8084
      - name: DEFENDER_TYPE
        value: daemonset
      - name: DEFENDER_LISTENER_TYPE
        value: "none"
...
• Pass the `--helm` option to `twistcli` to generate a Helm chart. Other options passed to `twistcli` configure the chart.
• Deploy Console and Defender with `helm install` rather than `kubectl create`.

**STEP 1 | Download the Twistlock software.**

**STEP 2 | Create a Console Helm chart.**

```
$ <PLATFORM>/twistcli console export kubernetes \
   --service-type LoadBalancer \ 
   --helm
```

**STEP 3 | Install Console.**

```
$ helm install \ 
   --namespace twistlock \ 
   --name twistlock-console \ 
   ./twistlock-console-helm.tar.gz
```

**STEP 4 | Configure Console.**

**STEP 5 | Create a Defender DaemonSet Helm chart.**

```
$ <PLATFORM>/twistcli defender export kubernetes \
   --address https://yourconsole.example.com:8083 \ 
   --helm \
   --user <ADMIN_USER> \ 
   --cluster-address twistlock-console
```

**STEP 6 | Install Defender.**

```
$ helm install \ 
   --namespace twistlock \ 
   --name twistlock-defender-ds \ 
   ./twistlock-defender-helm.tar.gz
```

**Alibaba Cloud Container Service for Kubernetes (ACK)**

*Alibaba Cloud Container Service for Kubernetes (ACK)* is a managed Kubernetes service. Use the standard Kubernetes install procedure to deploy Prisma Cloud to Alibaba ACK, but specify an Alibaba Cloud-specific `StorageClass` when configuring the deployment.

This procedure shows you how to use Helm charts to install Prisma Cloud, but all other install methods are supported.

**Prerequisites:**

• You have provisioned an ACK cluster.

**STEP 1 | Go to Releases, and copy the link to current recommended release.**

**STEP 2 | Download the release tarball to the system where you administer your cluster (where you run your `kubectl` commands).**

```
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```
STEP 3 | Unpack the Prisma Cloud release tarball.

```bash
$ mkdir twistlock
$ tar xvzf twistlock_<VERSION>.tar.gz -C prisma_cloud/
```

STEP 4 | Create a Helm chart for Prisma Cloud Console.

```bash
$ <PLATFORM>/twistcli console export kubernetes \
   --storage-class alicloud-disk-available \
   --service-type LoadBalancer \
   --helm
```

STEP 5 | Install Console.

```bash
$ helm install \
   --namespace twistlock \
   --name twistlock-console \
   ./twistlock-console-helm.tar.gz
```

STEP 6 | Change the PersistentVolumeClaim's reclaimPolicy.

```bash
$ kubectl get pv 
$ kubectl patch pv <pvc-name> -p '{"spec": \
   "%persistentVolumeReclaimPolicy": "Retain"}"'
```

STEP 7 | Get the public endpoint address for Console.

When the service is fully up, the LoadBalancer's IP address is shown.

```bash
$ kubectl get service -w -n twistlock
```

STEP 8 | Open a browser window, and navigate to Console.

By default, Console is served on HTTPS on port 8083 of the LoadBalancer:


STEP 9 | Continue with the rest of the install here.

Amazon Elastic Kubernetes Service (EKS)

Amazon Kubernetes Service (EKS) lets you deploy Kubernetes clusters on demand. Use our standard Kubernetes install method to deploy Prisma Cloud to EKS. The only difference between the EKS and standard Kubernetes install methods is:

- EKS with Kubernetes 1.10 — Create a storage class that utilizes Amazon Elastic Block Storage (EBS), and then specify the storageClassName when generating the Prisma Cloud Console deployment file.
- EKS with Kubernetes 1.11+ — You only need to specify the storageClassName when generating the Prisma Cloud Console deployment file. The gp2 storage class already exists.

For more information about Amazon EKS storage classes, see the Amazon EKS User Guide.

Prerequisites:

- You have deployed an Amazon EKS cluster.
- You have downloaded the Prisma Cloud software.
STEP 1 | For EKS with Kubernetes 1.10, define a storage class named gp2 that uses the Amazon EBS gp2 volume type. Create a file name `gp2-storage-class.yaml`, and enter the following YAML.

For EKS with Kubernetes 1.11+, skip this step.

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp2
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ext4
```

STEP 2 | For EKS with Kubernetes 1.10, create the storage class.

For EKS with Kubernetes 1.11+, skip this step.

```bash
$ kubectl create -f gp2-storage-class.yaml
```

STEP 3 | Generate the Prisma Cloud Console deployment file (for all versions).

```bash
$ twistcli console export kubernetes \\
  --service-type LoadBalancer \\
  --storage-class gp2
```

STEP 4 | Deploy Console.

```bash
$ kubectl create -f twistlock_console.yaml
```

STEP 5 | Wait for the service to come up completely.

```bash
$ kubectl get service -w -n twistlock
```

STEP 6 | Continue with the rest of the install here.

### Azure Kubernetes Service (AKS)

Use the following procedure to install Prisma Cloud in an AKS cluster. This setup uses dynamic PersistentVolumeClaim provisioning using Premium Azure Disk. When creating your Kubernetes cluster, be sure to specify a VM size that supports premium storage.

> **Prisma Cloud doesn’t support Azure Files as a storage class for persistent volumes. Use Azure Disks instead.**

**Prerequisites:**

- You have deployed an Azure Container Service (AKS) cluster. Use the `--node-vm-size` parameter to specify a VM size that supports Premium Azure Disks.
- You have installed Azure CLI 2.0.22 or later.
- You have downloaded the Prisma Cloud software.
STEP 1 | Use `twistcli` to generate the Prisma Cloud Console YAML configuration file, where `<PLATFORM>` can be linux or osx. Set the storage class to Premium Azure Disk.

```
$ <PLATFORM>/twistcli console export kubernetes \\
   --storage-class managed-premium \\
   --service-type LoadBalancer
```

STEP 2 | Deploy the Prisma Cloud Console in the Azure Kubernetes Service cluster.

```
$ kubectl create -f ./twistlock_console.yaml
```

STEP 3 | Wait for the service to come up completely.

```
$ kubectl get service -w -n twistlock
```

STEP 4 | Change the PersistentVolumeClaim's reclaimPolicy.

```
$ kubectl get pv
$ kubectl patch pv <pvc-name> -p '{"spec": \\
   {"persistentVolumeReclaimPolicy":"Retain"}}'
```

STEP 5 | Continue with the rest of the install here.

**Azure Container Service (ACS) with Kubernetes**

Use the following procedure to install Prisma Cloud in an ACS Kubernetes cluster.

> Microsoft will retire ACS as a standalone service on January 31, 2020.

**Prerequisites:**

- You have deployed an Azure Container Service with Kubernetes cluster.
- You have installed Azure CLI 2.0.22 or later on a Linux system.
- You have downloaded the Prisma Cloud software.

STEP 1 | Create a persistent volume for your Kubernetes cluster. ACS uses Azure classic disks for the persistent volume. Within the same Resource Group as the ACS instance, create a classic storage group.

STEP 2 | On a Windows based system use Disk Manager to create an unformatted, 100GB Virtual Hard Disk (VHD).

STEP 3 | Use Azure Storage Explorer to upload the VHD to the classic storage group.

STEP 4 | Make sure the disk is 'released' from a 'lease'.

STEP 5 | On your Linux host with Azure CLI installed, attach to your ACS Kubernetes Master.

```
$ az acs kubernetes get-credentials --resource-group pfoxacs --name pfox-acs
```
Merged "pfoxacsmgmt" as current context in /Users/paulfox/.kube/config

$ kubectl config use-context pfoxacsmgmt

**STEP 6 | Confirm connectivity to the ACS Kubernetes cluster.**

```bash
$ kubectl get nodes
NAME                    STATUS    ROLES     AGE       VERSION
k8s-agent-e32fd1a6-0    Ready     agent     4m        v1.7.7
k8s-agent-e32fd1a6-1    Ready     agent     5m        v1.7.7
k8s-master-e32fd1a6-0   Ready     master    4m        v1.7.7
```

**STEP 7 | Create a file named **persistent-volume.yaml**, and open it for editing.**

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: twistlock-console
  labels:
    app: twistlock-console
  annotations:
    volume.beta.kubernetes.io/storage-class: default
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteOnce
  azureDisk:
    diskName: pfox-classic-tl-console.vhd
    diskURI: https://pfoxacs.blob.core.windows.net/twistlock-console/pfox-classic-tl-console.vhd
    cachingMode: ReadWrite
    fsType: ext4
    readOnly: false
```

- **diskName**

  Name of the persistent disk created in the previous steps.

- **labels**

  Label for the persistent volume.

- **diskURI**

  Azure subscription path to the disk created in the previous steps.

**STEP 8 | Create the persistent volume:**

```bash
$ kubectl create -f ./persistent-volume.yaml
```

**STEP 9 | Generate the Console YAML configuration file:**

```bash
$ /linux/twistcli console export kubernetes \
   --persistent-volume-labels app:twistlock-console \
   --storage-class default
```

- **--persistent-volume-labels**

  `app:twistlock-console` label defined in the persistent-volume.yaml.
• **--storage-class**

  `default` must match the storage class of the Azure Disk.

**STEP 10 |** Deploy the Prisma Cloud Console in your cluster.

```bash
$ kubectl create -f ./twistlock-console.yaml
```

**STEP 11 |** Wait for the service to come up completely.

```bash
$ kubectl get service -w -n twistlock
```

**STEP 12 |** Continue with the rest of the install here.

### DC/OS Kubernetes

Kubernetes on DC/OS uses nested virtualization, where K8S nodes are actually privileged containers. This abstraction creates a mismatch between the host PID namespace Defender needs to see, and the PID namespace it actually sees.

When deploying Prisma Cloud on DC/OS Kubernetes, use the normal install flow for Console.

For installing Defender, pass the `--containerized-host` flag to `twistcli` when generating the DaemonSet deployment file. If you're generating the DaemonSet deployment file from the Console UI, set the Nodes runs inside containerized environment option to On.

```bash
$ <PLATFORM>/twistcli defender export kubernetes \ 
  --address https://yourconsole.example.com:8083 \ 
  --user <ADMIN_USER> \ 
  --cluster-address twistlock-console \ 
  --containerized-host
```

### Google Kubernetes Engine (GKE)

To install Prisma Cloud on Google Kubernetes Engine (GKE), use the standard Kubernetes install flow. Before getting started, create a ClusterRoleBinding, which grants the permissions required to create the Defender DaemonSet.

The Google Cloud Platform (GCP) service account that you use to create the Prisma Cloud Console resources, including Deployment controller and PersistentVolumeClaim, must have at least the Kubernetes Engine Developer role to be successful.

The GCP service account that you use to create the Defender resources, including DaemonSet, must have the Kubernetes cluster-admin role. If you try to create the Defender resources from a service account without this cluster-specific role, it will fail because the GCP Kubernetes Engine Developer role doesn't grant the developer sufficient permissions to create a ClusterRole (one of the Defender resources). You'll need to use an account with the GCP Kubernetes Engine Admin role to bind the Kubernetes cluster-admin role to your Kubernetes developer's service account.

It's probably best to create the ClusterRoleBinding before turning the cluster over any user (typically DevOps) tasked with managing and maintaining Prisma Cloud.

>`Run the command in the following procedure on ANY service account that attempts to apply the Defender Daemonset YAML or Helm chart, even if that service account already has`
elevated permissions with the GCP Kubernetes Engine Admin role. Otherwise, you'll get an error.

The following procedure uses a service account named your-dev-user@your-org.iam.gserviceaccount.com that has the GCP Kubernetes Engine Developer role. You'll also need access to a more privileged GCP account that has the Kubernetes Engine Admin role to create the ClusterRoleBinding in your cluster.

Prerequisites:
- You have deployed a GKE cluster.
- You have a Google Cloud Platform (GCP) service account with the Kubernetes Engine Developer role.
- You have access to a GCP account with at least the Kubernetes Engine Admin role.

STEP 1 | With the service account that has the GCP Kubernetes Engine Admin role set as the active account, run:

```bash
$ kubectl create clusterrolebinding your-dev-user-cluster-admin-binding \
    --clusterrole=cluster-admin \
    --user=your-dev-user@your-org.iam.gserviceaccount.com
```

STEP 2 | With the Kubernetes Engine Developer service account, continue with the standard Kubernetes install procedure for Prisma Cloud Console and Defenders, starting here.

IBM Kubernetes Service (IKS)

Use the following procedure to install Prisma Cloud in an IKS cluster. IKS uses dynamic PersistentVolumeClaim provisioning (ibmc-file-bronze is the default StorageClass) as well as automatic LoadBalancer configuration for the Prisma Cloud Console. You can optionally specify a StorageClass for premium file or block storage options. Use a retain storage class (not default) to ensure your storage is not destroyed even if you delete the PVC.

When installing Defenders, take note of the the IKS Kubernetes version. IKS Kubernetes version 1.10 uses Docker, and 1.11+ uses containerd as the container runtime. If using containerd, pass the --cri flag to twistcli (or enable the CRI option in the Console UI) when generating the Defender YAML or Helm chart.

STEP 1 | Use twistcli to generate the Prisma Cloud Console YAML configuration file, where <PLATFORM> can be linux or osx. Optionally set the storage class to premium storage class. For IKS with Kubernetes 1.10, use our standard Kubernetes instructions. Here is an example with a premium StorageClass with the retain option.

```bash
$ <PLATFORM>/twistcli console export kubernetes \
    --storage-class ibmc-file-retain-silver \
    --service-type LoadBalancer
```

STEP 2 | Deploy the Prisma Cloud Console in the IBM Kubernetes Service cluster.

```bash
$ kubectl create -f ./twistlock_console.yaml
```

STEP 3 | Wait for the service to come up completely.

```bash
$ kubectl get service -w -n twistlock
```
STEP 4 | Continue with the rest of the install here.

Redeploying Defenders

If Prisma Cloud Console is redeployed, the client and server certificates change. Redeploy your Defenders so that they can connect to the new Console without certificate issues. First, generate a new DaemonSet YAML configuration file with `twistcli`:

```
$ <PLATFORM>/twistcli defender export kubernetes \
    --address https://yourconsole.example.com:8083 \
    --user <ADMIN_USER> \
    --cluster-address twistlock-console
```

Then apply the changes to your Defender pods. The `kubectl apply` command lets you make in-place updates to resources.

```
$ kubectl apply -f defender.yaml
```

Troubleshooting

**RBAC issues**

If RBAC is enabled in your cluster, you might get the following error when trying to create a Defender DaemonSet.

```
Error creating: pods "twistlock-defender-ds-" is forbidden: unable to validate against any pod security policy ..Privileged containers are not allowed
```

If you get this error, then you must create a Role and RoleBinding so that Defender can run with the privileges it needs. Create a Role and RoleBinding for the twistlock namespace. You can use the following example Role and RoleBinding:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: twistlock-role
  namespace: twistlock
rules:
- apiGroups:
  - extensions
    resourceNames:
    - privileged
    resources:
    - podsecuritypolicies
    verbs:
    - use

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: twistlock-rolebinding
  namespace: twistlock
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: twistlock-role
```

Defender install issues in GKE

If you see the following error when trying to create the Defender DaemonSet, you’ve probably tried to create the Defender resources from a service account that has the GCP Kubernetes Engine Developer role. To fix the issue, grant the proper cluster role to the service account.

Error from server (Forbidden): error when creating "daemonset.yaml":
clusterroles.rbac.authorization.k8s.io is forbidden: User "your-dev-user@your-org.iam.gserviceaccount.com" cannot create clusterroles.rbac.authorization.k8s.io at the cluster scope: Required "container.clusterRoles.create" permission.

Error from server (Forbidden): error when creating "daemonset.yaml":
clusterrolebindings.rbac.authorization.k8s.io is forbidden: User "your-dev-user@your-org.iam.gserviceaccount.com" cannot create clusterrolebindings.rbac.authorization.k8s.io at the cluster scope: Required "container.clusterRoleBindings.create" permission.

If you see the following error when trying to create the Defender DaemonSet, you’ve probably tried to create the Defender resources from a service account with the Kubernetes Engine Admin role. To fix the issue, grant the proper cluster role to the service account.

Error from server (Forbidden): error when creating "daemonset.yaml":
clusterroles.rbac.authorization.k8s.io "twistlock-view" is forbidden: attempt to grant extra privileges: 
[ownerrules=[

Uninstall

To uninstall Prisma Cloud, delete the twistlock namespace. Deleting a namespace deletes everything under it.

Be careful when you delete the twistlock namespace. Console stores its data in the persistent volume (PV). If your PV is in the twistlock namespace (it is by default), the PV will also be deleted. You’ll lose all your data, and you won’t be able to restore Console.

Delete the twistlock namespace.

$ kubectl delete namespaces twistlock
OpenShift

Prisma Cloud supports OpenShift v3.9 and later.

Prisma Cloud Console is deployed as a ReplicationController, which ensures it's always running. Prisma Cloud Defenders are deployed as a DaemonSet, which ensures that an instance of Defender runs on every node in the cluster. You can run Defenders on OpenShift master and infrastructure nodes using node selectors.

The Prisma Cloud Console and Defender container images can be stored either in the internal OpenShift registry or your own Docker v2 compliant registry. Alternatively, you can configure your deployments to pull images from Prisma Cloud’s cloud registry.

This guide shows you how to generate deployment YAML files for both Console and Defender, and then deploy them to your OpenShift cluster with the oc client.

Cluster context

Prisma Cloud can segment your environment by cluster. For example, you might have three clusters: test, staging, and production. The cluster pivot in Prisma Cloud lets you inspect resources and administer security policy on a per-cluster basis.
Defenders in each DaemonSet are responsible for reporting which resources belong to which cluster. When deploying a Defender DaemonSet, Prisma Cloud tries to determine the cluster name through introspection. First, it tries to retrieve the cluster name from the cloud provider. As a fallback, it tries to retrieve the name from the corresponding kubeconfig file saved in the credentials store. Finally, you can override these mechanisms by manually specifying a cluster name when deploying your Defender DaemonSet.

Both the Prisma Cloud UI and twistcli tool accept an option for manually specifying a cluster name. Let Prisma Cloud automatically detect the name for provider-managed clusters. Manually specify names for self-managed clusters, such as those built with kops.

Radar lets you explore your environment cluster-by-cluster. You can also create stored filters (also known as collections) based on cluster names. Finally, you can scope policy by cluster. Vulnerability and compliance rules for container images and hosts can all be scoped by cluster name.

There are some things to consider when manually naming clusters:

- If you specify the same name for two or more clusters, they’re treated as a single cluster.
- For GCP, if you have clusters with the same name in different projects, they’re treated as a single cluster. Consider manually specifying a different name for each cluster.
Manually specifying names isn’t supported in Manage > Defenders > Manage > DaemonSet. This page lets you deploy and manage DaemonSets directly from the Prisma Cloud UI. For this deployment flow, cluster names are retrieved from the cloud provider or the supplied kubeconfig only.

Preflight checklist

To ensure that your installation goes smoothly, work through the following checklist and validate that all requirements are met.

Minimum system requirements

Validate that the components in your environment (nodes, host operating systems, orchestrator) meet the specs in System requirements.

For OpenShift installs, we recommend using the overlay or overlay2 storage drivers due to a known issue in RHEL. For more information, see https://bugzilla.redhat.com/show_bug.cgi?id=1518519.

Permissions

Validate that you have permission to:

- Push to a private docker registry. For most OpenShift setups, the registry runs inside the cluster as a service. You must be able to authenticate with your registry with docker login.
- Pull images from your registry. This might require the creation of a docker-registry secret.
- Have the correct role bindings to pull and push to the registry. For more information, see Accessing the Registry.
- Create and delete projects in your cluster. For OpenShift installations, a project is created when you run oc new-project.
- Run oc create commands.

Internal cluster network communication

TCP: 8083, 8084

External cluster network communication

TCP: 443

The Prisma Cloud Console connects to the Prisma Cloud Intelligence Stream (https://intelligence.twistlock.com) on TCP port 443 for vulnerability updates. If your Console is unable to contact the Prisma Cloud Intelligence Stream, follow the guidance for offline environments.

Install Prisma Cloud

Use twistcli to install the Prisma Cloud Console and Defenders. The twistcli utility is included with every release. After completing this procedure, both Prisma Cloud Console and Prisma Cloud Defenders will be running in your OpenShift cluster.

Download the Prisma Cloud software

Download the latest Prisma Cloud release to any system where the OpenShift oc client is installed.

STEP 1 | Go to Releases, and copy the link to current recommended release.
STEP 2 | Download the release tarball to your cluster controller.

```bash
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```

STEP 3 | Unpack the release tarball.

```bash
$ mkdir twistlock
$ tar xvzf twistlock_<VERSION>.tar.gz -C twistlock/
```

Create an OpenShift project for Prisma Cloud

Create a project named *twistlock*.

Login to the OpenShift cluster and create the *twistlock* project:

```bash
$ oc new-project twistlock
```

(Optional) Push the Prisma Cloud images to a private registry

When Prisma Cloud is deployed to your cluster, the images are retrieved from a registry. You have a number of options for storing the Prisma Cloud Console and Defender images:

- OpenShift internal registry.
- Private Docker v2 registry. You must create a docker-secret to authenticate with the registry.

Alternatively, you can pull the images from the Prisma Cloud cloud registry at deployment time. Your cluster nodes must be able to connect to the Prisma Cloud cloud registry (registry-auth.twistlock.com) with TLS on TCP port 443.

This guides shows you how to use both the OpenShift internal registry and the Prisma Cloud cloud registry. If you're going to use the Prisma Cloud cloud registry, you can skip this section. Otherwise, this procedure shows you how to pull, tag, and upload the Prisma Cloud images to the OpenShift internal registry's *twistlock* imageStream.

STEP 1 | Determine the endpoint for your OpenShift internal registry. Use either the internal registry’s service name or cluster IP.

```bash
$ oc get svc -n default
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)  AGE
docker-registry    ClusterIP   172.30.163.181   <none>        5000/TCP 88d
```

STEP 2 | Pull the images from the Prisma Cloud cloud registry using your access token. The major, minor, and patch numerals in the `<VERSION>` string are separated with an underscore. For example, 18.11.128 would be `18_11_128`.

```bash
$ docker pull \
  registry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/
defender:defender_<VERSION>
$ docker pull \
  registry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/
  console:console_<VERSION>
```
STEP 3 | Tag the images for the OpenShift internal registry.

$ docker tag \
registry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/defender:defender_<VERSION> \
172.30.163.181:5000/twistlock/private:defender_<VERSION>

$ docker tag \
registry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/console:console_<VERSION> \
172.30.163.181:5000/twistlock/private:console_<VERSION>

STEP 4 | Push the images to the twistlock project’s imageStream.

$ docker push 172.30.163.181:5000/twistlock/private:defender_<VERSION>
$ docker push 172.30.163.181:5000/twistlock/private:console_<VERSION>

Install Console

Use the twistcli tool to generate the Prisma Cloud Console deployment YAML. The twistcli tool is bundled with the release tarball. There are versions for Linux, macOS, and Windows.

The twistcli tool generates YAML for a ReplicationController, and other service configurations, such as a PersistentVolumeClaim, SecurityContextConstraints, and so on. Run the twistcli command with the --help flag for additional details about the command and supported flags.

You can optionally customize twistlock.cfg to enable additional features, such as custom compliance SCAP scanning. Then run twistcli from the root of the extracted release tarball.

Prisma Cloud Console uses a PersistentVolumeClaim to store data. There are two ways to provision storage for Console:

- **Dynamic provisioning:** Allocate storage for Console on-demand at deployment time. When generating the Console deployment YAML files with twistcli, specify the name of the storage class with the --storage-class flag. Most customers use dynamic provisioning.

- **Manual provisioning:** Pre-provision a persistent volume for Console, then specify its label when generating the Console deployment YAML files. OpenShift uses NFS mounts for the backend infrastructure components (e.g. registry, logging, etc.). The NFS server is typically one of the master nodes. Guidance for creating an NFS backed PersistentVolume can be found here. Also see Appendix: NFS PersistentVolume example.

STEP 1 | Generate a deployment YAML file for Console. A number of command variations are provided. Use them as a basis for constructing your own working command.

Prisma Cloud Console + dynamically provisioned PersistentVolume + image pulled from the OpenShift internal registry.

$ <PLATFORM>/twistcli console export openshift \
--storage-class "<STORAGE-CLASS-NAME>" \
--image-name "172.30.163.181:5000/twistlock/private:console_<VERSION>" \
--service-type "ClusterIP"

Prisma Cloud Console + manually provisioned PersistentVolume + image pulled from the OpenShift internal registry. Using the NFS backed PersistentVolume described in Appendix: NFS PersistentVolume example.
example, pass the label to the `--persistent-volume-labels` flag to specify the PersistentVolume to which the PersistentVolumeClaim will bind.

```
$ <PLATFORM>/twistcli console export openshift \
   --persistent-volume-labels "app-volume=twistlock-console" \
   --image-name "172.30.163.181:5000/twistlock/private:console_<VERSION>" \
   --service-type "ClusterIP"
```

Prisma Cloud Console + manually provisioned PersistentVolume + image pulled from the Prisma Cloud cloud registry. If you omit the `--image-name` flag, the Prisma Cloud cloud registry is used by default, and you are prompted for your access token.

```
$ <PLATFORM>/twistcli console export openshift \
   --persistent-volume-labels "app-volume=twistlock-console" \
   --service-type "ClusterIP"
```

### STEP 2 | Deploy Console.

```
$ oc create -f ./twistlock_console.yaml
```

You can safely ignore the error that says the twistlock project already exists.

### Create an external route to Console

Create an external route to Console so that you can access the web UI and API.

#### STEP 1 | From the OpenShift web interface, go to the **twistlock** project.

#### STEP 2 | Go to **Application > Routes**.

#### STEP 3 | Select **Create Route**.

#### STEP 4 | Enter a name for the route, such as **twistlock-console**.

#### STEP 5 | Hostname = URL used to access the Console, e.g. **twistlock-console.apps.ose.example.com**

#### STEP 6 | Path = /

#### STEP 7 | Service = **twistlock-console**

#### STEP 8 | Target Port = 8083 → 8083

#### STEP 9 | Select the **Security > Secure Route** radio button.

#### STEP 10 | TLS Termination = Passthrough (if using 8083)

If you plan to issue a **custom certificate for the Prisma Cloud Console** that is trusted and will allow the TLS establishment with the Prisma Cloud Console, then Select Passthrough TLS for TCP port 8083.

#### STEP 11 | Insecure Traffic = **Redirect**

#### STEP 12 | Click **Create**.
Configure Console

Create your first admin user, enter your license key, and configure Console’s certificate so that Defenders can establish a secure connection to it.

**STEP 1** | In a web browser, navigate to the external route you configured for Console, e.g. https://twistlock-console.apps.ose.example.com.

**STEP 2** | Create your first admin account.

**STEP 3** | Enter your license key.

**STEP 4** | Add a SubjectAlternativeName to Console’s certificate to allow Defenders to establish a secure connection with Console.

Use either Console’s service name, `twistlock-console` or `twistlock-console.twistlock.svc`, or Console’s cluster IP.

```bash
$ oc get svc -n twistlock
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>twistlock-console</td>
<td>LoadBalancer</td>
<td>172.30.41.62</td>
<td>172.29.61.32,172.29.61.32 8084:3184...</td>
</tr>
</tbody>
</table>

1. Go to Manage > Defenders > Names.
2. Click Add SAN and enter Console’s service name.
3. Click Add SAN and enter Console’s cluster IP.

Install Defender

Prisma Cloud Defenders run as containers on the nodes in your OpenShift cluster. They are deployed as a DaemonSet. Use the `twistcli` tool to generate the DaemonSet deployment YAML. The command has the following basic structure. It creates a YAML file named `defender.yaml` in the working directory.

```bash
$ <PLATFORM>/twistcli defender export openshift \
```
The command connects to Console's API, specified in `--address`, to generate the Defender DaemonSet YAML config file. The location where you run twistcli (inside or outside the cluster) dictates which Console address should be supplied.

The `--cluster-address` flag specifies the address Defender uses to connect to Console. For Defenders deployed inside the cluster, specify Prisma Cloud Console's service name, twistlock-console or twistlock-console.twistlock.svc, or cluster IP address. For Defenders deployed outside the cluster, specify either Console's external address, which is exposed by your external route.

If SELinux is enabled on the OpenShift nodes, pass the `--selinux-enabled` argument to twistcli.

For managed clusters, Prisma Cloud automatically gets the cluster name from the cloud provider. To override the the cloud provider’s cluster name, use the `--cluster` option. For self-managed clusters, manually specify a cluster name with the `--cluster` option.

**STEP 1** Generate the Defender DaemonSet YAML. A number of command variations are provided. Use them as a basis for constructing your own working command.

**Outside the OpenShift cluster + pull the Defender image from the Prisma Cloud cloud registry.** Use the OpenShift external route for your Prisma Cloud Console, `--address https://twistlock-console.apps.ose.example.com`. Designate Prisma Cloud’s cloud registry by omitting the `--image-name` flag.

```bash
$ <PLATFORM>/twistcli defender export openshift \
  --address https://twistlock-console.apps.ose.example.com \
  --cluster-address 172.30.41.62 \
  --selinux-enabled
```

**Outside the OpenShift cluster + pull the Defender image from the OpenShift internal registry.** Use the `--image-name` flag to designate an image from the OpenShift internal registry.

```bash
$ <PLATFORM>/twistcli defender export openshift \
  --address https://twistlock-console.apps.ose.example.com \
  --cluster-address 172.30.41.62 \
  --selinux-enabled \
  --image-name 172.30.163.181:5000/twistlock/private:defender_<VERSION>
```

**Inside the OpenShift cluster + pull the Defender image from the Prisma Cloud cloud registry.** When generating the Defender DaemonSet YAML with twistcli from a node inside the cluster, use Console’s service name (twistlock-console) or cluster IP in the `--cluster-address` flag. This flag specifies the endpoint for the Prisma Cloud Compute API and must include the port number.

```bash
$ <PLATFORM>/twistcli defender export openshift \
  --address https://172.30.41.62:8083 \
  --cluster-address 172.30.41.62 \
  --selinux-enabled
```

**Inside the OpenShift cluster + pull the Defender image from the OpenShift internal registry.** Use the `--image-name` flag to designate an image in the OpenShift internal registry.

```bash
$ <PLATFORM>/twistcli defender export openshift \
  --address https://172.30.41.62:8083 \
  --cluster-address 172.30.41.62 \
  --selinux-enabled \
  --image-name 172.30.163.181:5000/twistlock/private:defender_<VERSION>
```
**STEP 2** | Deploy the Defender DaemonSet.

```bash
$ oc create -f ./defender.yaml
```

**STEP 3** | Confirm the Defenders were deployed.

1. In Prisma Cloud Console, go to **Manage > Defenders > Manage** to see a list of deployed Defenders.

A Defender is installed on each host Twistlock protects.

<table>
<thead>
<tr>
<th>Version</th>
<th>Type</th>
<th>Listener Type</th>
<th>Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.127</td>
<td>Daemon Set on Linux</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>2.5.127</td>
<td>Daemon Set on Linux</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>2.5.127</td>
<td>Daemon Set on Linux</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

2. In the OpenShift Web Console, go to the Prisma Cloud project’s monitoring window to see which pods are running.
3. Using the OpenShift CLI to see the DaemonSet pod count.

```
$ oc get ds -n twistlock
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>AVAILABLE</th>
<th>NODE SELECTOR</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>READY</th>
<th>UP-TO-DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>twistlock-defender-ds</td>
<td>(none)</td>
<td>&lt;none&gt;</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>twistlock-defender-ds-2thrx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>twistlock-defender-ds-69p7m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>twistlock-defender-ds-mzwpg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>twistlock-console-vcf6k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The desired and current pod counts do not match. This is a job for the nodeSelector.

Control Defender deployments with NodeSelector

You can deploy Defenders to all nodes in an OpenShift cluster (master, infra, compute). Depending upon the nodeSelector configuration, Prisma Cloud Defenders may not get deployed to all nodes. Adjust the guidance in the following procedure according to your organization’s deployment strategy.

**STEP 1** Review the following OpenShift configuration settings.
1. The OpenShift master nodeSelector configuration can be found in `/etc/origin/master/master-config.yaml`. Look for any nodeSelector and nodeSelectorLabelBlacklist settings.

   ```yaml
   defaultNodeSelector: compute=true
   ``

2. Prisma Cloud project - The nodeSelector can be defined at the project level.

   ```bash
   $ oc describe project twistlock
   Name:                   twistlock
   Created:                10 days ago
   Labels:                 <none>
   Annotations:            openshift.io/description=
                           openshift.io/display-name=
                           openshift.io/node-selector=node-role.kubernetes.io/compute=true
                           openshift.io/sa.scc.mcs=s0:c17,c9
                           openshift.io/sa.scc.supplemental-groups=1000290000/10000
   Display Name:           <none>
   Description:            <none>
   Status:                 Active
   Node Selector:          node-role.kubernetes.io/compute=true
   Quota:                  <none>
   Resource limits:        <none>
   ``

   In this example the Prisma Cloud project default nodeSelector instructs OpenShift to only deploy Defenders to the `node-role.kubernetes.io/compute=true` nodes.

   **STEP 2** | The following command removes the Node Selector value from the Prisma Cloud project.
   ```bash
   $ oc annotate namespace twistlock openshift.io/node-selector=""
   ```

   **STEP 3** | Add a `Deploy_Prisma Cloud: true` label to all nodes to which Defender should be deployed.
   ```bash
   $ oc label node ip-172-31-0-55.ec2.internal Deploy_Prisma Cloud=true
   $ oc describe node ip-172-31-0-55.ec2.internal
   Name:                   ip-172-31-0-55.ec2.internal
   Roles:                 compute
   Labels:               Deploy_Prisma Cloud=true
                           beta.kubernetes.io/arch=amd64
                           beta.kubernetes.io/os=linux
                           kubernetes.io/hostname=ip-172-31-0-55.ec2.internal
                           logging-infra-fluentd=true
                           node-role.kubernetes.io/compute=true
                           region=primary
   Annotations:           volumes.kubernetes.io/controller-managed-attach-detach=true
   CreationTimestamp:  Sun, 05 Aug 2018 05:40:10 +0000
   ```

   **STEP 4** | Set the nodeSelector in the Defender DaemonSet deployment YAML.
   ```yaml
   version: extensions/v1beta1
   kind: DaemonSet
   metadata:
     name: twistlock-defender-ds
     namespace: twistlock
   spec:
   ```
STEP 5 | Check the desired and current count for the Defender DaemonSet deployment.

```bash
$ oc get ds -n twistlock
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>READY</th>
<th>UP-TO-DATE</th>
<th>AVAILABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>twistlock-defender-ds</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Deploy_Prisma Cloud=true</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Install Prisma Cloud with Helm charts

You can use `twistcli` to create Helm charts for Prisma Cloud Console and Defender. Helm is a package manager for Kubernetes, and `chart` is the moniker for a Helm package.

Follow the main install flow, except:

- Pass the `--helm` option to `twistcli` to generate a Helm chart. Other options passed to `twistcli` configure the chart.
- Deploy Console and Defender with `helm install` rather than `oc create`.

To create and install a Console Helm chart that dynamically provisions its persistent volume and pulls the container image from the OpenShift internal registry:

```bash
$ <PLATFORM>/twistcli console export openshift \
  --storage-class "<STORAGE-CLASS-NAME>" \
  --image-name "172.30.163.181:5000/twistlock/private:console_<VERSION>" \
  --service-type "ClusterIP" \
  --helm

$ helm install --namespace=twistlock twistlock-console-helm.tar.gz
```

To create and install a Defender DaemonSet Helm chart that pulls the Defender image from the OpenShift internal registry:

```bash
$ <PLATFORM>/twistcli defender export openshift \
  --address https://twistlock-console.apps.ose.example.com \
  --cluster-address 172.30.41.62 \
  --selinux-enabled \
  --image-name 172.30.163.181:5000/twistlock/private:defender_<VERSION> \
  --helm

$ helm install --namespace=twistlock twistlock-defender-helm.tar.gz
```
OpenShift 4

Prisma Cloud Console Helm charts fail to install on OpenShift 4 clusters due to a Helm bug. If you generate a Helm chart, and try to install it in an OpenShift 4 cluster, you'll get the following error:

```
Error: unable to recognize "": no matches for kind "SecurityContextConstraints" in version "v1"
```

To work around the issue, modify the generated Helm chart.

**STEP 1 |** Generate a Console Helm chart.

```
$ <PLATFORM>/twistcli console export kubernetes \
   --service-type LoadBalancer \
   --helm
```

**STEP 2 |** Unpack the chart into a temporary directory.

```
$ mkdir helm 
$ tar xvzf twistlock-console-helm.tar.gz -C helm/
```

**STEP 3 |** Open `helm/twistlock-console/templates/securitycontextconstraints.yaml` for editing.

**STEP 4 |** Change `apiVersion` from `v1` to `security.openshift.io/v1`.

```yaml
{{- if .Values.openshift }}
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: twistlock-console
...
```

**STEP 5 |** Repack the Helm chart, and install it in your OpenShift 4 cluster.

```
$ cd helm/
$ tar cvzf twistlock-console-helm.tar.gz twistlock-console/
```

**Uninstall**

To uninstall Prisma Cloud, delete the `twistlock` project, then delete the Prisma Cloud PersistentVolume.

**STEP 1 |** Delete the `twistlock` Project

```
$ oc delete project twistlock
```

**STEP 2 |** Delete the `twistlock` PersistentVolume

```
$ oc delete pv twistlock
```

**Appendix: NFS PersistentVolume example**

Create an NFS mount for the Prisma Cloud Console's PV on the host that serves the NFS mounts.
STEP 1 | `mkdir /opt/twistlock_console`

STEP 2 | Check selinux: `sestatus`

STEP 3 | `chcon -R -t svirt_sandbox_file_t -l s0 /opt/twistlock_console`

STEP 4 | `sudo chown nfsnobody /opt/twistlock_console`

STEP 5 | `sudo chgrp nfsnobody /opt/twistlock_console`

STEP 6 | Check perms with: `ls -IZ /opt/twistlock_console` (drwxr-xr-x. nfsnobody nfsnobody system_u:object_r:svirt_sandbox_file_t:s0)

STEP 7 | Create `/etc/exports.d/twistlock.exports`

STEP 8 | In the `/etc/exports.d/twistlock.exports` add in line `/opt/twistlock_console *(rw,root_squash)`

STEP 9 | Restart nfs mount `sudo exportfs -ra`

STEP 10 | Confirm with `sudo exportfs -a`

STEP 11 | Get the IP address of the Master node that will be used in the PV (eth0, openshift uses 172. for node to node communication). Make sure TCP 2049 (NFS) is allowed between nodes.

STEP 12 | Create a PersistentVolume for Prisma Cloud Console.

The following example uses a label for the PersistentVolume and the volume and claim pre-binding features. The PersistentVolumeClaim uses the app-volume: twistlock-console label to bind to the PV. The volume and claim pre-binding claimref ensures that the PersistentVolume is not claimed by another PersistentVolumeClaim before Prisma Cloud Console is deployed.

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: twistlock
labels:
  app-volume: twistlock-console
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    path: /opt/twistlock_console
    server: 172.31.4.59
  persistentVolumeReclaimPolicy: Retain
claimRef:
  name: twistlock-console
  namespace: twistlock
```

**Appendix: Implementing SAML federation with a Prisma Cloud Console inside an OpenShift cluster**

When federating Prisma Cloud Console that is accessed through an OpenShift external route with a SAML v2.0 Identity Provider (IdP), the SAML authentication request's `AssertionConsumerServiceURL`
value must be modified. Prisma Cloud automatically generates the `AssertionConsumerServiceURL` value sent in a SAML authentication request based on Console’s configuration. When Console is accessed through an OpenShift external route, the URL for Console’s API endpoint is most likely not the same as the automatically generated `AssertionConsumerServiceURL`. Therefore, you must configure the `AssertionConsumerServiceURL` value that Prisma Cloud sends in the SAML authentication request.

**STEP 1 |** Log into Prisma Cloud Console.

**STEP 2 |** Go to Manage > Authentication > SAML.

**STEP 3 |** In **Console URL**, define the `AssertionConsumerServiceURL`.

   In this example, enter `https://twistlock-console.apps.ose.example.com`
VMware Tanzu Enterprise PKS

VMware Enterprise PKS lets you deploy Kubernetes clusters on demand. Use our standard Kubernetes install procedure to deploy Prisma Cloud to PKS. The only difference between PKS and standard Kubernetes is the location of the Docker socket. A single line change in the Prisma Cloud configuration file lets you specify the path to the Docker socket in PKS. From there, follow the normal Kubernetes install procedure.

Preflight checklist

To ensure that your installation goes smoothly, work through the following checklist and validate that all requirements are met.

**General**
- You have a valid Prisma Cloud license key and access token.

**Cluster**
- You have provisioned a PKS cluster that meets the minimum system requirements.
- Prisma Cloud Defender requires elevated privileges. Ensure that the following permissions are set in your PKS cluster:
  - Set Privileged Containers to true (enabled).
  - Set DenyEscalatingExec to false (disabled). After Prisma Cloud is installed, you can utilize it to deny other privileged containers from starting and deny escalation of privileges.
  - The nodes in your cluster can reach Prisma Cloud’s cloud registry (registry-auth.twistlock.com).

**Permissions**
- You can create and delete namespaces in your cluster.
- You can Run `kubectl create` commands.

**Firewalls and external IP addresses**

Validate that the following ports are open:

**Prisma Cloud Console:**
- Incoming: 8083, 8084
- Outgoing: 443, 53

**Prisma Cloud Defenders:**
- Incoming: None
- Outgoing: 8084

**Install Prisma Cloud**

Prepare your PKS environment, then use the standard procedure for installing Prisma Cloud on Kubernetes.

**Download the Prisma Cloud software**

Download the Prisma Cloud software to your cluster’s controller node.
STEP 1 | Go to Releases, and copy the link to current recommended release.

STEP 2 | Download the release tarball to your cluster controller.

```bash
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```

STEP 3 | Unpack the Prisma Cloud release tarball.

```bash
$ mkdir twistlock
$ tar xzvf twistlock_<VERSION>.tar.gz -C twistlock/
```

STEP 4 | Open `twistlock/twistlock.cfg` and set the path to the Docker socket.

```bash
DOCKER_SOCKET=${DOCKER_SOCKET:-/var/vcap/data/sys/run/docker/docker.sock}
```

STEP 5 | In `twistlock.cfg`, set RUN CONSOLE AS ROOT to true.

```bash
RUN_CONSOLE_AS_ROOT=${RUN_CONSOLE_AS_ROOT:-true}
```

**Install Console and Defenders**

Proceed with the standard instructions for installing Prisma Cloud on Kubernetes.
Docker Swarm

This procedure is optimized to get Prisma Cloud installed and set up in your Docker Swarm cluster quickly. There are many ways to install Prisma Cloud, but we recommend that you start with this procedure first. You can tweak the install procedure after you have validated that this install method works.

The Prisma Cloud install supports Docker Swarm using Swarm-native constructs. Deploy Console as a service so you can rely on Swarm to ensure Console is always available. Deploy Defender as a global service to guarantee that Defender is automatically deployed to every worker node with a simple one-time configuration.

Install Prisma Cloud

After completing this procedure, both Prisma Cloud Console and Prisma Cloud Defenders will run in your Swarm cluster. This setup uses a load balancer (HAProxy) and external persistent storage so that Console can failover and restart on any Swarm worker node.

If you don’t have external persistent storage, you can configure Console to use local storage, but you must pin Console to the node with the local storage. Console with local storage is not recommended for production-grade setups.

In this procedure, Prisma Cloud images are pulled from Prisma Cloud’s cloud registry.

Prisma Cloud doesn’t support deploying Defender as a global service when SELinux is enabled on your underlying hosts. Defender requires access to the Docker socket to monitor your environment and enforce your policies. SELinux blocks access to the Docker socket because it can be a serious security issue. Unfortunately, Swarm doesn’t provide a way for legitimate services to run with elevated privileges. None of the --security-opts, --privileged, or --cap-add flags are supported for Swarm services. As a work-around, install single Container Defenders on each individual node in your cluster.

Set up a load balancer

Swarm uses a routing mesh inside the cluster. When you deploy Prisma Cloud Console as a replicated service, Swarm’s routing mesh publishes Console’s ports on every node.

A load balancer is required to facilitate Defender-to-Console communication. Console is deployed on an overlay network, and Defenders are deployed in the host network namespace. Because Defenders aren’t connected to the overlay network, they cannot connect to the Virtual IP (VIP) address of the Prisma Cloud Console service. Prepare your load balancer so that traffic is distributed to all available Swarm worker nodes. The nodes use Swarm’s routing mesh to forward traffic to the worker node that runs Console. The following diagram shows the setup:
The following example HAProxy configuration has been tested in our labs. Use it as a starting point for your own configuration.

> Whichever load balancer you use, be sure it supports TCP passthrough. Otherwise, Defenders might not be able to connect to the Console.

```plaintext
global
    ...
    ca-base /etc/ssl/certs
crt-base /etc/ssl/private
    ...
    ssl-default-bind-ciphers ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH
           +AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS
    ssl-default-bind-options no-sslv3
tune.ssl.default-dh-param 2048
defaults
    ...
frontend https_front
    stats uri /haproxy/stats
default_backend https_back
```
bind *:8083 ssl crt /etc/ssl/private/haproxy.pem
backend https_back
  balance roundrobin
  server node1 IP-OF-YOUR-SWARMWORKER:8083 weight 1 maxconn 100 check ssl verify none
  server node2 IP-OF-YOUR-SWARMWORKER:8083 weight 1 maxconn 100 check ssl verify none
  server node3 IP-OF-YOUR-SWARMWORKER:8083 weight 1 maxconn 100 check ssl verify none
frontend defender_front
  stats uri /haproxy?stats
default_backend defender_back
  option tcplog
  mode tcp
  bind *:8084
backend defender_back
  balance roundrobin
  mode tcp
  option tcp-check
  server node1 IP-OF-YOUR-SWARMWORKER:8084 check
  server node2 IP-OF-YOUR-SWARMWORKER:8084 check
  server node3 IP-OF-YOUR-SWARMWORKER:8084 check

A couple of notes about the config file:

- Traffic is balanced across three Swarm nodes. Specify as many Swarm nodes as needed under backend http_back, backend https_back, and backend defender_back.
- The port binding 8083 uses HTTPS, so you must create a certificate in PEM format before applying the configuration. See bind *:8083 ssl crt /etc/ssl/private/haproxy.pem under frontend https_front. The cert in this configuration is stored in /etc/ssl/private/haproxy.pem. Use the linked instructions to create a certificate. We recommend creating a certificate that is signed by your trusted CA.

(Optional) Set up a DNS record

Simplify the configuration of your environment by setting up a DNS A Record that points to your load balancer. Then use the load balancer’s domain name to:

- Connect to Console’s HTTP or HTTPS web interface,
- Interface with Console’s API,
- Configure how Defender connects to Console.

Set up persistent storage

Install a volume driver that can create persistent volumes that can be accessed from any node in the cluster. Because Console can be scheduled on any node, it must be able to access its data and backup folders from wherever it runs.

You can use any available volume plugin, then specify the plugin driver with the --volume-driver option when installing Prisma Cloud Console with twistcli. Every node in your cluster must have the proper permissions to create persistent volumes.

This procedure describes how to use the Google Cloud Platform and NFSv4 volume drivers, but you can use any supported volume plugin.

Set up persistent storage on GCP

Set up the gce-docker volume plugin on each cluster node, then create data and backup volumes for Console.

STEP 1 | Verify that Swarm is enabled on all nodes, and that they are connected to a healthy master.
STEP 2 | Install the GCP volume plugin. Run the following command on each node.

```bash
$ docker run -d \
  -v /:/rootfs \
  -v /run/docker/plugins:/run/docker/plugins \
  -v /var/run/docker.sock:/var/run/docker.sock \
  --privileged \
  mcuadros/gce-docker
```

STEP 3 | Create persistent volumes to hold Console's data and backups.

```bash
$ docker volume create \
  --driver=gce \n  --name twistlock-console \n  -o SizeGb=90

$ docker volume create \
  --driver=gce \n  --name twistlock-backup \n  -o SizeGb=90
```

Set up persistent storage on NFSv4

Set up an NFS server, then create data and backup volumes for Console. The NFS server should run on a dedicated host that runs outside of the Swarm cluster.

Prisma Cloud Console uses MongoDB to store data. There are some mount options required when accessing a MongoDB database from an NFSv4 volume.

- `nolock` — Disables the NLM sideband protocol to lock files on the server.
- `noatime` — Disables the NFS server from updating the inodes access time.
- `bg` — Backgrounds a mount command so that it doesn't hang forever in the event that there is a problem connecting to the server.

STEP 1 | Install an NFSv4 server:

```bash
$ sudo apt install nfs-kernel-server
```

STEP 2 | Configure the server.

1. Open `/etc/exports` for editing.

```bash
$ sudo vim /etc/exports
```

2. Append the following line to the file.

```bash
/srv/home *(rw,sync,no_root_squash)
```

STEP 3 | Start the server.

```bash
$ sudo systemctl start nfs-kernel-server.service
```

STEP 4 | Mount all other nodes.

```bash
$ sudo mount -o nolock,bg,noatime <server-ip>:/srv/home /<local>/srv/home
```
STEP 5 | Ensure all permissions are granted to twistlock user (2674).

STEP 6 | Create NFS volumes to hold Console’s data and backups.

```
$ docker volume create --driver local \
  --opt type=nfs \
  --opt o=addr=<IP address of the NFS Server>,rw,nolock,noatime,bg \
  --opt device=/srv/home \
  twistlock-console

$ docker volume create --driver local \
  --opt type=nfs \
  --opt o=addr=<IP address of the NFS Server>,rw,nolock,noatime,bg \
  --opt device=/srv/home \
  twistlock-backup
```

**Install Console**

Install Console as a Docker Swarm service.

**Prerequisites:**
- All the components in your environment (nodes, host operating systems, orchestrator, etc) meet the hardware and version specs in System requirements.
- Your Swarm cluster is up and running.
- Your persistent storage is configured correctly.
- Your load balancer is configured correctly for ports 8083 (HTTPS) and 8084 (TCP).
- You created a DNS record that points to your load balancer.

STEP 1 | Go to Releases, and copy the link to current recommended release.

STEP 2 | Connect to your master node.

```
$ ssh <SWARM-MASTER>
```

STEP 3 | Retrieve the release tarball.

```
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```

STEP 4 | Unpack the Prisma Cloud release tarball.

```
$ mkdir twistlock
$ tar xzvf twistlock_<VERSION>.tar.gz -C twistlock/
$ cd twistlock
```

STEP 5 | Install Console into your Swarm using the *twistcli* utility.

If you are using GCP:

```
$ ./linux/twistcli console install swarm --volume-driver "gcp"
```

If you are using NFSv4:

```
$ ./linux/twistcli console install swarm --volume-driver "local"
```
If you are using a local storage (not recommended for production environments):

```
$ ./linux/twistcli console install swarm --volume-driver "local"
```

**STEP 6** | At the prompt, enter your Prisma Cloud access token. The access token is required to retrieve the Prisma Cloud container images from the cloud repository.

**STEP 7** | Validate that Console is running. It takes a few moments for the replica count to go from 0/1 to 1/1.

```
$ docker service ls
```

```
ID               NAME                MODE           REPLICAS      IMAGE
pctny1pymjg8     twistlock-console   replicated     1/1
registry.twistlock.com/...
```

**STEP 8** | Open Console's dashboard in a web browser.

Console's published ports use Swarm's routing mesh (ingress network), so the Console service is accessible at the target port on every node, not just the host it runs on.

**STEP 9** | Open Prisma Cloud Console's web interface. By default, the web interface is available via HTTPS (port 8083). Go to https://<LOAD-BALANCER>:8083.

If you did not configure a load balancer, Console is reachable via HTTPS at https://<ANY-SWARM-NODE-IPADDR>:8083

**STEP 10** | Create your first admin user.

**STEP 11** | Enter your license key, and click OK.

You are redirected to the Console dashboard.

**Install Defender**

Defender is installed as a global service, which ensures it runs on every node in the cluster. Console provides a GUI to configure all the options required to deploy Defender into your environment.

**STEP 1** | Open Console.

**STEP 2** | Go to Manage > Defenders > Names.

**STEP 3** | Click Add SAN, and add the DNS name of your load balancer.

**STEP 4** | Go to Manage > Defenders > Deploy > Swarm.

**STEP 5** | Work through each of the configuration options:

1. Choose the DNS name of your load balancer. Defenders use this address to communicate with Console.
2. Choose the registry that hosts the Defender image. Select Prisma Cloud’s registry.
3. Set Deploy Defenders with SELinux Policy to Off.
4. Copy the generated curl-bash command.
STEP 6 | Connect to your Swarm master.

```
$ ssh <SWARM-MASTER>
```

STEP 7 | Paste the curl-bash command into your shell, then run it. You need sudo privileges to run this command.

```
$ curl -sSL -k --header "authorization: Bearer <TOKEN>" ...
```

STEP 8 | Validate that the Defender global service is running.

Open Console, then go to Manage > Defenders > Manage. The table lists all Defenders deployed to your environment (one per node).

Uninstall

To uninstall Prisma Cloud, reverse the install steps. Delete the Defender global service first, followed by the Console service.

STEP 1 | Delete the Defender global service.

1. Open Console, then go to Manage > Defenders > Deploy Swarm.
2. Scroll to the bottom of the page, then copy the last curl-bash command, where it says The script below uninstalls the Swarm Defenders from the cluster.
3. Connect your Swarm master.

```
$ ssh <SWARM-MASTER>
```
4. Paste the curl-bash command into your shell, then run it.

```
$ curl -sSL -k --header "authorization: Bearer <TOKEN>" ...
```

STEP 2 | Delete the Console service.

1. SSH to the node where you downloaded and unpacked the Prisma Cloud release tarball.
2. Run twistcli with the uninstall subcommand.

```
$ ./linux/twistcli console uninstall swarm
```

Using a private registry

For maximum control over your environment, you might want to store the Prisma Cloud container images in your own private registry, and then install Prisma Cloud from your private registry.

When you deploy Prisma Cloud as a service, Docker Swarm pulls the Console image from the specified registry, and then schedules it to run on a node in the cluster.

Docker Hub and Docker Trusted Registry

Prisma Cloud currently only supports Docker Hub and Docker Trusted Registry for Swarm deployments.

The key steps in the deployment workflow are:

STEP 1 | Log into your registry with docker login.

STEP 2 | Push the Console image your registry.
STEP 3 | Install Console using **twistcli**.

Set the `--registry-address` option to your registry and repository. Set the `--skip-push` option so that twistcli doesn’t try to automatically push the Console image to your registry for you.

**Unsupported registries**

If you are using an unsupported registry, you must manually make the Console image available on each node in your cluster. Unsupported registries include Quay.io, Artifactory, and Amazon EC2 Container Registry.

The method documented here supports any registry. The key steps in this deployment workflow are:

- Manually push the Console image to your registry. The **twistcli** tool is not capable of doing it for you.
- Manually pull the Console image to each node in your cluster.
- Run **twistcli** to deploy Console, bypassing any options that interact with the registry. In particular, use the `--skip-push` option because **twistcli** does not know how to authenticate and push to unsupported registries.

The commands in this procedure assume you are using Quay.io, but the same method can be applied to any registry. Adjust the commands for your specific registry.

**STEP 1** | Download the Prisma Cloud current release from Releases, and copy it to your master node.

**STEP 2** | Unpack the Prisma Cloud release tarball.

```bash
$ mkdir twistlock
$ tar xvzf twistlock_<VERSION>.tar.gz -C twistlock/
```

**STEP 3** | Login to your registry.

```bash
$ docker login quay.io
Username: 
Password: 
Email: 
```

**STEP 4** | Load the Console image shipped in the release tarball.

```bash
$ docker load < twistlock_console.tar.gz
```

**STEP 5** | Tag the Console image according to the format required by your registry.

```bash
$ docker tag twistlock/private:console_<VERSION> quay.io/<USERNAME>/
twistlock:console
```

**STEP 6** | Push the Console image to your registry.

```bash
$ docker push quay.io/<username>/twistlock:console
```

**STEP 7** | Connect to each node in your cluster, and pull the Console image.

```bash
$ docker pull quay.io/<username>/twistlock:console
```
STEP 8 | On your Swarm master, run `twistcli` to deploy Console.

```
./linux/twistcli console install swarm \
   --volume-driver "<VOLUME-DRIVER>" \
   --registry-address "quay.io/<USERNAME>"
```
Amazon ECS

This quickstart guide shows you how to deploy Prisma Cloud on a simple cluster that has a single infrastructure node and two worker nodes. Console runs on the infrastructure node, and an instance of Defender runs on each of the worker nodes.

Console is the Prisma Cloud management interface, and it runs as a service. The parameters of the service are described in a task definition, and the task definition is written in JSON format.

Defender protects your containerized environment according to the policies you set in Console.

To automatically deploy an instance of Defender on each worker node in your cluster, you will use a user data script in the worker node launch configuration. User data scripts run custom configuration commands when a new instance is started. You will set up the user data script to call the Prisma Cloud API to download, install, and start Defender.

This guide assumes you know very little about AWS ECS. As such, it is extremely prescriptive. If you are already familiar with AWS ECS and do not need assistance navigating the interface, simply read the section synopsis, which summarizes all key configurations.

The installation described in this article is meant to be "highly available" in that data is persisted across restarts of the nodes. If an infrastructure node were to go down, ECS should be able to reschedule the Console service on any healthy node, and Console should still have access to its state. To enable this capability, you must attach storage that is accessible from each of your infrastructure nodes, and Amazon Elastic File System (EFS) is an excellent choice.

When you have more than one infrastructure node, ECS can run Console on any one of them. Defenders need a reliable way to connect to Console, no matter where it runs. A load balancer automatically directs traffic to the node where Console runs, and offers a stable interface that Defenders can use to connect to Console and that operators can use to access its web interface.

We assume you are deploying Prisma Cloud to the default VPC. If you are not using the default VPC, adjust your settings accordingly.

Key details

There are a number of AWS resource identifiers and other details that are used throughout the install procedure. You should create a list of the following details for easy retrieval during the installation process.

Cluster name: retain this after creating the ECS cluster. Default value: pc-ecs-cluster.

Security group name: retain this after creating the security group. Default value: pc-security-group.

Mount command for console EFS: retain this after creating an EFS for the console.


License Key: License key for Prisma Cloud.

Version: The version of Prisma Cloud you are deploying, for example 20_04_169

Load Balancer's public DNS: retain this after configuring a loadbalancer for your infrastructure nodes.

Download the Prisma Cloud software

The Prisma Cloud release tarball contains all the release artifacts.
STEP 1 | Go to the Releases page, and copy the link to the current recommended release.

STEP 2 | Retrieve the release tarball.

```bash
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```

STEP 3 | Unpack the Prisma Cloud release tarball.

```bash
$ mkdir twistlock
$ tar xzvf prisma_cloud_compute_edition_<VERSION>.tar.gz  -C twistlock/
```

Create a cluster

Create an empty cluster named `pc-ecs-cluster`. Later, you will create launch configurations and auto-scaling groups to start EC2 instances in the cluster.

STEP 1 | Log into the AWS Management Console.

STEP 2 | Go to Services > Containers > Elastic Container Service.

STEP 3 | Click Create Cluster.

STEP 4 | Select Networking only, then click Next Step.

STEP 5 | Enter a cluster name, such as `pc-ecs-cluster`.

STEP 6 | Click Create.

Create a security group

Create a new security group named `pc-security-group` that opens ports 8083 and 8084. In order for Prisma Cloud to operate properly, these ports must be open. This security group will be associated with the EC2 instances started in your cluster.

Console’s web interface and API are served on port 8083. Defender and Console communicate over a secure web socket on port 8084.

Inbound connection to port 2049 is required to setup the NFS.

Open port 22 so that you can SSH to any machine in the cluster.

Additional hardening can be performed as desired for the below roles. For example, limiting access to port 22 only to source IPs from which you would connect to your instances via SSH.

STEP 1 | Go to Services > Compute > EC2.

STEP 2 | In the left menu, click NETWORK & SECURITY > Security Groups.

STEP 3 | Click Create Security Group.

STEP 4 | In Security group name, enter a name, such as `pc-security-group`.

STEP 5 | In Description, enter Prisma Cloud ports.

STEP 6 | In VPC, select your default VPC.
STEP 7 | Under the Inbound rules section, click Add Rule.
   1. Under Type, select Custom TCP.
   2. Under Port Range, enter 8083-8084.
   3. Under Source, select Anywhere.

STEP 8 | Click Add Rule.
   1. Under Type, select Custom TCP.
   2. Under Port Range, enter 2049.
   3. Under Source, select Anywhere.

STEP 9 | Click Add Rule.
   1. Under Type, select SSH.
   2. Under Source, select Anywhere.

STEP 10 | Click Create security group.

Create an EFS file system for Console

Create the Console EFS file system, then capture the mount command that will be used to mount the file system on every infrastructure node.

**Prerequisites:**

Prisma Cloud Console depends on an EFS file system with the following performance characteristics:

- **Performance mode**: General purpose.
- **Throughput mode**: Provisioned. Provision 0.1 MiB/s per deployed Defender. For example, if you plan to deploy 10 Defenders, provision 1 MiB/s of throughput.

The EFS file system and ECS cluster must be in the same VPC and security group.

STEP 1 | Log into the AWS Management Console.
STEP 2 | Go to Services > Storage > EFS.
STEP 3 | Click Create File System.
STEP 4 | Select a VPC, select the pc-security-group for each mount target, then click Next Step.
STEP 5 | Enter a value for Name, such as pc-efs-console
STEP 6 | Set your throughput mode to Provisioned, and adjust Throughput to 0.1 MiB/s per Defender that would be deployed.
STEP 7 | Click Next Step.
STEP 8 | For Configure client access, keep the default settings and click Next Step.
STEP 9 | Review your settings and select Create File System.
STEP 10 | Click on the Amazon EC2 mount instructions (from local VPC) link and copy the mount command (Using the NFS client) and set it aside as the Console mount command.
You will use this mount command to configure your launch configuration for the Console.

Set up a classic load balancer

Set up an AWS Classic Load Balancer, and capture the Load Balancer DNS name.

You'll create two load balancer listeners. One is used for Console’s UI and API, which are served on port 8083. Another is used for the websocket connection between Defender and Console, which is established on port 8084.

For detailed instructions on how to create a load balancer for Console, see Configure an AWS Load Balancer for ECS.

Deploy Console

Launch an infrastructure node that runs in the cluster, then start Prisma Cloud Console as a service on that node.

Create a launch configuration for the infrastructure node

Launch configurations are templates that are used by an auto-scaling group to start EC2 instances in your cluster.

Create a launch configuration named `pc-infra-node` that:

- Creates an instance type of t2.large, or higher. For more information about Console’s minimum requirements, see System requirements.
- Runs Amazon ECS-Optimized Amazon Linux 2 AMI.
- Uses the `ecsInstanceRole` IAM role.
- Runs a user data script that joins the `pc-ecs-cluster` and defines a custom attribute named `purpose` with a value of `infra`. Console tasks will be placed to this instance.

**STEP 1** | Go to Services > Compute > EC2.

**STEP 2** | In the left menu, click AUTO SCALING > Launch Configurations.

**STEP 3** | Click Create launch configuration.

**STEP 4** | Choose an AMI.
   1. Click AWS Marketplace.
   2. In the search box, enter Amazon ECS-Optimized Amazon Linux 2 AMI.
   3. Click Select for Amazon ECS-Optimized Amazon Linux 2 AMI.

**STEP 5** | Choose an instance type.
   1. Select t2.large.
   2. Click Next: Configure details.

**STEP 6** | Configure details.
   1. In Name, enter a name for your launch configuration, such as `pc-infra-node`.
   2. In IAM role, select `ecsInstanceRole`.

   *If this role doesn’t exist, see Amazon ECS Container Instance IAM Role.*
4. Expand **Advanced Details**.
5. In **User Data**, enter the following text in order to install the NFS utilities and mount the EFS file system:

```bash
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=pc-ecs-cluster
ECS_INSTANCE_ATTRIBUTES={"purpose": "infra"}
EOF

yum install -y nfs-utils
mkdir /twistlock_console
<CONSOLE_MOUNT_COMMAND> /twistlock_console
mkdir -p /twistlock_console/var/lib/twistlock
mkdir -p /twistlock_console/var/lib/twistlock-backup
mkdir -p /twistlock_console/var/lib/twistlock-config
```

`pc-ecs-cluster` must match your cluster name. If you’ve named your cluster something other than `pc-ecs-cluster`, then modify your User Data script accordingly.

`<CONSOLE_MOUNT_COMMAND>` is the Console mount command you copied from the AWS Management Console after creating your console EFS file system. The mount target must be `/twistlock_console`, not the `efs` mount target provided in the sample command.

6. (Optional) Under **IP Address Type**, select **Assign a public IP address to every instance**.
   With this option, you can easily SSH to this instance to troubleshoot issues.

7. Click **Next: Add Storage**.

**STEP 7 | Add Storage.**
1. Accept the defaults, and click **Next: Configure Security Group**.

**STEP 8 | Configure security group.**
1. Under **Assign a security group**, choose **Select an existing security group**.
2. Select `pc-security-group`.
3. Click **Review**.

**STEP 9 | Review.**
1. Review the configuration and select **Create launch configuration**.

**STEP 10 | Select an existing key pair, or create a new key pair so that you can access your instance.**

**STEP 11 | Click Create launch configuration.**

*Create an auto scaling group for the infrastructure node*

Launch a single instance of the infrastructure node into your cluster.

**STEP 1 | Go to** Services > Compute > EC2.

**STEP 2 | In the left menu, click** AUTO SCALING > Auto Scaling Groups.

**STEP 3 | Click** Create Auto Scaling group.
1. Select Launch Configuration
2. Select `pc-infra-node`. 
3. Click Next Step.

**STEP 4 | Configure Auto Scaling group details.**
1. In Group Name, enter pc-infra-autoscaling.
2. Set Group size to the desired value (typically, this is a value greater than 1).
3. Under Network, select your default VPC.
4. Under Subnet, select a public subnet, such as 172.31.0.0/20.
5. Click Next: Configure scaling policies.

**STEP 5 | Configure scaling policies.**
1. Select Keep this group at its initial size.
2. Click Next: Configure Notifications.

**STEP 6 | Configure Notifications.**
1. Click Next: Configure Tags.

**STEP 7 | Configure Tags.**
1. Under Key, enter Name.
2. Under Value, enter pc-infra-node.
3. Click Review.

**STEP 8 | Review the configuration and click Create Auto Scaling Group.**

After the auto scaling group spins up (it will take some time), validate that your cluster has one container instance, where a container instance is the ECS vernacular for an EC2 instance that has joined the cluster and is ready to accept container workloads:

- Go to Services > Containers > Elastic Container Service. The count for Container instances should be 1.
- Click on the cluster, then click on the ECS Instances tab. In the status table, there should be a single entry. Click on the link under the EC2 Instance column. In the details page for the EC2 instance, record the Public DNS.

**Copy the Prisma Cloud config file into place**

The Prisma Cloud API serves the version of the configuration file used to instantiate Console. Use scp to copy twistlock.cfg from the Prisma Cloud release tarball to /twistlock_console/var/lib/twistlock-config on the infrastructure node.

**STEP 1 | Upload twistlock.cfg to the infrastructure node.**
1. Go to the directory where you unpacked the Prisma Cloud release tarball.
2. Copy twistlock.cfg to the infrastructure node.

```
$ scp -i <PATH-TO-KEY-FILE> twistlock.cfg ec2-user@<ECS_INFRA_NODE_DNS_NAME>:~
```

**STEP 2 | SSH to the infrastructure node.**

```
$ ssh -i <PATH-TO-KEY-FILE> ec2-user@<ECS_INFRA_NODE_DNS_NAME>
```
STEP 3 | Copy the **twistlock.cfg** file into place.

```bash
$ sudo cp twistlock.cfg /twistlock_console/var/lib/twistlock-config
```

**Create a Prisma Cloud Console task definition**

Prisma Cloud provides a task definition template for Console. Download the template, then update the variables specific to your environment. Finally, load the task definition in ECS.

**Prerequisites:**
- The task definition provisions sufficient resources for Console to operate. Our template specifies reasonable defaults. For more information, see System requirements.

**STEP 1 | Download the Prisma Cloud Console task definition**, and open it for editing.

**STEP 2 | Update the value for **image** to point to Prisma Cloud's cloud registry:**

Replace the following placeholder strings with the appropriate values:
- `<ACCESS-TOKEN>` — Your Prisma Cloud access token. All characters must be lowercase. To convert your access token to lowercase, run:
  ```bash
  $ echo <ACCESS-TOKEN> | tr '[:upper:]' '[:lower:]'
  ```
- `<VERSION>` — Version of the Console image to use.
  For example: for version `20.04.177`, specify `20_04_177`. The image will look similar to `console:console_20_04_177`.

**STEP 3 | Update `<CONSOLE-DNS>` to the Load Balancer’s DNS name.**

**STEP 4 | Go to Services > Containers > Elastic Container Service.**

**STEP 5 | In the left menu, click **Task Definitions**.**

**STEP 6 | Click **Create new Task Definition**.**

**STEP 7 | In Step 1: Select launch type compatibility, select EC2, then click **Next step**.**

**STEP 8 | In Step 2: Configure task and container definitions, scroll to the bottom of the page and click **Configure via JSON**.**

**STEP 9 | Delete the contents of the window, and replace it with the Prisma Cloud Console task definition**

**STEP 10 | Click **Save**.**

1. (Optional) Change the task definition name before creating. The JSON will default the name to `pc-console`.

**STEP 11 | Click **Create**.**

**Launch the Prisma Cloud Console service**

Create the Console service using the previously defined task definition. A single instance of Console will run on the infrastructure node.
STEP 1 | Go to Services > Containers > Elastic Container Service.

STEP 2 | In the left menu, click Clusters.

STEP 3 | Click on your cluster.

STEP 4 | In the Services tab, then click Create.

STEP 5 | In Step 1: Configure service:
   1. For Launch type, select EC2.
   2. For Task Definition, select pc-console.
   3. In Service Name, enter pc-console.
   4. In Number of tasks, enter 1.
   5. Click Next Step.

STEP 6 | In Step 2: Configure network:
   1. For Load Balancer type, select Classic Load Balancer.
   2. For Service IAM role, leave the default ecsServiceRole.
   3. For Load Balancer Name, select previously created load balancer.
   4. Unselect Enable Service discovery integration
   5. Click Next Step.

STEP 7 | In Step 3: Set Auto Scaling, accept the defaults, and click Next.

STEP 8 | In Step 4: Review, click Create Service.

STEP 9 | Wait for the service launch to be completed and click View Service.

STEP 10 | Wait for the service Last status to change to running (can take a few minutes) and continue to Configure Prisma Cloud Console below.

Configure Prisma Cloud Console

Navigate to Console’s web interface, create your first admin account, then enter your license.

STEP 1 | Start a browser, then navigate to https://<Load Balancer DNS Name>:8083

STEP 2 | At the login page, create your first admin account. Enter a username and password.

STEP 3 | Enter your license key, then click Register.

Deploy Defender

Launch an infrastructure node that runs in the cluster

You are now ready to deploy your worker nodes. You will create worker nodes that run in the cluster, an ECS Task Definition for the Prisma Cloud Defender, then create a service of type Daemon to ensure that the Defender is deployed across your ECS cluster.

Create a launch configuration for worker nodes

Create a launch configuration named pc-worker-node that:

- Runs the Amazon ECS-Optimized Amazon Linux 2 AMI.
• Uses the ecsInstanceRole IAM role.
• Runs a user data script that joins the pc-ecs-cluster and runs the commands required to install Defender.

STEP 1 | Go to Services > Compute > EC2.

STEP 2 | In the left menu, click AUTO SCALING > Launch Configurations.

STEP 3 | Click Create Launch Configuration

STEP 4 | Choose an AMI:
  1. Click AWS Marketplace.
  2. In the search box, enter Amazon ECS-Optimized Amazon Linux 2 AMI.
  3. Click Select for Amazon ECS-Optimized Amazon Linux 2 AMI.

STEP 5 | Choose an instance type.
  1. Select t2.medium.
  2. Click Next: Configure details.

STEP 6 | Configure details.
  1. In Name, enter a name for your launch configuration, such as pc-worker-node.
  2. In IAM role, select ecsInstanceRole.
  3. Select Enable CloudWatch detailed monitoring.
  4. Expand Advanced Details,
  5. In User Data, enter the following text:

```bash
#!/bin/bash
echo ECS_CLUSTER=pc-ecs-cluster >> /etc/ecs/ecs.config
```

Where:
• ECS_CLUSTER must match your cluster name. If you’ve named your cluster something other than pc_ecs_cluster, then modify your User Data script accordingly.
  6. (Optional) Under IP Address Type, select Assign a public IP address to every instance.

With this option, you can easily SSH to any worker nodes instances and troubleshoot issues.

7. Click Next: Add Storage.

STEP 7 | Add Storage.
  • Accept the defaults, and click Next: Configure Security Group.

STEP 8 | Configure security group.
  1. Under Assign a security group, choose Select an existing security group.
  2. Select pc-security-group.
  3. Click Review.

STEP 9 | Review.
  • Review the configuration and select Create launch configuration.

STEP 10 | Select an existing key pair, or create a new key pair so that you can access your instance.

Create an auto scaling group for the worker nodes
Launch two worker nodes into your cluster.
STEP 1 | Go to Services > Compute > EC2.

STEP 2 | In the left menu, click AUTO SCALING > Auto Scaling Groups.

STEP 3 | Click Create Auto Scaling group:
   1. Select Launch Configuration
   2. Select pc-worker-node.
   3. Click Next Step.

STEP 4 | Configure Auto Scaling group details:
   1. In Group Name, enter pc-worker-autoscaling.
   2. Set Group size to 2.
   3. Under Network, select your default VPC.
   4. Under Subnet, select a public subnet, such as 172.31.0.0/20.
   5. Click Next: Configure scaling policies.

STEP 5 | Configure scaling policies.
   1. Select Keep this group at its initial size.
   2. Click Next: Configure Notifications.

STEP 6 | Configure Notifications.
   1. Click Next: Configure Tags.

STEP 7 | Configure Tags.
   1. Under Key, enter Name.
   2. Under Value, enter pc-worker-node.
   3. Click Review.

STEP 8 | Review the configuration and click Create Auto Scaling Group.

STEP 9 | After the auto scaling group spins up (it will take some time), validate that your cluster has three container instances.
   1. Go to Services > Containers > Elastic Container Service.
   2. The count for Container instances in your cluster should now be a total of three.

**Generate install bundle for Defender**

Generate install bundle which will be used in Defender's task definition.

STEP 1 | Retrieve the service parameter from the Prisma Cloud API.

```
$ curl -k \
-u "<username>:<password>" \
-X GET https://<load_balancer_dns>:8083/api/v1/certs/service-parameter \
-o service-parameter
```

STEP 2 | Ensure the `jq` package is installed.

STEP 3 | Retrieve and retain the installBundle from the Prisma Cloud API:

```
$ curl -k -s \
-u "<username>:<password>" \
```
Create a Prisma Cloud Defender task definition

Prisma Cloud provides a task definition template for Defender. Download the template, then update the variables specific to your environment. Finally, load the task definition in ECS.

**STEP 1** | Download the **Prisma Cloud Defender task definition**, and open it for editing.

**STEP 2** | Apply the following changes to the task definition:

1. Modify the *WS_ADDRESS* parameter to the DNS of the Console.
   - `<CONSOLE-DNS>` — The DNS name for the load balancer you created.
   - `<PORT>` — The port the DNS is listening on.
   
   *The default port is 8084.*

2. `<INSTALL-BUNDLE>` — Output from the installBundle endpoint.
3. `<SERVICE-PARAMETER>` — Output from the service-parameter endpoint.
4. Update the value for *image* to point to Prisma Cloud's public registry by replacing the following placeholder strings with the appropriate values:
   - `<ACCESS-TOKEN>` — Your Prisma Cloud access token. This is located in your Console under Manage > System > Intelligence.
     
     All characters must be lowercase.
     
     To convert your access token to lowercase, run:
     
     ```bash
     $ echo <ACCESS-TOKEN> | tr '[:upper:]' '[:lower:]'
     ```
   - `<VERSION>` — Version of the Defender image to use.
     
     For example: for version 20.04.177, specify `20_04_177`. The image will look similar to `defender:defender_20_04_177`.

**STEP 3** | Go to **Services > Containers > Elastic Container Service**.

**STEP 4** | In the left menu, click **Task Definitions**.

**STEP 5** | Click **Create new Task Definition**.

**STEP 6** | In **Step 1: Select launch type compatibility**, select *EC2*, then click **Next step**.

**STEP 7** | In **Step 2: Configure task and container definitions**, scroll to the bottom of the page and click **Configure via JSON**.

**STEP 8** | Delete the contents of the window, and replace it with the Prisma Cloud Console task definition

**STEP 9** | Click **Save**.

1. *(Optional) Change the task definition name before creating. The JSON will default the name to *pc-defender*. 

-X GET "https://<load_balancer_dns>:8083/api/v1/defenders/install-bundle?consoleaddr=<load_balancer_dns>&defenderType=appEmbedded" | jq -r 

'.installBundle' > install-bundle
STEP 10 | Click Create.

**Launch the Prisma Cloud Defender service**

Create the Defender service using the previously defined task definition. Using Daemon scheduling, one Defender will run per node in your cluster.

**STEP 1 |** Go to Services > Containers > Elastic Container Service.

**STEP 2 |** In the left menu, click Clusters.

**STEP 3 |** Click on your cluster.

**STEP 4 |** In the Services tab, then click Create.

**STEP 5 |** In Step 1: Configure service:
1. For Launch type, select EC2.
2. For Task Definition, select pc-defender.
3. In Service Name, enter pc-defender.
4. In Service Type, select Daemon.
5. Click Next Step.

**STEP 6 |** In Step 2: Configure network, accept the defaults, and click Next step.

**STEP 7 |** In Step 3: Set Auto Scaling, accept the defaults, and click Next step.

**STEP 8 |** In Step 4: Review, click Create Service.

**STEP 9 |** Click View Service.

**STEP 10 |** Verify that you have Defenders running on each node in your ECS cluster.
- Go to your Prisma Cloud Console and view the list of Defenders in Manage > Defenders > Manage (there should be a total for 3 defenders → one for each EC2 instance).

**Using a private registry**

For maximum control over your environment, you might want to store the Console container image in your own private registry, and then install Prisma Cloud from your private registry. When the Console service is started, ECS retrieves the image from your registry. This procedure shows you how to push the Console container image to Amazon's Elastic Container Registry (ECR).

**Prerequisites:**
- AWS CLI is installed on your machine. It is required to push the Console image to your registry.

**STEP 1 |** Go to the directory where you unpacked the Prisma Cloud release tarball.

```
$ cd twistlock/
```

**STEP 2 |** Load the Console image.

```
$ docker load < ./twistlock_console.tar.gz
```
STEP 3 | Go to Services > Containers > Elastic Container Service.

STEP 4 | In the left menu, click Repositories.

STEP 5 | Click Create repository.

STEP 6 | Follow the AWS instructions for logging in to the registry, tagging the Console image, and pushing it to your repo.

   Be sure to update your Console task definition so that the value for image points to your private registry.
DC/OS (Marathon/Mesos)

DC/OS is a distributed operating system. It integrates several open-source components to enable the management of multiple machines as if they were a single computer. DC/OS is built on the Apache Mesos distributed systems kernel and the Marathon container orchestration system. This procedure was tested on Mesosphere DC/OS 1.11.

To deploy Prisma Cloud to a Kubernetes cluster running on DC/OS, see the Install Kubernetes guide.

Deployment architecture

Prisma Cloud Console is deployed to a public slave node. Public slave nodes are accessible from outside the cluster. Because Console provides a management interface through a browser window, it must be accessible from outside the cluster.

Prisma Cloud Defender is deployed on every private slave node. Slave nodes run your applications. Slave nodes reside on a private subnet, so they are not accessible from outside the cluster.

The following diagram shows the components in a DC/OS cluster, and where Prisma Cloud gets deployed.

Install Console

Use the twistlock.sh script to install Console onto a public agent in your cluster.

Prerequisites:

- You have installed the DC/OS CLI tool.
• Ports 8083 and 8084 on the public agent are open. Console’s web interface and API are served on port 8083, and Defender communicates with Console on port 8084. If you deploy DC/OS with the Terraform-based Universal Installer, add 8083 and 8084 to the `public_agents_additional_ports` input in your `main.tf` configuration file.

**STEP 1** | Go to [Releases](#) and copy the link to current recommended release.

**STEP 2** | SSH to a public agent in your cluster.

```bash
$ dcos node ssh --master-proxy --mesos-id=<PUBLIC-AGENT>
```

**STEP 3** | Retrieve the Prisma Cloud release tarball.

```bash
$ wget <LINK-TO-CURRENT-RECOMMENDED-RELEASE>
```

**STEP 4** | Unpack the Prisma Cloud release tarball.

```bash
$ mkdir twistlock
$ tar xvzf twistlock_<VERSION>.tar.gz -C twistlock/
```

**STEP 5** | Install Prisma Cloud.

We recommend that you install both Console and Defender on the public agent. To simplify the installation of both components, use the `onebox` install target.

```bash
$ cd twistlock
$ sudo ./twistlock.sh -s onebox
```

Where:

- `-s` --
  Agree to the EULA.
- `-z` --
  (Optional) Emit additional debug messages. Use this option if an error occurs while running `twistlock.sh`.
- `onebox` --
  Install Console and Defender onto a single server.

**STEP 6** | Verify that Prisma Cloud has been properly installed, and that it is running on your host:

```bash
$ sudo docker ps --format "table {{.ID}}\t{{.Status}}\t{{.Names}}"
CONTAINER ID        STATUS              NAMES
764ecb72207e        Up 5 minutes        twistlock_defender
be5e385fea32        Up 5 minutes        twistlock_console
```

**STEP 7** | Open a browser window, and navigate to Console’s front page.

Go to https://<PUBLIC-AGENT>:8083

Where `<PUBLIC-AGENT>` is the hostname or IP address of the public agent. To get the IP address for your public agent, run the following command:

```bash
$ for id in $(dcos node --json | \ 
  jq --raw-output \
```
STEP 8 | Create your first admin user by entering a username and password.

STEP 9 | Enter your license key.

After your license is validated, the dashboard is unlocked.

STEP 10 | Go to Manage > Defenders > Names. Verify that the Subject Alternative Name (SAN) table contains the IP address for the public agent where Console runs. If not, click Add SAN, then enter the IP address.

Install Defender

Deploy Defender to all slave agents in your cluster. Use twistcli to generate the Defender app in JSON format, and then start it with the DC/OS CLI tool. By default, the Defender image is retrieved from Prisma Cloud’s cloud registry.

Prerequisites:
- You have deployed Console to a public agent in your cluster.

STEP 1 | Download twistcli to a host where you’ve installed the DC/OS CLI.

1. Open Compute Console and go to Manage > System > Downloads.
2. Under twistcli tool, download the version for your operating system.

STEP 2 | Generate the Defender app JSON using twistcli, where:

- `<PLATFORM>` can be linux or osx.
- `<CONSOLE>` is the IP address of the public agent where Console runs.
- `<ADMIN_USER>` is a Twistlock Console user with a role of Defender Manager or higher.
- `<NUMBER_OF_AGENTS>` is the number of private agent nodes in your cluster.

The following command connects to Console’s API (specified in --address) as user `<TWISTLOCK_USER>` (specified in --user), and generates a Defender app in JSON format according to the configuration options passed to twistcli.

```
$ <PLATFORM>/twistcli defender export dcos \
   --address https://<CONSOLE>:8083 \
   --user <TWISTLOCK_USER> \ 
   --cluster-address <CONSOLE> \ 
   --agents <NUMBER_OF_AGENTS>
```

STEP 3 | Deploy the Defender app on your cluster using the dcos CLI tool.

Alternatively, you could deploy the Defender app using the DC/OS web interface, Marathon web interface, or Marathon REST API.

```
$ dcos marathon app add ./dcos.json
```
STEP 4 | Validate the Defender app is running.

$ dcos marathon app list
Windows

Prisma Cloud can secure Windows containers running on Windows Server 2016 and Windows Server 2019 hosts. A single instance of Prisma Cloud Console can simultaneously protect both Windows and Linux containers on both Windows and Linux hosts. Prisma Cloud’s Intelligence Stream includes vulnerability data from Microsoft, so as new CVEs are reported, Prisma Cloud can detect them in your Windows images.

The architecture for Defender on Windows is different than Defender on Linux. Rather than running as a Docker container (as it does on Linux), Defender runs as a Windows service. And rather than implementing runtime protection in userspace (as it does on Linux), Windows drivers are used. This is because there is no concept of capabilities in Windows Docker containers like there is on Linux. Defender on Windows runs as service so it can acquire the permissions it needs to secure the containers on your host. When you deploy the Defender, it appears as a service. The Defender type "Container Defender - Windows" means that Defender is capable of securing your containers, not that it’s deployed as a container.

To deploy Defender on Windows, you’ll copy a PowerShell script from the Prisma Cloud Console and run it on the host where you want to install Defender.

Feature matrix

Full feature parity on both Windows and Linux is a key objective. With every release, Prisma Cloud expands Defender’s capabilities on Windows.

The following table shows the current state of Prisma Cloud’s Windows support:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Vulnerability</th>
<th>Compliance</th>
<th>Runtime defense</th>
<th>Firewalls</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>&gt;Process</td>
<td>&gt;Network</td>
</tr>
<tr>
<td>Linux</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Windows 2016</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Windows 2019</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Windows containers support a block action for runtime defense but do not support a block action for vulnerability and compliance.

As a quick review, Prisma Cloud runtime defense builds a model of allowed activity for each container image during a so-called learning period. After the learning period has completed, any violation of the model triggers an action as defined by your policy (alert, prevent, block).

As Prisma Cloud builds the model, any interactive tasks that are run are logged. These interactive tasks can be viewed in the History tab of each model. On Windows, Prisma Cloud cannot currently detect when interactive tasks are run with the `docker exec` command, although Prisma Cloud does correctly record interactive tasks run from a shell inside a container with the `docker run -it <IMAGE> sh` command. No matter how the interactive task is run, however, the model will correctly allow a process if it’s in learning mode, and it will take action if the model is violated when in enforcement mode.
Deploying Defender on Windows

Prisma Cloud Console must be first installed on a Linux host. Prisma Cloud Defenders are then installed on each Windows host you want to protect. For more information about installing Console, see Getting Started. The Onebox install is the fastest way to get Console running on a stand-alone Linux machine.

Defenders are deployed with a PowerShell 64-bit script, `defender.ps1`, which downloads the necessary files from Console. Defender is registered as a Windows service.

Run the Prisma Cloud Defender deployment PowerShell script from a Windows PowerShell 64-bit shell.

After the install is completed, Prisma Cloud files can be found in the following locations:

- `C:\Program Files\Prisma Cloud\`
- `C:\ProgramData\Prisma Cloud\`

Prerequisites:

- Windows Server 2016 or Windows Server 2019. Prisma Cloud is not supported on Windows 10 or Hyper-V.
- Docker for Windows (1.12.2-cs2-ws-beta) or higher. For more information about installing Docker on Windows, see Windows Containers on Windows Server.

STEP 1 | Log into Console.

STEP 2 | Go to Manage > Defenders > Deploy.

STEP 3 | In Choose the Defender type, select Docker on Windows.

STEP 4 | Copy the curl script and run it on your host to install Windows Defender.

If you install Windows locally on your laptop, the 'netsh' commands are not needed. They are only applicable to the GCE environment.

Registry scanning

To scan Windows images in your registry, you must install at least one Windows Defender. Prisma Cloud automatically distributes the scan job across available Defenders. To scan registries that hold both Windows and Linux images, install at least one Linux Defender and one Windows Defender in your environment.

Uninstalling Defender

You can uninstall Defender directly from the Console UI.

You can also manually uninstall Defender from the command line by running:

```
C:\Program Files\Twistlock\scripts\defender.ps1 -uninstall
```

Since Defender runs as a Windows service, decommissioning it will stop the service. Some remnant files might need to be deleted manually.

STEP 1 | Go to Manage > Defenders > Manage.

This page shows a list of Defenders deployed in your environment and connected to Console.
STEP 2 | Click the Decommission button.
Defender types

Defenders enforce the policies you set in Console. They come in a number of different flavors. Each flavor is designed for protecting specific types of cloud-native resources and for optimal deployment into the environment, with full support for automated workflows. Use the following flow chart to choose the best Defender for the job.

In general, deploy Container Defender whenever you can. It offers the most features, it can simultaneously protect both containers and host, and nothing needs to be embedded inside your containers for Defender to be able to protect them.
Start

Is Twistlock Console installed?

Yes

Do you control the underlying infrastructure (cluster, hosts, OS)?

Yes

Host and its containers

Secure a serverless function

Embed Serverless Defender into the function. Twistlock supports runtime protection for AWS Lambda.

No

It's a black box controlled by a cloud service.

No

Can your host run containers?

Yes

Secure a container

Embed RASP Defender into the container (or task), then Twistlock can protect it anywhere it runs.

Examples:
Azure Container Instances
Pivotal PAS
DC/OS Containerizer

No

Choose the right RASP Defender type.

Fargate?

Yes

No

Docker container?

Yes

No

No

Deploy RASP Defender

Are you securing a container?

Yes

No

Are you securing a serverless function?

Yes

Embed Serverless Defender

No

No
Container Defender (Linux and Windows)

Install Container Defender on any host that runs a container workload. Container Defender protects both your containers and the underlying host. Docker must be installed on the host because this Defender type runs as a container.

Container Defender offers the richest set of capabilities. The deployment is also the simplest. After deploying Container Defender to a host, it can immediately protect and monitor your containers and host. No additional steps are required to rebuild your containers with an agent inside. Container Defender should always be your first choice whenever possible.

There are some minimum requirements to run Container Defender. You should have full control over the host where Container Defender runs. It must be able to run alongside the other containers on the host with select kernel capabilities. And it must be able to run in the host's network and process namespace.

Deploy one Container Defender per host. Container Defender can be deployed in several ways:

- With cluster constructs. Container orchestrators often provide native capabilities for deploying agents, such as Defender, to every node in the cluster. Prisma Cloud leverages these capabilities to install Defender. Kubernetes and OpenShift, for example, offer DaemonSets, and Docker Swarm offers global services. As such, Container Defender is deployed as a DaemonSet on Kubernetes and a global service on Swarm.
- As a stand-alone entity. Stand-alone Container Defenders are installed on hosts that are not part of a cluster.

Host Defender (Linux and Windows)

Host Defender utilizes Prisma Cloud's model-based approach for protecting hosts that do not run containers. This Defender type lets you extend Prisma Cloud to protect all the hosts in your environment, regardless of their purpose. Defender runs as a systemd service on Linux and a Windows service on Windows. If Docker Engine is detected on the host, installation of this Defender type is blocked; install Container Defender instead.

Deploy one Host Defender per host. Do not deploy Host Defender if you've already deployed Container Defender to a host. Container Defender offers the same host protection capabilities as Host Defender.

Serverless Defender

Serverless Defenders offer runtime protection for AWS Lambda functions. Serverless Defender must be embedded inside your functions. Deploy one Serverless Defender per function.

App Embedded Defender

App Embedded Defenders offer runtime protection for containers.

Deploy App Embedded Defender anywhere you can run a container, but you can't run Container Defender. Container-on-demand services are a typical use case for App Embedded Defender. They abstract away the underlying cluster, host, operating system, and software modules (such as Docker Engine) and present them as a single black box. Hooks into the operating system that Container Defender needs to monitor and protect resources aren't available in these environments. Instead, embed App Embedded Defender directly inside the container to establish a point of control. Prisma Cloud supports an automated workflows for embedding App Embedded Defenders.

Deploy one App Embedded Defender per container. For Fargate, deploy one Defender per task.

App Embedded Defender offers three deployment mechanisms: Fargate, Dockerfile, and manual.
**Fargate**

If you have an AWS Fargate task, deploy App Embedded Fargate Defender.

A key attribute of the App Embedded Fargate Defender is that you don’t need to change how the container images in the task are built. The process of embedding the App Embedded Defender simply manipulates the task definition to inject a Prisma Cloud sidecar container, and start existing task containers with a new entrypoint, where the entrypoint binary is hosted by the Prisma Cloud sidecar container. The transformation of an unprotected task to a protected task takes place at the task definition level only. The container images in the task don’t need to be manually modified. This streamlined approach means that you don’t need to maintain two versions of an image (protected and unprotected). You simply maintain the unprotected version, and when you protect a task, Prisma Cloud dynamically injects App Embedded Defender into it.

The Prisma Cloud sidecar container has a couple of jobs:

- Hosts the Defender binary that gets injected into containers in the task.
- Proxies all communication to Console. Even if you have multiple containers in a task, it appears as a single entity in Console’s dashboard.
- Synchronizes policy with Console and sends alerts to Console.

**Dockerfile**

The Docker image format, separate from the runtime, is becoming a universal runnable artifact. If you’re not using Fargate, but something else that runs a Docker image, such as Azure Container Instances or Pivotal PAS, embed App Embedded Defender using the Dockerfile method.

Provide a Dockerfile, and Prisma Cloud returns a new version of the Dockerfile in a bundle. Rebuild the new Dockerfile to embed Prisma Cloud into the container image. When the container starts, Prisma Cloud App Embedded Defender starts as the parent process in the container, and it immediately invokes your program as its child.

There are two big differences between this approach and the Fargate approach:

- With the Fargate approach, you don’t change the actual image. With the Dockerfile approach, you have the original image and a new protected image. You must modify the way your containers are built to embed App Embedded Defender into them. You need to make sure you tag and deploy the right image.
- Each Defender binary makes its own connection to Console. In the Console dashboard, they are each counted as unique applications.

Nothing prevents you from protecting a Fargate task using the Dockerfile approach, but it’s inefficient.

**Manual**

Use the manual approach to protect almost any type of runtime. If you’re not running a Docker image, but you still want Prisma Cloud to protect it, deploy App Embedded Defender with the manual method. Download the App Embedded Defender, set up the required environment variables, then start your program as an argument to the App Embedded Defender.

If you choose the manual approach, you have to figure out how to deploy, maintain, and upgrade your app on your own. While the configuration is more complicated, it’s also the most universal option because you can protect almost any executable.
PCF Defender

PCF Defenders run on your PCF infrastructure and scan the droplets in your blobstores for vulnerabilities. The PCF Defender is delivered as a tile that can be installed from your PCF Ops Manager Installation Dashboard.

Defender capabilities

The following table summarizes the key functional differences between Defender types.

<table>
<thead>
<tr>
<th>Capabilities</th>
<th>Defender type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&gt;Container</td>
</tr>
<tr>
<td>Deployment methods</td>
<td>Console UI</td>
</tr>
<tr>
<td></td>
<td>API</td>
</tr>
<tr>
<td></td>
<td>twistcli</td>
</tr>
<tr>
<td>Vulnerability management</td>
<td>Y</td>
</tr>
<tr>
<td>Compliance</td>
<td>Y</td>
</tr>
<tr>
<td>Runtime defense</td>
<td>Behavioral modeling</td>
</tr>
<tr>
<td></td>
<td>Process</td>
</tr>
<tr>
<td></td>
<td>Networking</td>
</tr>
<tr>
<td></td>
<td>File system</td>
</tr>
<tr>
<td></td>
<td>Forensics</td>
</tr>
<tr>
<td>Firewalls</td>
<td>WAAS</td>
</tr>
<tr>
<td></td>
<td>CNNF</td>
</tr>
<tr>
<td></td>
<td>Radar (visualization)</td>
</tr>
</tbody>
</table>

1 Container Defender also supports all of the Host Defender capabilities.

2 Serverless functions are scanned for vulnerabilities and compliance by the Console. In the Console, create a configuration that points to your repository of functions in your cloud provider.

3 Set up your Jenkins build to scan your container images before they’re pushed to the repository.

Connectivity

Defender must be able to communicate with Console over the network because it pulls policies down and sends data (alerts, events, etc) back to Console.
In simple environments, where your hosts run on the same subnet, you can connect to Console using the host’s IP address or hostname. In more complex environments, where your setup runs in the cloud, it can be more difficult to determine how Defender connects to Console. When setting up Defender, use whichever address routes over your configuration and lets Defender connect to Console.

For example, Console might run in one Virtual Private Cloud (VPC) in AWS, and your containers might run in another VPC. Each VPC might have a different RFC1918 address space, and communication between VPCs might be limited to specific ports in a security group. Use whichever address lets Defender connect to Console. It might be a publicly exposed IP address, a hostname registered with a DNS, or a private address NAT’ed to the actual IP address assigned to Console. For more information about setting up name resolution in complex networks, see Best practices for DNS and certificate management.

Deployment scenarios

Install the Defender type that best secures the resource you want to protect. Install Defender on each host that you want Prisma Cloud to protect. Container Defenders protect both the containers and the underlying host. Host Defenders are designed for legacy hosts that have no capability for running containers. Host Defenders protect the host only. For serverless technologies, embed Defender directly in the resource.

The scenarios here show examples of how the various Defender types can be deployed.

Scenario #1

Stand-alone Container Defenders are installed on hosts that are not part of a cluster. Stand-alone Container Defenders might be required in any number of situations.

For example, a very simple evaluation setup might consist of two virtual machines.

- 1 — One VM runs Onebox (Console + Container Defender).
- 2 — To protect the container workload on a second VM, install another stand-alone Container Defender.

Scenario #2

For clusters, such as Kubernetes, OpenShift, and Swarm, Prisma Cloud utilizes orchestrator-native constructs, such as DaemonSets, to guarantee that Defender runs on every node in the cluster. For example, the following setup has three different types of Defender deployments.

- 1 — In the cluster, Container Defenders are deployed as a DaemonSet. (Assume this is a Kubernetes cluster; it would be a similar construct, but with a different name, for Swarm, DC/OS, etc).
• 2 — On the host dedicated to scanning registry images, which runs outside the cluster, a stand-alone Container Defender is deployed.

• 3 — On the legacy database server, which doesn’t run containers at all, a Host Defender is deployed. Host Defenders are a type of stand-alone Defender that run on hosts that don’t have Docker installed.

Scenario #3

Managed services that run functions and containers on-demand isolate the runtime from the underlying infrastructure. In these types of environments, Defender cannot access the host’s operating system with elevated privileges to observe activity and enforce policies in the runtime. Instead, Defender must be built into the runtime, and control application execution and detect and prevent real-time attacks from within. App Embedded Defender can be deployed to protect any container, regardless of the platform or runtime, whether it’s Docker, runC, or Diego on PCF.

• 1 — Serverless Defender is embedded into each AWS Lambda function.
Install Defender

This section shows you how to install Defender. The type of Defender you install depends on what you're securing.

- Single Container Defender
- Cluster Container Defender
- App Embedded Defender
- App Embedded Defender for Fargate
- PCF Defender for VMWare Tanzu TAS
- Serverless Defender
- Serverless Defender (Lambda layer)
- Host Defender

Install a single Container Defender

Single Container Defenders are installed with a curl-bash script. Install Container Defender on each host that you want Prisma Cloud to protect.

**Prerequisites:**

- Your system meets all minimum system requirements.
- You have already installed Console, and it can be accessed over the network from the host where you want to install Defender.
- Ports 8083 and 8084 are open on the host where Console runs.
- Port 8084 is open on the host where Defender runs. Console and Defender communicate with each other over a web socket on port 8084.
- You have sudo access to the host where Defender will be installed.

**STEP 1 | Verify that the host machine where you install Defender can connect to Console.**

```
$ curl -sk -D - https://<CONSOLE_IP_ADDRESS|HOSTNAME>:8083/api/v1/_ping
```

If curl returns an HTTP response status code of 200, you have connectivity to Console. If you customized the setup when you installed Console, you might need to specify a different port.

**STEP 2 | Log into Console.**
STEP 3 | Go to Manage > Defenders > Deploy.

**Deploy a single Twistlock Defender**

1. In the first drop-down menu (1), select the way Defender connects to Console.

   A list of IP addresses and hostnames are pre-populated in the drop-down list. If none of the items are valid, go to Manage > Defenders > Names, and add a new Subject Alternative Name (SAN) to Console's certificate. After adding a SAN, your IP address or hostname will be available in the drop-down list.

   *Selecting an IP address in a evaluation setup is acceptable, but using a DNS name is more resilient. If you select Console’s IP address, and Console’s IP address changes, your Defenders will no longer be able to communicate with Console.*

2. In the second drop-down list (2), select the Defender type. Both Linux and Windows platforms are supported.

3. In the third drop-down list (3), leave the listener type set to *None*.

4. In the final field (4), copy the install command, which is generated according to the options you selected.

**STEP 4 |** On the host where you want to install Defender, paste the command into a shell window, and run it.

**Verify the install**

Verify that Defender is installed and connected to Console.

*Defender can be deployed and run with full functionality when dockerd is configured with SELinux enabled (--selinux-enabled=true). All features will work normally and without any additional configuration steps required. Prisma Cloud automatically detects the SELinux configuration on a per-host basis and self-configures itself as needed. No action is needed from the user.*

In Console, go to Manage > Defenders > Manage.
Your new Defender should be listed in the table, and the status box should be green and checked.

Install Container Defender in a cluster

Container orchestrators provide native capabilities for deploying agents, such as Defender, to every node in the cluster. Prisma Cloud leverages these capabilities to install Defender.

The process for deploying Container Defender to a cluster can be found in the dedicated orchestrator-specific install guides.

If you don't have kubectl access to your cluster (or oc access for OpenShift), you can deploy Defender DaemonSets directly from the Console UI.

Prerequisites:

- You've created a kubeconfig credential for your cluster so that Prisma Cloud can access it to deploy the Defender DaemonSet.

**STEP 1** | Log into Prisma Cloud Console.

**STEP 2** | Go to Manage > Defenders > Manage.

**STEP 3** | Click DaemonSets.

**STEP 4** | For each cluster in the table, click Actions > Deploy.

The table shows a count of deployed Defenders and their version number.

> This Defender install flow doesn’t let you manually configure a cluster name. Cluster names let you segment your views of the environment. For most cases, this shouldn’t be a problem because if you’re deploying to a managed cluster, then Prisma Cloud retrieves the cluster name directly from the cloud provider. If you must manually specify a name, deploy your Defenders from Manage > Defenders > Deploy > DaemonSet or use twistcli.
App Embedded Defender

App Embedded Defenders monitor your containers to ensure they execute as designed, protecting them from suspicious processes and outbound network connections. See the article on Defender types to learn when to deploy App Embedded Defenders.

App Embedded Defender policies let you define:

- Process allow or deny lists. Enables verification of launched processes against policy.
- Outgoing connections allow or deny lists. Enables verification of domain name resolution against policy for outgoing network connections.

Besides runtime policy, you can also configure the WAAS application firewall to protect front-end containers.

App Embedded Defender is the only supported option for securing containers at runtime when you're using nested virtualization Nested virtualization is also known as Docker-in-Docker. Docker-in-Docker is a setup where you have a Docker container that itself has Docker installed, and from within the container you use Docker to pull images, build images, run containers, and so on. To secure the containers inside a container, use App Embedded Defender.

Securing containers

To secure a container, embed the App Embedded Defender into it. You can embed App Embedded Defenders with the Console UI, twistcli, or Prisma Cloud API. App Embedded Defender has been tested on Azure Container Instances, DC/OS (unified container runtime), PCF PAS, and Google Serverless containers.

The steps are:
1. Define your policy in Prisma Cloud Console.
2. Embed the App Embedded Defender into the container.
3. Start the service.

The embed process takes a Dockerfile as input, and returns a ZIP file with an augmented Dockerfile and App Embedded Defender binaries. Rebuild your container image with the new Dockerfile to complete the embedding process. The embed process modifies the container's entrypoint to run App Embedded Defender, which in turn starts the original entrypoint program.

When embedding App Embedded Defender, specify a unique identifier for your container image. This gives you a way to uniquely identify the App Embedded Defender in the environment.

When securing your apps with runtime rules, target rules to apps using the App ID. (Because the App Embedded Defender runs inside the container, it can't reliably get information such as image and container names.)
Embed App Embedded Defender

Embed App Embedded Defender into a container image from Console’s UI.

**Prerequisites:**
- The container where you're embedding App Embedded Defender can reach Console’s port 8084 over the network.
- You have the Dockerfile for your image.

**STEP 1** | Open Console, and go to **Manage > Defenders > Deploy**.

**STEP 2** | In the first drop-down list, select the DNS name or IP address that App Embedded Defender uses to connect to Console.

**STEP 3** | In the second drop-down list, select the App Embedded Defender type.

**STEP 4** | In **Deployment Type**, select **Dockerfile**.
**Embed App Embedded Defender manually**

Embed App Embedded Defender into a container image manually. Modify your Dockerfile with the supplied information, download the App Embedded Defender binaries into the image’s build context, then rebuild the image.

**Prerequisites:**
- The container where you’re embedding App Embedded Defender can reach Console over the network on port 8084.
- The host where you’re updating your container image with App Embedded Defender can reach Console over the network on port 8083.
- You have the Dockerfile for your image.

**STEP 1** | Open Console, and go to **Manage > Defenders > Deploy**.

**STEP 2** | In the first drop-down list, select the DNS name or IP address that App Embedded Defender uses to connect to Console.

**STEP 3** | In the second drop-down list, select the App Embedded Defender type.

**STEP 4** | In **Deployment Type**, select **Manual**. A set of instructions for embedding App Embedded Defender into your images is provided.

1. Download the App Embedded Defender binaries into the directory that holds your image’s build context.

```bash
$ curl -u <username> https://<CONSOLE>:8083/api/v1/images/twistl<CONSOLE>
```

2. Retrieve the keys App Embedded Defender needs to connect to Console. This value will be set as the value for the INSTALL_BUNDLE environment variable in your Dockerfile.

```bash
$ curl -k -u <CONSOLE_ADMIN_USER> https://<CONSOLE>:8083/api/v1/defenders/install-bundle
```
The curl command returns a JSON object:

```
{"bundle":"eyJj..."}
```

Set INSTALL_BUNDLE to the value for bundle. For example:

```
ENV INSTALL_BUNDLE="eyJj..."
```

3. Open your Dockerfile for editing.
4. In the Dockerfile, add the App Embedded Defender to the image.

```
ADD twistlock_defender_app_embedded.tar.gz /twistlock/
```

5. In the Dockerfile, add the specified environment variables. Replace the values for <DEFENDER_APP_ID>, <CONSOLE>, and <INSTALL_BUNDLE>.

```
ENV DEFENDER_TYPE="appEmbedded"
ENV DEFENDER_APP_ID="my-app"
ENV WS_ADDRESS="wss://<CONSOLE>:8084"
ENV DATA_FOLDER="/twistlock/"
ENV INSTALL_BUNDLE=""
```

6. Modify the run or entrypoint command such that the command that starts your app is an argument to App Embedded Defender. For example, to start the hello program under the control of App Embedded Defender, specify the following entrypoint.

```
ENTRYPOINT ["/twistlock/defender", "app-embedded", "hello"]
```

**STEP 5** | Rebuild your image.

```
$ docker build .
```

**STEP 6** | Tag and push the updated image to your repository.

*Embed App Embedded Defender with twistcli*

Prisma Cloud supports automation for embedding App Embedded Defender into container images with either twistcli or the API. This section shows you how to use twistcli. To learn how to use the API, see the API docs.

**Prerequisites:**
- The container where you're embedding App Embedded Defender can reach Console's port 8084 over the network.
- You have the Dockerfile for your image.

**STEP 1** | Download twistcli.

1. Log into Console, and go to Manage > System > Downloads.
2. Download the twistcli binary for your platform.

**STEP 2** | Generate the artifacts for an updated container with twistcli. A file named `app_embedded_embed_help.zip` is created.

```
$ ./twistcli app-embedded embed \
--user <USER> \
--address "https://<CONSOLE>:8083" \
```
--console-host "<CONSOLE>" \
--app-id "<DEFENDER-ID>" \
--data-folder "<DATA-FOLDER>"

Dockerfile

- **<USER>** --
  Name of a Prisma Cloud user with a minimum role of Defender Manager.
- **<CONSOLE>** --
  DNS name or IP address for Console.
- **<APP-ID>** --
  Unique identifier for the App Embedded Defender. For example, my-app.
- **<DATA-FOLDER>** --
  Readable and writable directory in the container's filesystem. For example, /twistlock/.

**STEP 3 | Unpack `app_embedded_embed_help.zip`.**

```bash
$ mkdir tmp
$ unzip app_embedded_embed_help.zip -d tmp/
```

**STEP 4 | Build the updated image.**

```bash
$ cd tmp/
$ docker build .
```

**STEP 5 | Tag and push the updated image to your repository.**

**App Embedded Defender for Fargate**

App Embedded Defenders for Fargate monitor your tasks to ensure they execute as designed, protecting tasks from suspicious processes and outbound network connections.

App Embedded Defender policies let you define:

- Process allow or deny lists. Enables verification of launched processes against policy.
- Outgoing connections allow or deny lists. Enables verification of domain name resolution against policy for outgoing network connections.

Besides runtime policy, you can also configure the WAAS application firewall to protect front-end Fargate tasks.

**Architecture**

When you embed the App Embedded Defender into your Fargate task, Prisma Cloud modifies the task definition. The updated task definition includes a Prisma Cloud sidecar container. The sidecar container handles all communication with Console, including retrieving policies and sending audits. It also hosts the App Embedded Defender binaries, which are shared with the task's other containers through a shared volume. The embed process modifies each containerDefinition to:

- Mount the Prisma Cloud sidecar container's shared volume to gain access to the App Embedded Defender binaries.
- Start the original entrypoint command under the control of App Embedded Defender.

App Embedded Defenders do not communicate directly with Console. All communication is proxied through the Prisma Cloud sidecar container. The following diagram illustrates the setup:
WAAS for Fargate

All the capabilities of standard WAAS are available for Fargate tasks. The only difference is that Fargate Defenders run as a reverse proxies to all other containers in the task. As such, when you set up WAAS for Fargate, you must specify the exposed external port where Fargate Defender can listen, and the port (not exposed to the Internet) where your web application listens. WAAS for Fargate forwards the filtered traffic to your application port - unless an attack is detected and you chose Prevent in your WAAS for Fargate rule.

For more information on the type of attacks that Prisma Cloud detects and prevents, see Prisma Cloud WAAS.

Securing Fargate tasks

To secure a Fargate task, embed the Prisma Cloud Fargate Defender into it. The steps are:

1. Define your policy in Prisma Cloud Console. By default, there are no rules in the App Embedded runtime policy. App Embedded Defenders dynamically retrieve policies from Console as they are updated. You can embed the App Embedded Defender into a task with very simple initial policies, then refine them later as needed.
2. Embed the Fargate Defender into your task definition.
3. Start the service.

When securing Fargate tasks with runtime rules and WAAS, target rules to tasks using the Scope fields. For runtime, scope rules by image and container name. Policy is applied per-container in the task.
For WAAS, scope rules by App ID. Policy is applied per-task. The WAAS firewall listens on a specific port, and since all containers run in the same network namespace, it applies to the entire task.
Embed App Embedded Defender into Fargate tasks

Prisma Cloud cleanly separates the code developers produce from the Fargate containers we protect. Developers don’t need to change their code to accommodate Prisma Cloud. They don’t need to load any special libraries, add any files, or change any manifests. When a container is ready to be deployed to test or production, run your task definition through a transform tool to automatically embed the Fargate Defender, then load the new task definition into AWS.

The method for embedding the Fargate Defender was designed to seamlessly integrate into the CI/CD pipeline. You can call the Prisma Cloud API to embed the Fargate Defender into your task definition.

Prerequisites:
- The task where you’re embedding the App Embedded Defender can reach Console’s port 8084 over the network.
- You have the task definition.
Your task definition must include matching `entrypoint` and `cmd` parameters from the Dockerfile(s) of the image(s) in your task. Because Prisma Cloud does not see the actual images as part of the embedding flow, it depends on having these parameter present to reliably insert the App Embedded Defender into the task startup flow. If your Dockerfile does not include an `entrypoint` parameter, a default one, such as `/bin/sh`, must be used in the task definition. However, because the `cmd` parameter is optional, if your Dockerfile does not include a `cmd` parameter, one is not required in the task definition.

**STEP 1** | Log into Prisma Cloud Console.

**STEP 2** | Go to Manage > Defenders > Deploy.

**STEP 3** | In the first drop-down list, choose the name or IP address App Embedded Defender should use to connect to Console.

- A list of IP addresses and hostnames are pre-populated in the drop-down list. If none of the items are valid, select the **Names** tab and add a new subject alternative name (SAN) using **Add SAN** button. After adding a SAN, your IP address or hostname will be available in the drop-down list in the Deploy tab.

- Selecting an IP address in a evaluation setup is acceptable, but using a DNS name is more resilient. If you select Console’s IP address, and Console’s IP address changes, your Defenders will no longer be able to communicate with Console.

**STEP 4** | In the second drop-down list, choose the **Defender type** of App Embedded.

**STEP 5** | Set the **Deployment type** to Fargate Task.

**STEP 6** | Embed the Fargate Defender into your task definition.

1. Copy and paste your task definition into the left-hand box.
2. Click **Generate Protected Task**.
3. Copy the updated task definition from the right-hand box.

**STEP 7** | In AWS, create a new task definition using the new Prisma Cloud protected task. If you already have an existing task definition, create a new revision.

After running your task, view audits in Prisma Cloud Console. Go to Monitor > Events, and select **App Embedded Audits**.

**VMWare Tanzu TAS (PCF) Defender**

You can deploy a dedicated Defender on all Diego cells (Hosts) in your environment. PCF Defender supports the following functions:

- Vulnerability scanning for running apps.
- Vulnerability and compliance scanning for the underlying Diego cell hosts.
- Blobstore scanning for buildpack-based apps.
- Runtime protection (process, networking, and file system).

Defender is deployed as BOSH Director addon. Addons are BOSH release jobs that run on each Diego cell host. Defender runs as a service under the root user. The Defender service is monitored by the Bosh agent, with the help of Monit. Note that the Defender service isn’t a Garden container.

Console lets you deploy Defender to multiple TAS environments. In Console, Defenders report which Cloud Controller they report to.
Currently, PCF Defender doesn't support blocking for runtime rules, vulnerability rules, and compliance rules. The block action stops the entire container. The app lifecycle is controlled by the Tanzu TAS framework, so Prisma Cloud cannot effectively block running apps. PCF Defender, however, does support the prevent action.

PCF Defender currently doesn't support custom compliance checks.

Install the PCF Defender

Go to the PCF Ops Manager Installation Dashboard to install PCF Defender.

**IMPORTANT:** If you're upgrading from a previous release, you must first uninstall any previous versions of PCF Defender. The new 20-09 tile has been rearchitected. The old tile created a dedicated VM in the TAS environment with a Defender installed, and supported blobstore scanning only. The new tile installs Defender on every Diego cell in the TAS environment, with expanded support for app scanning, host scanning, and runtime defense.

Prerequisites:
- Prisma Cloud Console has already been installed. One option is to install Console on Pivotal Container Service (PKS), although there are many options, including Onebox.

**STEP 1** In Prisma Cloud Console, go to Manage > System > Downloads, and download the PCF tile.

**STEP 2** In the Ops Manager Installation Dashboard, click Import a Product, and select the tile you downloaded.

**STEP 3** Retrieve the install command from Prisma Cloud Console. It's used to configure the tile.
1. Go to Manage > Defenders > Deploy > Single Defender.
2. Choose the DNS name or IP address the PCF Defender will use to connect to Console. If a suitable option is not available, go to Manage > Defenders > Names, and add a DNS name or IP address to the SAN table.
3. Set the Defender type to **PCF Defender**.
4. Copy the install command and set it aside.

**STEP 4 |** Go to the PCF Ops Manager Installation Dashboard.

**STEP 5 |** Add the Prisma Cloud tile to your staging area. Click the + button next to the version of the tile you want to install.

**STEP 6 |** Click the newly added **Prisma Cloud for PCF** tile.

**STEP 7 |** Configure the tile.

1. In **Prisma Cloud Component Configuration**, paste the install command you copied from Prisma Cloud Console, then click **Save**.

   By default Prisma Cloud performs strict validation of your Cloud Controller's (CC) TLS certificate. If you're using self-signed certificates, this check will fail. To add your custom certificates to trusted cert list, you need to add the custom CA's cert on the VM where the Prisma Cloud tile runs. For more details about how to do this, refer to Pivotal's trusted certificates article.

   To skip strict validation of your Cloud Controller's (CC) TLS certificate, enable **Skip Cloud Controller TLS validation**. Strict validation verifies the name, signer, and validity date of the CC's certificate. Even with strict validation disabled, the session is still encrypted. Skip strict validation when:
   - You're using self-signed certificates.
   - You're using certificates signed by a CA that isn't in your cert store.
   - When there's a mismatch between the address you're using to connect to the CC and the common name (CN) or subject alternative name (SAN) in the CC's certificate.

2. In **Credentials**, select your preferred authentication method: Basic Authentication or Certificate-based Authentication:

   For Basic Authentication, enter your Prisma Cloud Console credentials, then click **Save**.

   For certificate-based Authentication, paste the certificate and private key used for authentication in PEM format, then click **Save**.

   Notes:
   - Your role must be Defender Manager or higher.
   - For Certificate-based Authentication, the root CA used to sign the certificate used for authentication must be entered under Manage > Authentication > System Certificates > Advanced Certificate Configuration.

**STEP 8 |** Install the Prisma Cloud tile. Return to the Ops Manager Installation Dashboard, click **Review Pending Changes**, select both **Prisma Cloud for PCF** and **Pivotal Application Service**, then click **Apply changes**.

   *Pivotal Application Service must be staged when installing the Prisma Cloud tile.*

**STEP 9 |** After the changes are applied, validate that Prisma Cloud Defenders are running in your environment.

1. Log into Prisma Cloud Console.
2. Go to Manage > Defenders > Manage.

   In the table of deployed Defenders, you should see a Defender of type **PCF**, one per Diego cell.
Prisma Cloud reports the agentID in the Host field. To correlate an agentID to a Diego cell IP address, and determine exactly which host runs a Defender, login to an Diego cell, and inspect /var/vcap/instance/dns/records.json. This file shows how the agentID maps to a host IP address.

If a PCF Defender disconnects from Console for more than one day, all data it collected is purged from Console. The Defender is also removed from the table in Manage > Defenders > Manage. The period of time that data from a disconnected Defender is retained (by default, one day) can be configured in Manage > Defenders > Manage > Defenders > Advanced Settings.

Serverless Defender (auto-protect)

Serverless auto-protect lets you automatically add the Serverless Defender to the AWS Lambda functions deployed in your account. Prisma Cloud uses the AWS API to deploy the Serverless Defender as a Lambda layer based on the auto-protect rules.

It is an additional option for deploying the Serverless Defender, on top of manually adding it as a dependency or adding it as a Lambda layer.

Serverless auto-protect supports the following runtimes:

- Node.js 10.x
- Python 2.7, 3.6, and 3.7

Limitations

Auto-protect is implemented with a layer. AWS Lambda has a limit of five layers per function. If your functions have multiple layers, and they might exceed the layer limit with auto-protect, consider protecting them with the embedded option.
Required permissions

Prisma Cloud needs the following permissions to automatically protect Lambda functions in your AWS account. Add the following policy to an IAM user or role:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "PrismaCloudServerlessAutoProtect",
         "Effect": "Allow",
         "Action": [
            "lambda:PublishLayerVersion",
            "lambda:UpdateFunctionConfiguration",
            "lambda:GetLayerVersion",
            "lambda:GetFunctionConfiguration",
            "iam:SimulatePrincipalPolicy"
         ],
         "Resource": "*"
      }
   ]
}
```

Serverless auto-protect rules

To secure one or multiple AWS Lambda functions using serverless auto-protect:

1. Define a serverless protection runtime policy.
2. Define a serverless WAAS policy.
3. Add a serverless auto-protect rule.

Defining your runtime protection policy

Prisma Cloud ships with a default runtime policy for all serverless functions that blocks all processes from running except the main process. This default policy protects against command injection attacks.

You can customize the policy with additional rules. By default, new rules apply to all functions (*), but you can target them to specific functions by function name.

When functions are invoked, they connect to Compute Console and retrieve the latest policy. To ensure that functions start executing at time=0 with your custom policy, you must redefine the policy. Predefined policy is embedded into your function along with the Serverless Defender by way of the `TW_POLICY` environment variable.

**STEP 1** | Log into Prisma Cloud Console.
**STEP 2** | Go to Defend > Runtime > Serverless Policy.
**STEP 3** | Click Add rule.
**STEP 4** | In the General tab, enter a rule name.
**STEP 5** | (Optional) Target the rule to specific functions.
   In Functions, enter a function name. Use pattern matching to refine how the rule is applied.
**STEP 6** | Set the rule parameters in the Processes, Networking, and File System tabs.
STEP 7 | Click Save.

Defining your serverless WAAS policy

Prisma Cloud lets you protect your serverless functions against application layer attacks by utilizing the serverless Web Application and API Security (WAAS).

By default, the serverless WAAS is disabled. To enable it, add a new serverless WAAS rule.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to Defend > Firewalls > Cloud Native Application Firewall > Serverless.

STEP 3 | Click Add rule.

STEP 4 | In the General tab, enter a rule name.

STEP 5 | (Optional) Target the rule to specific functions.

In Functions, enter a function name. Use pattern matching to refine how the rule is applied.

STEP 6 | Set the protections you want to apply (SQLi, CMDi, Code injection, XSS, LFI).

STEP 7 | Click Save.

Add a serverless auto-protect rule

The serverless auto-protect rules let you specify which functions you want to protect. You can define a specific account by referencing the relevant credential, region, function pattern name, and runtimes. Each auto-protect rule is evaluated separately.

STEP 1 | Open Compute Console, and go to Manage > Defenders > Deploy > Serverless Auto-Protect.

STEP 2 | Click on Add rule.

STEP 3 | In the dialog, enter the following settings:

1. Choose the rule name.
2. In Provider - only AWS is supported.
3. Specify a region.
4. Specify a console name.
5. Select or create credentials so that Prisma Cloud can access your account.
   - AWS – Specify either an IAM user credential (access key ID and secret access key) or IAM role.
6. Specify the functions.
7. Wildcards are supported.
8. Specify the runtimes.
9. Click Add.

STEP 4 | The new rule should now appear in the Rules table.

STEP 5 | Click on Apply.

By default, the serverless auto-protect rules are evaluated every 24 hours.
Serverless Defender

Serverless Defender protects serverless functions at runtime. It monitors your functions to ensure they execute as designed.

Per-function policies let you control:

- Process activity. Enables verification of launched subprocesses against policy.
- Network connections. Enables verification of inbound and outbound connections, and permits outbound connections to explicitly allowed domains.
- File system activity. Controls which parts of the file system functions can access.

Currently, Prisma Cloud supports AWS Lambda functions. The following runtimes are supported:

- C# (.NET Core 2.1)
- Java 8
- Node.js 10.x
- Python 2.7, 3.6, and 3.7

Currently, only users with the Administrator role can see the list of deployed Serverless Defenders in Manage > Defenders > Manage.

Securing serverless functions

To secure an AWS Lambda function, embed the Prisma Cloud Serverless Defender into it. The steps are:

1. (Optional) If you are not using a deployment framework like SAM or Serverless Framework, download a ZIP file that contains your function source code and dependencies.
2. Embed the Serverless Defender into the function.
3. Deploy the new function or upload the updated ZIP file to AWS.
4. Define a serverless protection runtime policy.
5. Define a serverless WAAS policy.

(Optional) Download your function as a ZIP file

Download your function’s source code from AWS as a ZIP file.

STEP 1 | From Lambda’s code editor, click Actions > Export function.

STEP 2 | Click Download deployment package.

Your function is downloaded to your host as a ZIP file.

STEP 3 | Create a working directory, and unpack the ZIP file there.

In the next step, you’ll download the Serverless Defender files to this working directory.

Embed Serverless Defender into C# functions

In your function code, import the Serverless Defender library and create a new protected handler that wraps the original handler. The protected handler will be called by AWS when your function is invoked. Update the project configuration file to add Prisma Cloud dependencies and package references.

Prisma Cloud supports .NET Core 2.1.

STEP 1 | Open Compute Console, and go to Manage > Defenders > Deploy > Single Defender.

STEP 2 | Choose the DNS name or IP address Serverless Defender uses to connect to Console.
STEP 3 | **In Choose Defender type**, select **Serverless**.

STEP 4 | **In Runtime**, select **C#**.

STEP 5 | **Download the Serverless Defender package to your workstation.**

STEP 6 | **Unzip the Serverless Defender bundle into your working directory.**

STEP 7 | **Embed the serverless Defender into the function by importing the Prisma Cloud library and wrapping the function’s handler.**

  Function input and output can be a struct or a stream. Functions can be synchronous or asynchronous. The context parameter is optional in .NET, so it can be omitted.

```csharp
using Twistlock;

public class ... {
    // Original handler
    public ApplicationLoadBalancerResponse Handler(ApplicationLoadBalancerRequest request, ILambdaContext context) {
        ...
    }

    // Application load balancer example
    // Twistlock protected handler
    public ApplicationLoadBalancerResponse ProtectedHandler(ApplicationLoadBalancerRequest request, ILambdaContext context) {
        return Twistlock.Serverless.Handler<ApplicationLoadBalancerRequest, ApplicationLoadBalancerResponse>(Handler, request, context);
    }
    ...
}
```

STEP 8 | **Add the Twistlock package as a dependency in your nuget.config file.**

  If a nuget.config file doesn’t exist, create one.

```xml
<configuration>
    <packageSources>
        <add key="local-packages" value="./twistlock"/>
    </packageSources>
</configuration>
```

STEP 9 | **Reference the Twistlock package in your csproj file.**

```xml
<Project>
    <ItemGroup>
        <PackageReference Include="Twistlock" Version="19.11.462"/>
        <TwistlockFiles Include="twistlock/**" Exclude="twistlock/twistlock.19.11.462.nupkg"/>
    </ItemGroup>
    <Target Name="CopyCustomContentOnPublish" AfterTargets="Publish">
        <Copy SourceFiles="@{(TwistlockFiles)}" DestinationFolder="$(PublishDir)/twistlock"/>
    </Target>
</Project>
```
STEP 10 | Generate the value for the TW_POLICY environment variable by specifying your function's name.

Serverless Defender uses TW_POLICY to determine how to connect to Compute Console to retrieve policy and send audits.

Copy the value generated for TW_POLICY, and set it aside.

STEP 11 | Upload the protected function to AWS, and set the TW_POLICY environment variable.

**Embed Serverless Defender into Java functions**

To embed Serverless Defender, import the Twistlock package and update your code to start Serverless Defender as soon as the function is invoked. Prisma Cloud supports both Maven and Gradle projects. You'll also need to update your project metadata to include Serverless Defender dependencies.

Prisma Cloud supports both predefined interfaces in the AWS Lambda Java core library: RequestStreamHandler (where input must be serialized JSON) and RequestHandler.

AWS lets you specify handlers as functions or classes. In both cases, Twistlock.Handler(), the entry point to Serverless Defender, assumes the entry point to your code is named handleRequest. After embedding Serverless Defender, update the name of the handler registered with AWS to be the wrapper method that calls Twistlock.Handler() (for example, protectedHandler).

Prisma Cloud supports both service struct and stream input (serialized struct). Even though the Context parameter is optional for unprotected functions, it's mandatory when embedding Serverless Defender.

Prisma Cloud supports Java 8.

STEP 1 | Open Compute Console, and go to Manage > Defenders > Deploy > Single Defender.

STEP 2 | Choose the DNS name or IP address Serverless Defender uses to connect to Console.

STEP 3 | In Choose Defender type, select Serverless.

STEP 4 | In Runtime, select Java.

STEP 5 | In Package, select Maven or Gradle.

The steps for embedding Serverless Defender differ depending on the build tool.

STEP 6 | Download the Serverless Defender package to your workstation.

STEP 7 | Unzip the Serverless Defender bundle into your working directory.

STEP 8 | Embed Serverless Defender into your function by importing the Prisma Cloud package and wrapping the function's handler.

```java
import com.twistlock.serverless.Twistlock;

public class ... implements RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {
```
STEP 9 | Update your project configuration file.

1. **Maven**

   Update your pom.xml file. Don’t create new sections for the Prisma Cloud configurations. Just update existing sections. For example, don’t create a new `<plugins>` section if one exists already. Just append a `<plugin>` section to it.

   Add the assembly plugin to include the Twistlock package in the final function JAR. Usually the shade plugin is used in AWS to include packages to standalone JARs, but it doesn’t let you include local system packages.

```xml
<project>
  <build>
    <!-- Add assembly plugin to create a standalone jar that contains Twistlock library -->
    <plugins>
      <plugin>
        <artifactId>maven-assembly-plugin</artifactId>
        <configuration>
          <appendAssemblyId>false</appendAssemblyId>
          <descriptors>
            <descriptor>assembly.xml</descriptor>
          </descriptors>
        </configuration>
        <executions>
          <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
              <goal>attached</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>

    <!-- Add Twistlock resources -->
    <resources>
      <resource>
        <directory>${project.basedir}</directory>
        <includes>
```

2. Create an assembly.xml file, which packs all dependencies in a standalone JAR.

```xml
<assembly>
  <id>twistlock-protected</id>
  <formats>
    <format>jar</format>
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <dependencySets>
    <!-- Unpack runtime dependencies into runtime jar -->
    <dependencySet>
      <unpack>true</unpack>
      <scope>runtime</scope>
    </dependencySet>
    <!-- Unpack local system dependencies into runtime jar -->
    <dependencySet>
      <unpack>true</unpack>
      <scope>system</scope>
    </dependencySet>
  </dependencySets>
</assembly>
```

**STEP 10 | Gradle**

Update your build.gradle file.

1. Add Twistlock package reference in the project configuration file i.e build.gradle

```groovy
dependencies {
  compile {
    files('twistlock/twistlock-19.11.462.jar')
  }
}
```

```groovy
task buildZip(type: Zip) {
```
STEP 11 | In AWS, set the name of the Lambda handler for your function to protectedHandler.

STEP 12 | Generate the value for the TW_POLICY environment variable by specifying your function’s name.

Serverless Defender uses TW_POLICY to determine how to connect to Compute Console to retrieve policy and send audits.

Copy the value generated for TW_POLICY, and set it aside.

STEP 13 | Upload the protected function to AWS, and set the TW_POLICY environment variable.

**Embed Serverless Defender into Node.js functions**

Import the Serverless Defender module, and configure your function to start it. Prisma Cloud supports Node.js 10.x.

STEP 1 | Open Compute Console, and go to Manage > Defenders > Deploy > Single Defender.

STEP 2 | Choose the DNS name or IP address Serverless Defender uses to connect to Console.

STEP 3 | In Choose Defender type, select Serverless.

STEP 4 | In Runtime, select Node.js.

STEP 5 | Download the Serverless Defender package to your workstation.

STEP 6 | Unzip the Serverless Defender bundle into your working directory.

STEP 7 | Embed the serverless Defender into the function by importing the Prisma Cloud library and wrapping the function’s handler.

1. For asynchronous handlers:

```java
// Async handler
var twistlock = require('./twistlock');
extports.handler = async (event, context) => {
    .
    
};
extports.handler = twistlock.asyncHandler(exports.handler);
```
2. For synchronous handlers:

```javascript
// Non-async handler
var twistlock = require('./twistlock');
exports.handler = (event, context, callback) => {
  
  
};
exports.handler = twistlock.handler(exports.handler);
```

STEP 8 | Generate the value for the TW_POLICY environment variable by specifying your function’s name.

Serverless Defender uses TW_POLICY to determine how to connect to Compute Console to retrieve policy and send audits.

Copy the value generated for TW_POLICY, and set it aside.

STEP 9 | Upload the protected function to AWS, and set the TW_POLICY environment variable.

Embed Serverless Defender into Python functions

Import the Serverless Defender module, and configure your function to invoke it. Prisma Cloud supports Python 2.7, 3.6, and 3.7.

STEP 1 | Open Compute Console, and go to Manage > Defenders > Deploy > Single Defender.

STEP 2 | Choose the DNS name or IP address Serverless Defender uses to connect to Console.

STEP 3 | In Choose Defender type, select Serverless.

STEP 4 | In Runtime, select Python.

STEP 5 | Download the Serverless Defender package to your workstation.

STEP 6 | Unzip the Serverless Defender bundle into your working directory.

STEP 7 | Embed the serverless Defender into the function by importing the Prisma Cloud library and wrapping the function’s handler.

```
import twistlock.serverless
@twistlock.serverless.handler
def handler(event, context):
  
  
```

STEP 8 | Generate the value for the TW_POLICY environment variable by specifying your function’s name.

Serverless Defender uses TW_POLICY to determine how to connect to Compute Console to retrieve policy and send audits.

Copy the value generated for TW_POLICY, and set it aside.

STEP 9 | Upload the protected function to AWS, and set the TW_POLICY environment variable.
• Prisma Cloud Serverless Defender includes native node.js libraries. If you are using webpack, please refer to tools such as native-addon-loader to make sure these libraries are included in the function ZIP file.

**Upload the protected function to AWS**

After embedding Serverless Defender into your function, upload it to AWS. If you are using a deployment framework such as SAM or Serverless Framework just deploy the function with your standard deployment procedure. If you are using AWS directly, follow the steps below:

**STEP 1 | Upload the new ZIP file to AWS.**

1. In **Designer**, select your function so that you can view the function code.
2. Under **Code entry type**, select **Upload a .ZIP file**.
3. Specify a runtime and the handler.
   - Validate that **Runtime** is a supported runtime, and that **Handler** points to the function's entry point.
4. Click **Upload**.

5. Click **Save**.

**STEP 2 | Set the TW_POLICY environment variable.**

1. In Designer, open the environment variables panel.
2. For **Key**, enter **TW_POLICY**.
3. For **Value**, paste the rule you copied from Compute Console.
4. Click **Save**.

**Defining your runtime protection policy**

Prisma Cloud ships with a default runtime policy for all serverless functions that blocks all processes from running except the main process. This default policy protects against command injection attacks.

You can customize the policy with additional rules. By default, new rules apply to all functions (*), but you can target them to specific functions by function name.

When functions are invoked, they connect to Compute Console and retrieve the latest policy. To ensure that functions start executing at time=0 with your custom policy, you must predefined the policy. Predefined policy is embedded into your function along with the Serverless Defender by way of the **TW_POLICY** environment variable.
Defining your serverless WAAS policy

Prisma Cloud lets you protect your serverless functions against application layer attacks by utilizing the serverless Web Application and API Security (WAAS).

By default, the serverless WAAS is disabled. To enable it, add a new serverless WAAS rule.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to Defend > Runtime > Serverless Policy.

STEP 3 | Click Add rule.

STEP 4 | In the General tab, enter a rule name.

STEP 5 | (Optional) Target the rule to specific functions.

In Functions, enter a function name. Use pattern matching to refine how the rule is applied.

STEP 6 | Set the rule parameters in the Processes, Networking, and File System tabs.

STEP 7 | Click Save.

Serverless Defender as a Lambda layer

Prisma Cloud Serverless Defenders protect serverless functions at runtime. Currently, Prisma Cloud supports AWS Lambda functions.

Lambda layers are ZIP archives that contain libraries, custom runtimes, or other dependencies. Layers let you add reusable components to your functions, and focus deployment packages on business logic. They are extracted to the /opt directory in the function execution environment. For more information, see the AWS Lambda layers documentation.

Prisma Cloud delivers Serverless Defender as a Lambda layer. Deploy Serverless Defender to your function by wrapping the handler and setting an environment variable.

The following runtimes are supported for Serverless Defender as a Lambda layer:

- Node.js 10.x
- Python 2.7, 3.6, and 3.7
Securing serverless functions

To secure an AWS Lambda function with the Serverless Defender layer:
1. Download the Serverless Defender Lambda layer ZIP file.
2. Upload the layer to AWS.
3. Define a serverless protection runtime policy.
4. Define a serverless WAAS policy.
5. Add the layer to your function, update the handler, and set an environment variable. After completing this integration, Serverless Defender runs when your function is invoked.

Download the Serverless Defender layer

Download the Serverless Defender layer from Compute Console.

**STEP 1 |** Open Console, then go to Manage > Defenders > Deploy.

**STEP 2 |** Choose the DNS name or IP address that Serverless Defender uses to connect to Console.

**STEP 3 |** Set the Defender type to Serverless.

**STEP 4 |** Select a runtime.

Prisma Cloud supports Lambda layers for Node.js and Python only.

**STEP 5 |** For Deployment Type, select Layer.

**STEP 6 |** Download the Serverless Defender layer. A ZIP file is downloaded to your host.

Upload the Serverless Defender layer to AWS

Add the layer to the AWS Lambda service as a resource available to all functions.

**STEP 1 |** In the AWS Management Console, go to the Lambda service.

**STEP 2 |** Click Layers.

**STEP 3 |** In Name, enter twistlock.

**STEP 4 |** Click Upload, and select the file you just downloaded, twistlock_defender_layer.zip

1. Select the compatible runtimes: Python or Node.js.
2. Click Create.
Defining your runtime protection policy

Prisma Cloud ships with a default runtime policy for all serverless functions that blocks all processes from running except the main process. This default policy protects against command injection attacks.

You can customize the policy with additional rules. By default, new rules apply to all functions (*), but you can target them to specific functions by function name.

When functions are invoked, they connect to Compute Console and retrieve the latest policy. To ensure that functions start executing at time=0 with your custom policy, you must predefine the policy. Predefined policy is embedded into your function along with the Serverless Defender by way of the TW_POLICY environment variable.

**STEP 1** | Log into Prisma Cloud Console.

**STEP 2** | Go to **Defend > Runtime > Serverless Policy**.

**STEP 3** | Click **Add rule**.
STEP 4 | In the **General** tab, enter a rule name.

STEP 5 | (Optional) Target the rule to specific functions.

   In **Functions**, enter a function name. Use pattern matching to refine how the rule is applied.

STEP 6 | Set the rule parameters in the **Processes**, **Networking**, and **File System** tabs.

STEP 7 | Click **Save**.

---

**Defining your serverless WAAS policy**

Prisma Cloud lets you protect your serverless functions against application layer attacks by utilizing the serverless Web Application and API Security (WAAS).

By default, the serverless WAAS is disabled. To enable it, add a new serverless WAAS rule.

---

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Defend > Firewalls > Cloud Native Application Firewall > Serverless**.

STEP 3 | Click **Add rule**.

STEP 4 | In the **General** tab, enter a rule name.

STEP 5 | (Optional) Target the rule to specific functions.

   In **Functions**, enter a function name. Use pattern matching to refine how the rule is applied.

STEP 6 | Set the protections you want to apply (**SQLi**, **CMDi**, **Code injection**, **XSS**, **LFI**).

STEP 7 | Click **Save**.

---

**Embed the Serverless Defender**

Embed the Serverless Defender as a layer, and run it when your function is invoked. If you are using a deployment framework such as **SAM** or **Serverless Framework** you can reference the layer from within the configuration file.

**Prerequisites:**

- You already have a Lambda function.
- Your Lambda function is written for Node.js 10.x or Python.
- Your function’s execution role grants it permission to write to CloudWatch Logs. Note that the **AWSLambdaBasicExecutionRole** grants permission to write to CloudWatch Logs.

---

STEP 1 | Go to the function designer in the AWS Management Console.

STEP 2 | Click on the **Layers** icon.

---

STEP 3 | In the **Referenced Layers** panel, click **Add a layer**.
1. In the **Select from list of runtime compatible layers**, select **twistlock**.
2. In the **Version** drop-down list, select **1**.
3. Click **Add**.

When you return to the function designer, you'll see that your function now uses one layer.

**STEP 4** | Update the handler for your function to be **twistlock.handler**.
STEP 5 | Set the **TW_POLICY** and **ORIGINAL_HANDLER** environment variable, which specifies how your function connects to Compute Console to retrieve policy and send audits.

1. In Compute Console, go to Manage > Defenders > Deploy > Single Defender.
2. For Defender type, select Serverless.
3. In Set the Twistlock environment variable, enter the function name.
4. Copy the generated Value.
5. In AWS Console, open your function in the designer, and scroll down to the Environment variables panel.
6. For Key, enter TW_POLICY.
7. For Value, paste the rule you copied from Compute Console.
8. For ORIGINAL_HANDLER, this is the original value of handelr for your function before your modification.

STEP 6 | Click Save to preserve all your changes.

---

**Install a Host Defender**

Host Defenders are installed with a curl-bash script. Install Host Defender on each host that you want Prisma Cloud to protect.

**Prerequisites:**
• Your system meets all minimum system requirements.
• You have already installed Console, and it can be accessed over the network from the host where you want to install Defender.
• Ports 8083 and 8084 are open on the host where Console runs. Console and Defender communicate with each other over a web socket on port 8084.
• You have sudo access to the host where Defender will be installed.

STEP 1 | Verify that the host machine where you install Defender can connect to Console.

$ curl -sk -D - https://<CONSOLE_IP_ADDRESS|HOSTNAME>:8083/api/v1/_ping

If curl returns an HTTP response status code of 200, you have connectivity to Console. If you customized the setup when you installed Console, you might need to specify a different port.

STEP 2 | Log into Console.

STEP 3 | Go to Manage > Defenders > Deploy.

1. In the first drop-down menu (1), select the way Defender connects to Console.

   A list of IP addresses and hostnames are pre-populated in the drop-down list. If none of the items are valid, go to Manage > Defenders > Names, and add a new Subject Alternative Name (SAN) to Console's certificate. After adding a SAN, your IP address or hostname will be available in the drop-down list.

   Selecting an IP address in a evaluation setup is acceptable, but using a DNS name is more resilient. If you select Console’s IP address, and Console’s IP address changes, your Defenders will no longer be able to communicate with Console.

2. In the second drop-down list (2), select Host Defender - Linux or Host Defender - Windows.

3. In the final field, copy the install command, which is generated according to the options you selected.

STEP 4 | On the host where you want to install Defender, paste the command into a shell window, and run it.

Verify the install

Verify that Defender is installed and connected to Console.

In Console, go to Manage > Defenders > Manage. Your new Defender should be listed in the table, and the status box should be green and checked.

Deploy Prisma Cloud Defender from the GCP Marketplace

Prerequisites: You need access to a Prisma Cloud SaaS Console. You can sign up for a free trial of Prisma Cloud on the Google Cloud Marketplace.

STEP 1 | Find Prisma Cloud Defender in the GCP Marketplace. Click Configure.
STEP 2 | Create Cluster, if you don't have an existing Kubernetes cluster. Otherwise, continue to the next step.
STEP 3 | Select an existing namespace to install Defender, or Create a namespace (recommended). The default new namespace is “twistlock”.

STEP 4 | Enter the App instance name for the Defender the installation. This name displays on the Application section of the GKE portal:

STEP 5 | Specify the following information about your Prisma Cloud SaaS Console (go through steps 6-8 to get these info):
STEP 6 | To get the URL for your Prisma Cloud Console:

1. Log into your Prisma Cloud portal (e.g., https://app.prismacloud.io/).
2. Navigate to Compute > System.
3. Copy the URL in Path to Console. GCP uses this URL to get all the setup artifacts from your Prisma Cloud Console. In this example, it's https://us-east1.cloud.twistlock.com/us-1-111573360.

STEP 7 | To get a token for your Prisma Cloud Compute Console.

1. Go to Compute > Authentication.
2. Copy the API token, and paste it into the GCP Marketplace form.
STEP 8 | Specify the IP address or domain name of your Prisma Cloud Compute Console.

The Defenders that you are deploying will use this IP address to communicate with Prisma Cloud. It's almost the same as the URL, but remove the protocol (https://) and the path (everything trailing the first “/”). In this example, us-east1.cloud.twistlock.com.

STEP 9 | When the form is filled out, click Deploy.
STEP 10 | Go to Prisma Cloud SaaS Console to confirm the deployment is successful.

1. In the GKE console, review the status of your deployment:

![GKE console screenshot]

2. In Prisma Cloud Console, go to Compute > Defender to review the status of your deployment:

![Prisma Cloud console screenshot]

Decommission Defenders

Regularly decommissioning stale Defenders keeps your view of the environment clean and conserves licenses. Defenders can be decommissioned from the Console UI or the Prisma Cloud API.

Prisma Cloud automatically decommissions stale Defenders for you. In large scale environments, manually decommissioning Defenders could be onerous. If left undone, however, it can lead to lots of Defenders being left in a permanently offline state, cluttering your view of environment. To keep your view clean, Console automatically decommissions Defenders that haven't been connected to Console for more than one day. This keeps the list of connected Defenders valid to a 24-hour window. The refresh period can be configured up to a maximum of 365 days under Manage > Defenders > Manage > Advanced Settings > Automatically remove disconnected Defenders after (days).

We recommend letting Prisma Cloud automatically decommission stale Defenders rather than using the UI or API.

Decommission Defenders manually

Decommissioning Defenders can be done manually from Console.
Go to Manage > Defenders > Manage, where you will find a list of all Defenders connected to Console. Click Actions > Decommission for each respective Defender.

**Decommission Defenders with the API**

The following endpoint can be used to decommission a Defender.

**Path**

```
DELETE /api/v1/defenders/[hostname]
```

**Description**

Deletes a Defender from the database. This endpoint does not actually uninstall Defender. Use the fully qualified domain name (FQDN) of the host. You can find the FQDN of the host in Manage > Defenders > Actions > Manage.

**Example request**

```
$ curl -X DELETE \\
-u <USERNAME>:<PASSWORD> \\
'https://<CONSOLE>:8083/api/v1/defenders/0aqa-cto.sandbox'
```

**Force uninstall Defender**

The preferred method for uninstalling Defenders is via the Console UI. However, if a Defender instance is not connected to Console, or is otherwise not manageable through the Console UI, it can be manually removed.

On the Linux host where Container Defender runs, use the following command:

```
$ sudo /var/lib/twistlock/scripts/twistlock.sh -u
```

*If you run this command on the same Linux host where the Prisma Cloud Console is installed, it also uninstalls Prisma Cloud Console.*

On the Linux host where Host Defender runs, use the following command:

```
$ sudo /var/lib/twistlock/scripts/twistlock.sh -u defender-server
```

On the Windows host where Defender runs, use the following command:

```
C:\Program Files\Prisma Cloud\scripts\defender.ps1 -uninstall
```
Upgrade

Console notifies you when new versions of Prisma Cloud are available. You can upgrade Prisma Cloud without losing any of your data or configurations. After upgrading Console, all your deployed Defenders will automatically upgrade themselves.

- Upgrade Prisma Cloud
- Upgrade Onebox
- Kubernetes
- OpenShift
- Helm charts
- Docker Swarm
- Amazon ECS
- Manually upgrade single Container Defenders
- Manually upgrade Defender DaemonSets
- Manually upgrade Defender DaemonSets (Helm)
- Manually upgrade Docker Swarm Defenders
Upgrade Prisma Cloud

You can upgrade Prisma Cloud without losing any of your data or configurations. Upgrade Console first. After upgrading Console, upgrade your Defenders, and other Prisma Cloud components.

Before upgrading, check the **Breaking changes** section in the release notes to see if there are any special instructions or requirements.

You can upgrade from an immediate previous major version only. If your installation is more than one major release behind, you must upgrade in steps. For example, you cannot directly upgrade from version 18.11 to 19.07. You must upgrade from version 18.11 to 19.03, and then from 19.03 to 19.07.

Console notifies you when new versions of Prisma Cloud are available. Notifications are displayed in the top right corner of the dashboard.

When you upgrade Console, the old Console container is completely replaced with a new container. Because Prisma Cloud stores state information outside of the container, all your rules and settings are immediately available to the upgraded Prisma Cloud containers.

Prisma Cloud state information is stored in a database in the location specified by `DATA_FOLDER`, which is defined in `twistlock.cfg`. By default, the database is located in `/var/lib/twistlock`.

**Overview of the upgrade process**

First upgrade Console. Next, upgrade your Defenders. Finally, upgrade all other Prisma Cloud components, such as the Jenkins plugin. The upgrade process is vastly simplified when automatic Defender upgrades is enabled (it’s enabled by default).

The steps in the upgrade process are:

1. Upgrade Console.
2. Upgrade all deployed Defenders.
   - If Defender auto-upgrade is enabled — Console will upgrade deployed Defenders for you. If Console fails to upgrade one or more Defenders, it displays a banner at the top of the UI. If you've created an alert for Defender health events, Console emits a message on the alert channel for any Defender it fails to upgrade. Manually upgrade any Defenders that Console could not auto-upgrade.
   - If Defender auto-upgrade is disabled — Manually upgrade all deployed Defenders.
3. Validate that all deployed Defenders have been upgraded.
   1. Review deployed Defenders and DaemonSets under **Manage > Defenders > Manage**.
   2. Filter the the **Status** column by **Upgrade**.
   3. If any Defenders have the **Upgrade** status, manually upgrade them.
4. Manually upgrade all other Prisma Cloud Compute components, such as the Jenkins plugin, so that their versions exactly match Console's version.

Version numbers of installed components

The currently installed version of Console is displayed in the bell menu.

The versions of your deployed Defenders are listed under Manage > Defenders > Manage:
Prisma Cloud Compute components

The versions of all deployed components should match exactly. To support the multi-step upgrade process, older versions of Prisma Cloud components can continue to interoperate with newer versions of Console in a limited way. Plan to upgrade all Prisma Cloud components as soon as possible.

After you upgrade Console, upgrade the following components:

- Defenders. Console can automatically upgrade most Defender types for you. App-embedded Defenders and PCF Defenders (also known as Twistlock for Pivotal Platform) must be manually upgraded.
- Jenkins plugin.
- twistcli.
- If you’re using projects, supervisor Consoles must match the Central Console version.

Version mismatches

Console interoperates with older components on a best-effort basis. When older components interact with Console, Console displays some indicators in the dashboard:

- In Monitor > Events, any audits generated by older Defenders are marked with an out-of-date indicator. Links to the rules that triggered the audit are disabled (explanation follows).
- In Monitor > Vulnerabilities and Monitor > Compliance, any scan reports generated by older components (Defender registry scanners, Jenkins plugins, twistcli) are marked with an out-of-date indicator.

Although older Defenders can interoperate with newer Consoles, their operation is restricted. Older Defenders fully protect your nodes using the policies and settings most recently cached before upgrading Console. They can emit audits to Console and local logs, including syslog. However, they cannot access any API endpoint other than the upgrade endpoint, and they cannot share any new data with Console. No new policies or settings can be pushed from Console to older Defenders. When Defender is in this state, its status is shown as ‘Upgrade needed’ in Manage > Defenders > Manage. To restore older Defenders to a fully operation state, upgrade them so that their versions match Console’s version.

Upgrading Console when using projects

When you have one or more tenant or scale Projects, upgrade all Supervisors before upgrading the Central Console. During the upgrade process, there may be periods where the Supervisors appear as disconnected. This is normal, because the Supervisors are disconnected while the upgrade is occurring and Central Console will recheck connectivity every 10 minutes. Within 10 minutes of upgrading all Supervisors and the Central Console, all Supervisors should appear healthy.
Upgrade each Supervisor and then the Central Console using the appropriate procedure:

- Console - Onebox
- Console - Kubernetes
- Console - Open Shift
- Console - Helm
- Console - Docker Swarm
- Console - Amazon ECS

Defender auto-upgrade support

Most Defender types can be auto-upgraded. A handful must still be manually upgraded. The following table summarizes the Defender types, and which ones can be auto-upgraded.

<table>
<thead>
<tr>
<th>Defender type</th>
<th>Auto-upgrade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Defender, which includes:</td>
<td></td>
</tr>
<tr>
<td>• Single Container Defenders</td>
<td>Y</td>
</tr>
<tr>
<td>• Cluster Container Defenders</td>
<td></td>
</tr>
<tr>
<td>• DaemonSets (Kubernetes, OpenShift)</td>
<td></td>
</tr>
<tr>
<td>• Swarm global service</td>
<td></td>
</tr>
<tr>
<td>• DC/OS app</td>
<td></td>
</tr>
<tr>
<td>Serverless Defender</td>
<td>Y* (see Serverless Defender auto-protect)</td>
</tr>
<tr>
<td>App embedded Defender</td>
<td>N</td>
</tr>
<tr>
<td>PCF Defender</td>
<td>N</td>
</tr>
<tr>
<td>Host Defender</td>
<td>Y</td>
</tr>
</tbody>
</table>

Enabling Defender auto-upgrade

By default, Defender auto-upgrade is enabled. You can check and change the setting in Console.

**STEP 1** | Open Prisma Cloud Compute Console.

**STEP 2** | Go to Manage > Defenders > Manage.

**STEP 3** | Click on Advanced Settings.

**STEP 4** | Set Automatically upgrade Defenders to On or Off.
### Advanced Defender settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatically upgrade defenders</td>
<td>On</td>
</tr>
<tr>
<td>Local Defender API port</td>
<td>9998</td>
</tr>
<tr>
<td>Automatically remove disconnected Defenders after (days)</td>
<td>1</td>
</tr>
</tbody>
</table>
Upgrade Onebox

Upgrade Prisma Cloud Onebox. First upgrade Console. Console will then automatically upgrade all deployed Defenders for you.

If you’ve disabled Defender auto-upgrade or if Console fails to upgrade one or more Defenders, manually upgrade your Defenders.

You must manually upgrade App-embedded Defenders and PCF Defenders.

Upgrading Console

To upgrade Console, rerun the install script for the latest version of Prisma Cloud. Use this method for any Console that was originally installed with the `twistlock.sh` script.

**STEP 1** | Download the latest Prisma Cloud release package from Releases.

**STEP 2** | Unpack the downloaded tarball.

Optional: you may wish to unpack the tarball to a different folder than any previous tarballs.

```bash
$ mkdir twistlock_<VERSION>
$ tar -xzf prisma_cloud_compute_edition_<VERSION>.tar.gz -C twistlock_<VERSION>/
```

The setup package contains updated versions of `twistlock.sh` and `twistlock.cfg`.

**STEP 3** | Check the version of Prisma Cloud that will be installed:

```bash
$ grep DOCKER_TWISTLOCK_TAG twistlock.cfg
```

**STEP 4** | Upgrade Prisma Cloud while retaining your current data and configs by using the `-j` option. The `-j` option merges your current configuration with any new configuration settings in the new version of the software.

You must use the same install target in your upgrade as your original installation. There are two install targets: `onebox` and `console`, where `onebox` installs both Console and Defender onto a host and `console` just installs Console.

To upgrade your `onebox` install, run:

```bash
$ sudo ./twistlock.sh -syj onebox
```

To upgrade your `console` install, run:

```bash
$ sudo ./twistlock.sh -syj console
```

**STEP 5** | Go to Manage > Defenders > Manage and validate that Console has upgraded your Defenders.
Kubernetes

Upgrade Prisma Cloud running in your Kubernetes cluster.

First upgrade Console. Console will then automatically upgrade all deployed Defenders for you.

If you've disabled Defender auto-upgrade or if Console fails to upgrade one or more Defenders, manually upgrade your Defenders.

You must manually upgrade App-embedded Defenders and PCF Defenders.

Upgrading Console

Since Prisma Cloud objects can be specified with configuration files, we recommend declarative object management for both install and upgrade.

You should have kept good notes when initially installing Prisma Cloud. The configuration options set in twistlock.cfg and the parameters passed to twistcli in the initial install are used to generate working configurations for the upgrade.

Prerequisites: You know how you initially installed Prisma Cloud, including all options set in twistcli.cfg and parameters passed to twistcli.

STEP 1 | Download the latest Prisma Cloud release to the host where you manage your cluster with kubectl.

STEP 2 | If you customized twistlock.cfg, port those changes forward to twistlock.cfg in the latest release. Otherwise, proceed to the next step.

STEP 3 | Generate new YAML configuration file for the latest version of Prisma Cloud. Pass the same options to twistcli as you did in the original install. The following example command generates a YAML configuration file for the default basic install.

```
$ <PLATFORM>/twistcli console export kubernetes --service-type LoadBalancer
```

STEP 4 | If you're upgrading from 19.03, then you must first delete the old ReplicationController. Starting with 19.07, Prisma Cloud Console is managed by a Deployment controller.

This is a one time step only. After upgrading to 19.07, you no longer need to manually delete the ReplicationController when upgrading to newer versions of Prisma Cloud.

```
$ kubectl delete rc twistlock-console -n twistlock
```

STEP 5 | Update the Prisma Cloud objects.

```
$ kubectl apply -f twistlock_console.yaml
```

STEP 6 | Go to Manage > Defenders > Manage and validate that Console has upgraded your Defenders.
OpenShift

Upgrade Prisma Cloud running in your OpenShift cluster.

First upgrade Console. Console will then automatically upgrade all deployed Defenders for you.

If you’ve disabled Defender auto-upgrade or if Console fails to upgrade one or more Defenders, manually upgrade your Defenders.

You must manually upgrade App-embedded Defenders and PCF Defenders.

Upgrading Console

STEP 1 | Download the latest Twistlock release to the host where you manage your cluster with `oc`.

STEP 2 | If you customized `twistlock.cfg`, port those changes forward to `twistlock.cfg` in the latest release. Otherwise, proceed to the next step.

STEP 3 | (Optional) If you’re storing Twistlock images in the cluster’s internal registry, pull the latest images from Twistlock’s cloud registry and push them there. >>>>>>> master:upgrade/upgrade_openshift.adoc Otherwise, proceed to the next step.

1. Pull the latest Prisma Cloud images using URL auth.

```
$ sudo docker pull registry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/console:console_<VERSION>
```

2. Retag the images so that they can be pushed to your

```
$ sudo docker tag \nregistry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/console:console_<VERSION> \ndocker-registry.default.svc:5000/twistlock/private:console_<VERSION>
```

3. Push the Prisma Cloud images to your cluster’s internal registry.

```
$ sudo docker push docker-registry.default.svc:5000/twistlock/private:console_<VERSION>
```

STEP 4 | Generate new YAML configuration file for the latest version of Twistlock. Pass the same options to `twistcli` as you did in the original install. The following example command generates a YAML configuration file for the default basic install.

```
$ <PLATFORM>/twistcli console export openshift --persistent-volume-labels "app-volume=twistlock-console" --service-type "ClusterIP"
```
If you want to pull the image from the internal registry:

```bash
$ <PLATFORM>/twistcli console export openshift \
   --persistent-volume-labels "app-volume=twistlock-console" \
   --image-name "docker-registry.default.svc:5000/twistlock/
   private:console_<VERSION>" \
   --service-type "ClusterIP"
```

For other command variations, see the OpenShift install guide.

**STEP 5 |** Update the Twistlock objects.

```
$ oc apply -f twistlock_console.yaml
```

**STEP 6 |** Go to Manage > Defenders > Manage and validate that Console has upgraded your Defenders.
Helm charts

If you installed Prisma Cloud into your Kubernetes or OpenShift cluster with Helm charts, you can upgrade with the `helm upgrade` command.

First upgrade Console. Console will then automatically upgrade all deployed Defenders for you.

If you've disabled Defender auto-upgrade or if Console fails to upgrade one or more Defenders, manually upgrade your Defenders.

You must manually upgrade App-embedded Defenders and PCF Defenders.

Upgrading Console

Generate an updated Helm chart for Console, and then upgrade to it.

**STEP 1 |** Download the latest Prisma Cloud release.

**STEP 2 |** Create an updated Console Helm chart.

```bash
$ <PLATFORM>/twistcli console export kubernetes \
   --service-type LoadBalancer \ 
   --helm
```

**STEP 3 |** Install the updated chart.

```bash
$ helm upgrade twistlock-console \
   --namespace twistlock \ 
   --recreate-pods \ 
   ./twistlock-console-helm.tar.gz
```

**STEP 4 |** Go to Manage > Defenders > Manage and validate that Console has upgraded your Defenders.
Docker Swarm

Upgrade Prisma Cloud running in your Swarm cluster.
First upgrade Console. Console will then automatically upgrade all deployed Defenders for you.
If you’ve disabled Defender auto-upgrade or if Console fails to upgrade one or more Defenders, manually upgrade your Defenders.
You must manually upgrade App-embedded Defenders and PCF Defenders.

Upgrading Console

To upgrade Console in a Docker Swarm cluster, rerun the install procedure with the latest Prisma Cloud release. Use the same configuration options in `twistlock.cfg` and `twistcli` as you used in the initial install.

**Prerequisites:** You know how you initially installed Prisma Cloud, including all options set in `twistcli.cfg` and parameters passed to `twistcli`.

**STEP 1 |** Connect to your master node.

```
$ ssh <SWARM-MASTER>
```

**STEP 2 |** Download the latest Prisma Cloud release.

```
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```

**STEP 3 |** Unpack the Prisma Cloud release tarball.

```
$ mkdir twistlock_<VERSION>
$ tar xzf twistlock_<VERSION>.tar.gz -C twistlock_<VERSION>/
$ cd twistlock_<VERSION>
```

**STEP 4 |** If you customized `twistlock.cfg` during the original install, port those changes forward to `twistlock.cfg` in the latest release. Otherwise, proceed to the next step.

**STEP 5 |** Update the Prisma Cloud Console by running the same `twistcli` command used during the original install.

```
$ ./linux/twistcli console install swarm --volume-driver <DRIVER>
```

**STEP 6 |** Go to Manage > Defenders > Manage and validate that Console has upgraded your Defenders.
Amazon ECS

Upgrade Prisma Cloud running on Amazon ECS.

First upgrade Console. Console will then automatically upgrade all deployed Defenders for you.

If you’ve disabled Defender auto-upgrade or if Console fails to upgrade one or more Defenders, manually upgrade your Defenders.

You must manually upgrade App-embedded Defenders and PCF Defenders.

Upgrading Console (Amazon ECS)

To upgrade Console, update the service with a new task definition that points to the latest image.

This procedure assumes you’re using images from Prisma Cloud’s registry. If you’re using your own private registry, push the latest Console image there first.

Copy the Prisma Cloud config file into place

**STEP 1** | Go to the Releases page and download the latest release to your local machine.

```bash
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```

**STEP 2** | Unpack the Prisma Cloud release tarball.

```bash
$ mkdir twistlock
$ tar xvzf twistlock_<VERSION>.tar.gz -C twistlock/
```

**STEP 3** | Upload the `twistlock.cfg` files to the host that runs Console.

```bash
$ scp twistlock.cfg <ECS_INFRA_NODE>:/twistlock_console/var/lib/twistlock-config
```

Create a new revision of the task definition

Create a new revision of the task definition.

**STEP 1** | Log into the Amazon ECS console.

**STEP 2** | In the left menu, click Task Definitions.

**STEP 3** | Check the box for the Prisma Cloud Console task definition, and click Create new revision.

**STEP 4** | Scroll to the bottom of the page and click Configure via JSON.

1. Update the `image` field to point to the latest Console image.

   For example, if you were upgrading from Prisma Cloud version 2.4.88 to 2.4.95, simply change the version string in the image tag.

   ```json
   "image": "registry-auth.twistlock.com/tw_<accesstoken>/twistlock/console:console_2_4_95"
   ```

2. Click Save.
STEP 5 | Click **Create**.

**Update the Console service**

Update the Console service.

STEP 1 | In the left menu of the Amazon ECS console, click **Clusters**.

STEP 2 | Click on your cluster.

STEP 3 | Select the **Services** tab.

STEP 4 | Check the box next the Console service, and click **Update**.

STEP 5 | In **Task Definition**, select the version of the task definition that points to the latest Console image.

STEP 6 | Validate that **Cluster**, **Service name**, and **Number of tasks** are correct. These values are set based on the values for the currently running task, so the defaults should be correct. The number of tasks must be 1.

STEP 7 | Set **Minimum healthy percent** to 0.

This lets ECS safely stop the single Console container so that it can start an updated Console container.

STEP 8 | Set **Maximum percent** to 100.

STEP 9 | Click **Next**.

STEP 10 | In the **Configure network** page, accept the defaults, and click **Next**.

STEP 11 | In the **Set Auto Scaling** page, accept the defaults, and click **Next**.

STEP 12 | Click **Update Service**.

It takes a few moments for the old Console service to be stopped, and for the new service to be started. Open Console, and validate that the UI shows new version number in the bottom left corner.

STEP 13 | Go to **Manage > Defenders > Manage** and validate that Console has upgraded your Defenders.

If Console fails to upgrade any Defender, upgrade it **manually**.
Manually upgrade single Container Defenders

The Console user interface lets you upgrade all Defenders in a single shot. This method minimizes the effort required to upgrade all your deployed Defenders.

Alternatively, you can select which Defenders to upgrade. Use this method when you have different maintenance windows for different deployments. For example, you might have an open window on Tuesday to upgrade thirty Defenders in your development environment, but no available window until Saturday to upgrade the remaining twenty Defenders in your production environment. In order to give you sufficient time to upgrade your environment, older versions of Defender can coexist with the latest version of Defender and the latest version of Console.

**Prerequisites:** You have already upgraded Console.

**STEP 1 |** Open Console.

**STEP 2 |** On the left menu bar, go to **Manage > Defender > Manage** and click **Defenders** to see a list of all your deployed stand-alone Container Defenders.

**STEP 3 |** Upgrade your stand-alone Defenders. You can either:

- Upgrade all Defenders at the same time by clicking **Upgrade all**.
- Upgrade a subset of your Defenders by clicking the individual **Actions > Upgrade** button in the row that corresponds to the Defender you want to upgrade.

> **The Restart and Decommission buttons are not available for DaemonSet Defenders. They are only available for stand-alone Defenders.**
Manually upgrade Defender DaemonSets

Manually upgrade Defender DaemonSets in your environment.

Manually upgrade Defender DaemonSets with twistcli (Kubernetes)

Delete the Defender DaemonSet, then rerun the original install procedure.

**Prerequisites:** You know all the parameters passed to `twistcli` when you initially deployed the Defender DaemonSet. You'll need them to recreate a working configuration file for your environment.

**STEP 1 | Delete the Defender DaemonSet.**

$ kubectl -n twistlock delete ds twistlock-defender-ds
$ kubectl -n twistlock delete sa twistlock-service
$ kubectl -n twistlock delete secret twistlock-secrets

**STEP 2 | Determine the Console service’s external IP address.**

$ kubectl get service -o wide -n twistlock

**STEP 3 | Generate a `defender.yaml` file.** Pass the same options to `twistcli` as you did in the original install. The following example command generates a YAML configuration file for the default install.

The following command connects to Console’s API (specified in `--address`) as user `<ADMIN>` (specified in `--user`), and retrieves a Defender DaemonSet YAML config file according to the configuration options passed to `twistcli`. In this command, there is just a single mandatory configuration option. The `--cluster_address` option specifies the address Defender uses to connect to Console, and the value is encoded in the DaemonSet YAML file.

$ `<PLATFORM>/twistcli defender export kubernetes` \
   `--address https://yourconsole.example.com:8083` \
   `--user <ADMIN_USER>` \
   `--cluster-address twistlock-console`

- `<PLATFORM>` can be linux or osx.
- `<ADMIN_USER>` is the name of an admin user.

**STEP 4 | Deploy the Defender DaemonSet.**

$ kubectl create -f defender.yaml

**STEP 5 | In Prisma Cloud, go to Manage > Defenders > Manage > DaemonSets to see a list of deployed Defenders.**

Manually upgrade Defender DaemonSets with twistcli (OpenShift)

Delete the Defender DaemonSet, then rerun the original install procedure.
Prerequisites: You know all the parameters passed to `twistcli` when you initially deployed the Defender DaemonSet. You'll need them to recreate a working configuration file for your environment.

**STEP 1 |** Delete the Defender DaemonSet.

```
$ oc -n twistlock delete ds twistlock-defender-ds
$ oc -n twistlock delete sa twistlock-service
$ oc -n twistlock delete secret twistlock-secrets
```

**STEP 2 |** Determine the Console service's external IP address.

```
$ oc get service -o wide -n twistlock
```

**STEP 3 |** Generate a `defender.yaml` file. Pass the same options to `twistcli` as you did in the original install. The following example command generates a YAML configuration file for the default install.

The following command connects to Console's API (specified in `--address`) as user `<ADMIN>` (specified in `--user`), and retrieves a Defender DaemonSet YAML config file according to the configuration options passed to `twistcli`. In this command, there is just a single mandatory configuration option. The `--cluster-address` option specifies the address Defender uses to connect to Console, and the value is encoded in the DaemonSet YAML file.

```
$ <PLATFORM>/twistcli defender export openshift \
--address https://yourconsole.example.com:8083 \
--user <ADMIN_USER> \
--cluster-address twistlock-console \
--selinux-enabled
```

- `<PLATFORM>` can be linux or osx.
- `<ADMIN_USER>` is the name of an admin user.

**STEP 4 |** Deploy the Defender DaemonSet.

```
$ oc create -f defender.yaml
```

**STEP 5 |** In Prisma Cloud, go to Manage > Defenders > Manage > DaemonSets to see a list of deployed Defenders.

Manually upgrade Defender DaemonSets from Console

Upgrade the Defender DaemonSets directly from the Console UI.

If you can't access your cluster with kubectl or oc, then you can upgrade Defender DaemonSets directly from the Console UI.

Prerequisites: You've created a kubeconfig credential for your cluster so that Prisma Cloud can access it to upgrade the Defender DaemonSet.

**STEP 1 |** Log into Prisma Cloud Console.

**STEP 2 |** Go to Manage > Defenders > Manage.

**STEP 3 |** Click DaemonSets.
STEP 4 | For each cluster in the table, click Actions > Upgrade.

The table shows a count of deployed Defenders and their new version number.
Manually upgrade Defender DaemonSets (Helm)

Generate an updated Helm chart for the Defender DaemonSet, and then upgrade to it.

**STEP 1 | Create an updated Defender DaemonSet Helm chart.**

```
$ <PLATFORM>/twistcli defender export kubernetes \ 
   --address https://yourconsole.example.com:8083 \ 
   --user <ADMIN_USER> \ 
   --cluster-address twistlock-console \ 
   --helm
```

**STEP 2 | Install the updated chart.**

```
$ helm upgrade twistlock-defender-ds \ 
   --namespace twistlock \ 
   --recreate-pods \ 
   ./twistlock-console-helm.tar.gz
```
Manually upgrade Docker Swarm Defenders

Rerun the original Defender install procedure to upgrade the Defender global service. You cannot upgrade global service Defenders directly from the Console UI.
Technology overviews

This section describes how key Prisma Cloud components work.

- Prisma Cloud Advanced Threat Protection
- App-specific network intelligence
- Container Runtimes
- Radar
- Serverless Radar
- Prisma Cloud Rules Guide - Docker
- Defender architecture
- Telemetry
Prisma Cloud Advanced Threat Protection

Prisma Cloud Advanced Threat Protection (ATP) is a collection of malware signatures and IP reputation lists aggregated from commercial threat feeds, open source threat feeds, and Prisma Cloud Labs. It is delivered to your installation via the Prisma Cloud Intelligence Stream.

The data in ATP is used by Prisma Cloud’s runtime defense system to detect suspicious activities, such as a container communicating with a botnet herder or Tor entry node. You can augment ATP by importing custom malware data and importing IP reputation lists. ATP is the combination of both the Prisma Cloud-provided data set and your own custom data set.

The following hypothetical scenario illustrates how ATP protects your containers:

1. An attacker exploits a vulnerability in an app running in a container.
2. The attacker attempts to download malware into the container from a distribution point.
3. ATP detects both the connection to the malware server and the write of the malicious file to the container file system.
4. An audit is generated. Audits can be viewed in Console under Monitor > Events > Container Audits.

ATP also protects hosts. Host audits can viewed under Monitor > Events > Host Audits.

Enabling ATP

ATP is enabled in the default rules that ship with the product, with the effect set to alert. You can impose more stringent control by setting effect to prevent or block. Runtime defense for file systems lets you actively stop (block) any container that tries to download malware.

To disable ATP, create or modify a runtime rule, select the General tab, and set Enable Prisma Cloud Advanced Threat Protection to Off. When ATP is disabled, container interaction with malicious files or IP endpoints does not trigger a runtime event.
App-specific network intelligence

Prisma Cloud can learn about the settings for your apps from their configuration files, and use this knowledge to detect runtime anomalies. No special configuration is required to enable this feature.

In addition to identifying ports that are exposed via the EXPOSE directive in a Dockerfile, or the -p argument passed to docker run, Prisma Cloud can identify port settings from an app’s configuration file. This enables Prisma Cloud to detect, for example, if the app has been commandeered to listen on an unexpected port, or if a malicious process has managed to listen on the app’s port to steal data.

Consider the following scenario:

1. You create an Apache image. The default port for httpd, specified in /etc/apache2/apache2.conf, is 80. In your Dockerfile, you use EXPOSE to bind port 80 in the container to port 80 on the host.
2. A user runs your Apache image with the -P option, mapping port 80 in the container to a random ephemeral port on the host.
3. The running container is compromised. An attacker kills the Apache process, listens on that port himself, and harvests data from other containers on the same subnet.
4. Prisma Cloud detects the runtime anomaly, and either alerts you or blocks the container.

Prisma Cloud protects your containers by combining static analysis of the image with runtime analysis of the container. The Prisma Cloud Intelligence Stream delivers app-specific knowledge so that Defender can inspect an image and:

- Identify processes that the container will execute.
- Correlate the processes with their configuration files.
- Parse the configuration files to extract information such as port assignments.

Runtime analysis completes the picture. Some information can only be determined at runtime. For example, MongoDB might be deployed to a container without a configuration file. At runtime, MongoDB is launched with the --port parameter, dynamically specifying the port it will listen on. Static analysis tells us that MongoDB is part of the container image, but in this case, only dynamic analysis tells us which port it listens on.

Additional apps will be added periodically, and your installation will be automatically updated via the Prisma Cloud Intelligence Stream.

Supported Apps

Prisma Cloud Intelligence Stream currently delivers app-specific knowledge for:

- Apache
- Elasticsearch
- HAPProxy
- Kibana
- MariaDB
- MongoDB
- MySQL
- Nginx
- PostgreSQL
- RabbitMQ
- Redis
- Tomcat
- WordPress
• BusyBox

If you would like to see coverage for a specific app, open a support ticket and make a request.
Container Runtimes

Docker Engine is a general purpose container runtime. Docker can run containers from images, but it can also build images from Dockerfiles. Docker supports multiple different environments and orchestrators, including Kubernetes.

Container Runtime Interface (CRI) is a plugin interface that lets Kubernetes use a wide variety of container runtimes, including Docker Engine. The interface implements only the features needed to run containers from images. Its goal is to be as simple as possible to complete its given task. Since its range of capabilities is tightly scoped, it can be more easily secured.
Radar

Radar is the primary interface for monitoring and understanding your environment. It is the default view when you first log into Console. It is designed to let you visualize and navigate through all of Prisma Cloud’s data. For example, you can visualize connectivity between microservices, then instantly drill into the per-layer vulnerability analysis tool, assess compliance, and investigate incidents, all without leaving the Radar canvas.
Radar makes it easy to conceptualize the architecture and connectivity of large environments, identify risks, and zoom in on incidents that require response. Radar provides a visual depiction of inter- and intra-network connections between containers, apps, and cluster services across your environment. It shows the ports associated with each connection, the direction of traffic flow, and internet accessibility. When Cloud Native Network Firewall is enabled, Prisma Cloud automatically generates the mesh shown in Radar based on what it has learned about your environment.

Radar’s principal pivot is the container view and host view. In the container view, each image with running containers is depicted as a node in the graph. In the host view, each systemd service, or *app*, is depicted as a node in the graph. Clicking on a node pops up an overlay that shows vulnerability, compliance, and runtime issues.

Radar refreshes its view every 24 hours. The Refresh button has a red marker when new data is available to be displayed. In order to get full visibility into your environment, Defender should be installed on every host in your environment.

**Cluster pivot**

Radar segments your environment by cluster. The main view lists all clusters in your environment. Clicking a card open the image pivot, which shows you all the namespaces and containers in the cluster.
Defenders report which resources belong to which cluster. For managed clusters, Prisma Cloud automatically retrieves the name from the cloud provider. As a fallback, Prisma Cloud can retrieve the name from your kubeconfig file. Finally, you can manually specify the cluster name.

The cluster pivot is currently supported for Kubernetes and OpenShift clusters only. All other running containers in your environment are collected in the **Non-Cluster Containers** view.

**Image pivot**

Radar lays out nodes on the canvas to promote easy analysis of your containerized apps. Interconnected nodes are laid out so network traffic flows from left to right. Traffic sources are weighted to the left, while destinations are weighted to the right. Single, unconnected nodes are arranged in rows at the bottom of the canvas.

Nodes are color-coded based on the highest severity vulnerability or compliance issue they contain, and reflect the currently defined vulnerability and compliance policies. Color coding lets you quickly spot trouble areas in your deployment.
- Dark Red — High risk. One or more critical severity vulnerabilities detected.
- Red — High severity vulnerabilities detected.
- Orange — Medium vulnerabilities detected.
- Green — Denotes no vulnerabilities detected.

The numeral encased by the circle indicates the number of containers represented by the node. For example, a single Kubernetes DNS node may represent five services. The color of the circle specifies the state of the container’s runtime model. A blue circle means the container’s model is still in learning mode. A black circle means the container’s model is activated. A globe symbol indicates that a container can access the Internet.

Connections between running containers are depicted as arrows in Radar. Click on an arrow to get more information about the direction of the connection and the port.
The initial zoomed out view gives you a bird’s-eye view of your deployments. Deployments are grouped by namespace. A red pool around a namespace indicates an incident occurred in a resource associated with that namespace.
Zooming in provides more detail about each running container. Click on an individual pod to drill down into its vulnerability report, compliance report, and runtime anomalies.
Host pivot

Radar shows the hosts in your environment, how they communicate with each other over the network, and their security posture.

Each node in the host pivot represents a host machine. The mesh shows host-to-host communication. The color of a node represents the most severe issue detected.

- Dark Red — High risk. One or more critical severity issues detected.
- Red — High severity issues detected.
- Orange — Medium issues detected.
- Green — No issues detected.

When you click on an node, an overlay shows a summary of all information Prisma Cloud knows about the host. Use the links to drill down into scan reports, audits, and other data.
Cloud pivot

You can't secure what you don't know about. Prisma Cloud cloud discovery finds all cloud-native services deployed in AWS, Azure, and Google Cloud. Cloud Radar helps you visualize what you've deployed across different cloud providers and accounts using a map interface. The map tells you what services are running in which data centers, which services are protected by Prisma Cloud, and their security posture.

Clicking on a marker on the map shows more details about the services deployed in the account/region. Both registries and serverless functions can be secured directly from the info pop-up by clicking Protect.
Filtering and search lets you narrow your focus to the data of interest. For example, filters can narrow your view to just the serverless functions in your Azure development team accounts.

By default, there's no data in Cloud Radar. To populate Cloud Radar, configure cloud discovery scans.

Service account monitoring

Kubernetes has a rich RBAC model based around the notion of service and cluster roles. This model is fundamental to the secure operation of the entire cluster because these roles control access to resources.
and services within namespaces and across the cluster. While these service accounts can be manually inspected with kubectl, it’s difficult to visualize and understand their scope at scale.

Radar provides a discovery and monitoring tool for service accounts. Every service account associated with a resource in a cluster can easily be inspected. For each account, Prisma Cloud shows detailed metadata describing the resources it has access to and the level of access it has to each of them. This visualization makes it easy for security staff to understand role configuration, assess the level of access provided to each service account, and mitigate risks associated with overly broad permissions.

Clicking on a node opens an overlay, and reveals the service accounts associated with the resource.

Clicking on the service accounts lists the service roles and cluster roles.
Service account monitoring is available for Kubernetes and OpenShift clusters. When you install the Defender DaemonSet, enable the ‘Monitor service accounts’ option.

**Istio monitoring**

When Defender DaemonSets are deployed with Istio monitoring enabled, Prisma Cloud can discover the service mesh and show you the RBAC capabilities for each service (e.g., this pod can read service X using REST/grpc on the following endpoints). Services integrated with Istio display the Istio logo.

Clicking on an Istio node opens an overlay with additional data about the service.
Click on the Istio button, then click on the link to get more details about the service roles.

Istio monitoring is available for Kubernetes and OpenShift clusters. When you install the Defender DaemonSet, enable the 'Monitor Istio' option.
Serverless Radar

Serverless Radar helps you to visualize and inspect the attack surface of the serverless functions in your environment. Although Prisma Cloud supports multiple serverless environments, currently serverless radar supports AWS Lambda only.

Serverless functions use different interconnect patterns than containers. Serverless apps are highly decomposed and interact with services using cloud provider-specific gateways, rather than directly with each other or through service meshes. Security teams can have difficulty conceptualizing these interactions, identifying which functions interface with which high value assets, and pinpointing unacceptable exposure.

Even though cloud providers secure the underlying infrastructure that enables Functions as a Service (including isolating functions from each other), it’s still easy to deploy functions with vulnerabilities, insecure configurations, and overly permissive roles. The underlying platform might be secure, but sensitive data can still be lost when an insecure function with read access to an S3 bucket is compromised.

Prisma Cloud offers a serverless-specific view in Radar. Serverless Radar uses a three panel view to show the invocation methods for each function, the services they use, and the permissions granted to access those services.

Layout

Serverless Radar shows you how functions interface with other services in their environment.

The left-most column shows how functions are invoked. This is known as the trigger or event source. Triggers publish events, and Lambda functions are the custom code that process those events.

The middle column shows all the functions in your environment. Functions are colored maroon, red, orange, yellow, or green to let you quickly assess their security posture. By default, functions are colored by their most severe vulnerabilities, but you can view functions by highest severity compliance issue or runtime events. For vulnerability results, you must configure Prisma Cloud to scan your functions. For runtime issues, you must embed Serverless Defender into your functions.

The right-most column shows the services with which each function interfaces. Drilling into the function data reveals the permissions each function has been granted to access those services.

Lines connect triggers to functions to services, letting security teams to visualize the entire connectivity flow and access rights. Clicking on individual functions highlights their interconnects in the radar, and opens a pop-up that lets you drill into the details.
Exploring the data

Prisma Cloud finds, scans, and displays the $LATEST version and all published versions of your functions. Clicking a node in Serverless Radar lets you inspect a function's configuration and explore all the security-related data that Prisma Cloud has indexed about it.

For example, clicking on the or-test2:$LATEST function opens a popup with summary findings. This particular function has two high risk compliance issues. Clicking on the compliance link takes you to a list of compliance issues for the function.

Compliance issue 437 indicates overly permissive access to one or more services. Expanding the issue reveals the reason why this compliance issue was raised, with a list of non-compliant service access configurations. One of the misconfigured access policy is for S3.
Returning to the first pop-up window, and clicking into the S3 service, you can see that all the actions for the function’s execution role are tightly scoped, except for the last one. It allows all actions on all resources, and could easily be an erroneous configuration overlooked when it was pushed into production.

Icons and colors

Nodes are color coded based on the highest severity vulnerability or compliance issue they contain, and reflect the currently defined vulnerability and compliance policies. Color coding lets you quickly spot trouble areas in your deployment. Use the drop-down list at the top of the view to choose how you want nodes colored.

- **Maroon** -- High risk. One or more critical severity issues detected.

- **Red** -- High severity issues detected.

- **Orange** -- Medium severity issues detected.

- **Yellow** -- Low severity issues detected.
- Green — Denotes no issues detected.

- Gray — Prisma Cloud hasn't been configured to scan this function for vulnerability and compliance issues.

To configure Prisma Cloud to scan the function, click on the node, and then click **Protect** in the pop-up.

- Alias annotation — AWS lets you create **aliases** to manage the process of promoting new function versions into production. They're conceptually similar to symbolic links in the UNIX file system. Prisma Cloud uses a marker to indicate that an alias points to a specific version of a function.

  Clicking on the node reveals the aliases that point to the function.
Notes

There can be a discrepancy between what the AWS Lambda designer shows your function can do and its effective permissions when IAM permission boundaries are considered.

For example, if a role is set with permission boundary for DynamoDB, then even though the function's execution role has permission to access DynamoDB, it still might be blocked by the permission boundary. The function designer in AWS's console shows that the function has permission to DynamoDB, but it might not be accurate.
Setting up Serverless Radar

Serverless Radar uses the AWS APIs to discover and inspect the functions in your environment. Create an IAM user or role for Prisma Cloud, provide the credentials to Console, and then enable Serverless Radar. With this basic setup, Prisma Cloud will show the triggers, services, and permissions for each function.

Prerequisites:

- Prisma Cloud needs an AWS service account to scan your serverless functions. In AWS, you've created an IAM user or role with the following permission policy:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "apigateway:GET",
                "cloudfront:ListDistributions",
                "cloudwatch:GetMetricData",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeRules",
                "elasticloadbalancing:DescribeTargetGroups",
                "events:ListRules",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListAttachedRolePolicies",
                "iam:ListRolePolicies",
                "lambda:GetFunction",
                "lambda:GetPolicy",
                "lambda:ListFunctions",
                "lambda:ListAliases",
                "lambda:ListEventSourceMappings",
                "logs:DescribeSubscriptionFilters",
                "s3:GetBucketNotification"
            ]
        }
    ]
}
```
STEP 1 | Open Console.

STEP 2 | Go to Defend > Compliance > Cloud Platforms.

STEP 3 | Click Add account, and configure an AWS account.

STEP 4 | Select the checkbox for the credential.

STEP 5 | Click Add.

STEP 6 | For the account just added, select the Serverless Radar checkbox.

STEP 7 | Click the yellow save button.

After Prisma Cloud finishes scanning your environment, you should see your functions in Serverless Radar.

What’s next?

To see vulnerability and compliance information in Serverless Radar, configure Prisma Cloud to scan the contents of each function.
Prisma Cloud Rules Guide - Docker

This article provides a list of all rules and their intended behavior in Prisma Cloud Console UI. The purpose of this article is to help users better understand the intention of each rule in the Console and its corresponding effect on the host environment.

Running Docker commands through Defender

To access Docker daemon through Defender, you must explicitly specify Defender's host and port. For example:

```
$ docker -H <DEFENDER_HOST_ADDRESS>:9998 run alpine
```

It is possible to make the management traffic between the Docker client and the Docker daemon flow through Defender by default via two environment variables. Those can be configured on a remote machine that accesses Docker daemon on some host (such as dev laptop), or the host itself for users who do not have root privileges (which should be the majority of users).

```
$ export DOCKER_HOST=tcp://<defender host address>:9998
$ export DOCKER_TLS_VERIFY=1
```

Once set, default calls to Docker flow through Defender (e.g., docker ps, docker run alpine). Throughout this guide however, in this guide, we have followed the default command without setting environment variables.

About this reference environment

This guide is designed as a reference document for all access rule policies enlisted in Prisma Cloud Console and their intended affect on host environment. These commands are run from a Docker client to a Prisma Cloud Defender using the access control feature. Access control rules can be configured at Defend > Access > Docker.

We have organized this document using the same structure as the Prisma Cloud product UI, which follows the structure in the Docker Remote API documentation. Note that there may be minor differences in the structure as the Docker Remote API evolves; this document is currently aligned with the documentation for API v 1.24 and will be updated periodically with new releases.

*For understanding purposes all rules are set to deny and their corresponding influence on host environment is recorded.*

Defend access rules

Navigate to Defend > Access > Docker.

Containers

For more information about the Docker API for containers, see https://docs.docker.com/engine/api/v1.30/#tag/Container.

*container_list - List containers*

Affects docker ps command on host which is used to list all running containers.
Command:
```
docker -H 10.0.0.1:9998 --tlsverify ps
```
Response:
```
[Prisma Cloud] The command container_list denied for user admin by rule Deny
```

**container_create - Create a container**
Affects docker create command used to create a new container.
Command:
```
docker -H 10.0.0.1:9998 --tlsverify create morello/docker-whale
```
Response:
```
[Prisma Cloud] The command container_create denied for user admin by rule Deny
```

**container_inspect - Inspect a container**
Affects docker inspect command used for returning information about the container.
Command:
```
docker -H 10.0.0.1 --tlsverify inspect ubuntu_bash2
```
Response:
```
[Prisma Cloud] The command container_inspect denied for user admin by rule inspect
```

**container_top - List processes running inside a container**
Affects docker top command used to display the running processes of a container.
Command:
```
docker -H 10.0.0.1:9998 --tlsverify top ubuntu_bash
```
Response:
```
[Prisma Cloud] The command container_top denied for user admin by rule Deny
```

**container_logs - Get container logs**
Affects docker logs command used for returning logs from the container present at the time of execution.
Command:
```
docker -H 10.0.0.1 --tlsverify logs ubuntu_bash2
```
Response:

container_changes - Inspect changes on a container's filesystem
Affect docker commit command and restricts any changes to the container.
Command:

docker -H 10.0.0.1 --tlsverify commit --change "ENV DEBUG true" cc2d57988b aqsa/testimage:version3

Response:

container_export - Export a container
Affects docker export command that exports a container's filesystem as a tar archive
Command:

docker -H 10.0.0.1:9998 --tlsverify export twistlock_console -o saved.tar

Response:

container_stats - Get container stats based on resource usage
Affects docker stats command on host which returns live data stream for running containers.
Command:

docker -H 10.0.0.1 --tlsverify stats silly_stallman

Response:

container_resize - Resize a container
Affects docker logs command used for returning logs from the container present at the time of execution. This related to the size of the window of how output is returned from the container. It is called TTY.
Command:

Response:

container_start - Start a container
Affects docker start command used to start one or more stopped containers
Command:

docker -H 10.0.0.1:9998 --tlsverify start ubuntu_bash

Response:

[Prisma Cloud] The command container_start denied for user admin by rule Deny all

container_stop - Stop a container
Affects docker stop command used to stop running container

Command:

docker -H 10.0.0.1:9998 --tlsverify stop ubuntu_bash

Response:

[Prisma Cloud] The command container_stop denied for user admin by rule Deny

container_restart - Restart a container
Affects docker restart command on host, used to restart a container.

Command:

docker -H 10.0.0.1:9998 --tlsverify restart ubuntu_bash

Response:

[Prisma Cloud] The command container_restart denied for user admin by rule Deny

container_kill - Kill a container
Affects docker kill command used to kill a running container.

Command:

docker -H 10.0.0.1:9998 --tlsverify kill ubuntu_bash

Response:

[Prisma Cloud] The command container_kill denied for user admin by rule Deny

container_rename - Rename a container
Affects docker rename command on host that is used to rename a container.

Command:

docker -H 10.0.0.1:9998 --tlsverify rename ubuntu_bash unbuntu
Response:

[Prisma Cloud] The command container_rename denied for user admin by rule Deny
Error: failed to rename container named ubuntu_bash

**container_pause** - Pause a container

Affects docker pause command on host which is used to pause all processes within one or more containers.

Command:

docker -H 10.0.0.1 --tlsverify pause focused_cori

Response:

[Prisma Cloud] The command container_pause denied for user admin by rule Deny

**container_unpause** - Unpause a container

Affects docker unpause command on host which is used to un-suspend all processes in a container.

Command:

docker -H 10.0.0.1 --tlsverify unpause silly_stallman

Response:

[Prisma Cloud] The command container_unpause denied for user admin by rule unpause

**container_attach** - Attach to a container

Affects docker attach command on host where defender is deployed.

Command:

docker -H 10.0.0.1 --tlsverify attach mycontainer

Response:

[Prisma Cloud] The command container_attach denied for user admin by rule attach persistent connection closed

**container_attachws** - Attach to a container (websocket)

Affects docker attach command on host where defender is deployed. Attach to the container id via websocket. Implements websocket protocol handshake according to RFC 6455

Command:

docker -H 10.0.0.1 --tlsverify attach mycontainer
Response:

**[Prisma Cloud]** The command container_attach denied for user admin by rule
attach persistent connection closed

**container_wait - Wait a container**

Affects docker wait command used to block until a container stops, then print its exit code.

Command:

```bash
docker -H 10.0.0.1:9998 --tlsverify wait ubuntu_bash
```

Response:

**[Prisma Cloud]** The command container_wait denied for user admin by rule Deny

**container_delete - Remove a container**

Affects docker rm command used for deleting a container.

Command:

```bash
docker -H 10.0.0.1:9998 --tlsverify rm <container>
```

Response:

**[Prisma Cloud]** The command container_delete denied for user admin by rule delete

**container_archive - Gets an archive of filesystem resource in a container**

Get a tar archive of a resource in the filesystem of container id. Affects docker cp command.

Command:

```bash
docker -H 10.0.0.1:9998 --tlsverify cp <container> > latest.tar
```

Response:

**[Prisma Cloud]** The command container_copy denied for user admin by rule delete

**container_extract - Extract an archive of files or folders to a directory in a container**

Affects docker export command. Uploads a tar archive to be extracted to a path in the filesystem of container id.

Command:

```bash
docker -H 10.0.0.1:9998 --tlsverify cp <container> > latest.tar
```
Response:

[Prisma Cloud] The command container_exec_start denied for user admin by rule exec

Images

For more information about the Docker API for images, see https://docs.docker.com/engine/api/v1.30/
tag/Image.

image_list - List images
Affects docker images command used to list all images
Command:

docker -H 10.0.0.1:9998 --tlsverify images

Response:

[Prisma Cloud] The command image_list denied for user admin by rule Deny

image_build - Build image from a Dockerfile
Affects docker build command that is used to build an image from a Dockerfile.
Command:

docker -H 172.18.0.1:9998 --tlsverify build -t aqsa/testimage:v2 .

Response:

[Prisma Cloud] The command image_build denied for user admin by rule Default - deny all

image_create - Create an image
Affects docker pull command which is used to pull an image
Command:

docker -H 10.0.0.1:9998 --tlsverify pull ubuntu:latest

Response:

[Prisma Cloud] The command image_create denied for user admin by rule Deny

image_inspect - Inspect an image
Description
Affects docker inspect command used for returning information about the container.
Command:

docker -H 10.0.0.1:9998 --tlsverify inspect 28e7d49f8e6d
Response:

**image_inspect** - Get the history of an image
Affects docker history `<image>` command.
Command:
```
docker -H 172.18.0.1:9998 --tlsverify history twistlock
```
Response:

**image_history** - Get the history of an image
Affects docker history `<image>` command.
Command:
```
docker -H 172.18.0.1:9998 --tlsverify history twistlock
```
Response:

**image_push** - Push an image on the registry
Affects command docker push for pushing an image to repository
Command:
```
docker -H 10.0.0.1:9998 --tlsverify push ubuntu:latest
```
Response:

**image_tag** - Tag an image into a repository
Affects docker tag command used to tag an image in the repository
Command:
```
docker -H 10.0.0.1:9998 --tlsverify tag ubuntu:latest aqsa:tag
```
Response:

**image_delete** - Remove an image
Affects docker rmi command used to delete an image
Command:
```
docker -H 10.0.0.1:9998 --tlsverify rmi aqsa/testimage:version3
```
Response:
images_search - Search images
Affects docker search command which gives a list of available images matching the search item.
Command:
```
docker -H 10.0.0.1:9998 --tlsverify search twistlock
```
Response:
```
[Prisma Cloud] The command images_search denied for user admin by rule deny
```

MISC
Misc other docker commands.

docker_check_auth - Check auth configuration
Validates credentials for a registry and get identity token, if available, for accessing the registry without password. Affects docker login on the host.
Command:
```
docker -H 172.18.0.1:9998 --tlsverify login
```
Response:
```
[Prisma Cloud] The command docker_info denied for user admin by rule Default - deny all
```

docker_info - Display system-wide information
Affects docker info command used to display system-wide information
Command:
```
docker -H 10.0.0.1:9998 --tlsverify info
```
Response:
```
[Prisma Cloud] The command docker_info denied for user admin by rule Deny
```

docker_version - Show the docker version information
Affects docker version command on host which is used to find docker version.
Command:
```
docker -H 10.0.0.1 --tlsverify version
```
Response:
```
[Prisma Cloud] The command docker_version denied for user admin by rule version
```
**docker_ping - Ping the docker server**
The goal of this api is to ping the Docker server and make sure it is up and running.

**Command:**

It is intended to be called by an external monitoring system. It does not have a direct docker CLI command.

**container_commit - Create a new image from a container’s changes**
Affects docker commit command used for committing container’s file changes etc into a new image.

**Command:**

```bash
docker -H 10.0.0.1 --tlsverify commit --change "ENV DEBUG true" cc2d57988b aqsa/testimage:version3
```

**Response:**

```bash
[Prisma Cloud] The command container_commit denied for user admin by rule commit
```

**docker_events - Monitor docker’s events**
Affects docker events command on host which is used to return real time events from the server.

**Command:**

```bash
docker -H 10.0.0.1 --tlsverify events
```

**Response:**

```bash
[Prisma Cloud] The command docker_events denied for user admin by rule events
```

**images_archive - Get a tarball containing all images**
Affects docker save command to save images to a tar archive

**Command:**

```bash
docker -H 172.17.0.1:9998 --tlsverify save $(docker images -q) -o home/aqsa/mydockersimages.tar
```

**Response:**

```bash
[Prisma Cloud] The command images_archive denied for user admin by rule Default - deny all
```

**images_load - Load a tarball with a set of images and tags into docker**
Affects docker load command to load an image from a tar archive or STDIN

**Command:**

```bash
docker -H 172.17.0.1:9998 --tlsverify load -i /home/aqsa/twistlock_1_6_81.tar.gz
```
Response: [Prisma Cloud] The command `images_load` denied for user admin by rule `Default - deny all`

**container_exec_create - Exec Create**

Affects `docker_exec` command to create any new container.

**Command:**

docker -H 10.0.0.1 --tlsverify exec -d ubuntu_bash2 touch /tmp/execWorks

**Response:**

[Prisma Cloud] The command `container_exec_create` denied for user admin by rule `exec`

**container_exec_start - Exec Start**

Affects `docker exec` command used for running a command in a running container.

**Command:**

docker -H 10.0.0.1 --tlsverify exec -d ubuntu_bash2 touch /tmp/execWorks

**Response:**

[Prisma Cloud] The command `container_exec_start` denied for user admin by rule `exec`

**container_exec_inspect - Exec Inspect**

Affects `docker exec` command used for running a command in a running container.

**Command:**

docker -H 10.0.0.1 --tlsverify exec -d ubuntu_bash2 touch /tmp/execWorks

**Response:**

[Prisma Cloud] The command `container_exec_start` denied for user admin by rule `exec`

**container_archive_head**

**Command:**

docker -H 10.0.0.1 --tlsverify unpause silly_stallman

**Response:**

[Prisma Cloud] The command `container_unpause` denied for user admin by rule `unpause`

**container_copyfiles**

Affects `docker cp` command used to copy files from and to containers and local file system on host.
Command:
```bash
docker -H 10.0.0.1 --tlsverify cp file mycontainer:~
```

Response:
```plaintext
[Prisma Cloud] The command container_copyfiles denied for user admin by rule unpause
```

Volumes

For more information about the Docker API for volumes, see https://docs.docker.com/engine/api/v1.30/#tag/Volume.

**volume_list - List volumes**

Affects docker volume ls command to list all volumes

Command:
```bash
docker -H 10.0.0.1:9998 --tlsverify volume ls
```

Response:
```plaintext
[Prisma Cloud] The command volume_list denied for user admin by rule Deny
```

**volume_create - Create a volume**

Affects docker volume create command to create a volume

Command:
```bash
docker -H 10.0.0.1:9998 --tlsverify volume create
```

Response:
```plaintext
[Prisma Cloud] The command volume_create denied for user admin by rule Deny
```

**volume_inspect - Inspect a volume**

Affects docker volume inspect command to display detailed information on one or more volumes

Command:
```bash
docker -H 10.0.0.1:9998 --tlsverify volume inspect f1c7
```

Response:
```plaintext
[Prisma Cloud] The command volume_inspect denied for user admin by rule Deny
```

**volume_remove - Remove a volume**

Affects docker volume rm command to remove one or more volumes
Command:

docker -H 10.0.0.1:9998 --tlsverify volume rm f671

Response:

[Prisma Cloud] The command volume_remove denied for user admin by rule Deny

Networks

For information about the Docker API for networks, see https://docs.docker.com/engine/api/v1.30/#tag/Network.

**network_list - list networks**

Affects docker network ls to list networks

Command:

docker -H 172.17.0.1:9998 --tlsverify network ls

Response:

[Prisma Cloud] The command network_list denied for user admin by rule Default - deny all

**network_inspect - Inspect network**

Affects docker network inspect to display detailed information on one or more networks

Command:

docker -H 172.17.0.1:9998 --tlsverify network inspect 82b1c

Response:

[Prisma Cloud] The command network_inspect denied for user admin by rule Default - deny all

**network_create - Create a network**

Affects docker network create to create a network

Command:

docker -H 172.17.0.1:9998 --tlsverify network create new-network

Response:

[Prisma Cloud] The command network_create denied for user admin by rule Default - deny all

**network_connect - Connect a container to a network**

Affects docker network connect to connect a container to a network
Command:

docker -H 172.17.0.1:9998 --tlsverify network connect new-network container1

Response:

[Prisma Cloud] The command network_connect denied for user admin by rule
Default - deny all

**network_disconnect - Disconnect a container from a network**

Affects docker network disconnect to disconnect a container from a network

Command:

docker -H 172.17.0.1:9998 --tlsverify network disconnect new-network container1

Response:

[Prisma Cloud] The command network_disconnect denied for user admin by rule
Default - deny all

**network_remove - Remove a network**

Affects docker network rm to remove one or more networks

Command:

docker -H 172.17.0.1:9998 --tlsverify network rm new-network

Response:

[Prisma Cloud] The command network_remove denied for user admin by rule
Default - deny all

**Swarm nodes**

For more information about the Docker API for Swarm nodes, see https://docs.docker.com/engine/api/v1.30/#tag/Node.

**node_list - List nodes**

Affects docker node ls command to list nodes in the swarm

Command:

docker -H 172.17.0.1:9998 --tlsverify node ls

Response:

[Prisma Cloud] The command node_list denied for user admin by rule
Default - deny all
node_inspect - Inspect a node
Affects docker node inspect command to inspect a node in the swarm

Command:
```
docker -H 172.17.0.1:9998 --tlsverify node inspect swarm-manager
```

Response:
```
[Prisma Cloud] The command node_inspect denied for user admin by rule Default - deny all
```

Swarm
For more information about the Docker API for Swarm, see https://docs.docker.com/engine/api/v1.30/#tag/Swarm.

swarm_init - Initialize a new swarm
Affects docker swarm init command initialize a swarm.

Command:
```
docker -H 172.17.0.1:9998 --tlsverify swarm init
```

Response:
```
[Prisma Cloud] The command swarm_init denied for user admin by rule Default - deny all
```

swarm_join - Join an existing swarm
Affects docker swarm join command to Join a swarm as a manager node or worker node.

Command:
```
docker -H 172.17.0.1:9998 --tlsverify swarm join --token SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfc8p2is99znpy26u21k1-7p73s1dx5in4tatdymyhg9hu2192.168.99.121:2377
```

Response:
```
[Prisma Cloud] The command swarm_join denied for user admin by rule Default - deny all
```

swarm_leave - Leave a swarm
Affects docker swarm leave command to Remove the current node from the swarm.

Command:
```
docker -H 172.17.0.1:9998 --tlsverify swarm leave
```
Response:

[Prisma Cloud] The command swarm_leave denied for user admin by rule Default - deny all

**swarm_update - update a swarm**

Affects docker swarm update command to update attributes of a swarm

Command:

docker -H 172.17.0.1:9998 --tlsverify swarm update --cert-expiry 70h

Response:

[Prisma Cloud] The command swarm_update denied for user admin by rule Default - deny all

**Swarm services**

For more information about the Docker API for Swarm services, see [https://docs.docker.com/engine/api/v1.30/#tag/Service](https://docs.docker.com/engine/api/v1.30/#tag/Service).

**service_list - List services**

Affects docker service ls command to List services in the swarm.

Command:

docker -H 172.17.0.1:9998 --tlsverify service ls

Response:

[Prisma Cloud] The command service_list denied for user admin by rule Default - deny all

**service_create - Create a service**

Affects docker service create command to Create a new service.

Command:

docker -H 172.17.0.1:9998 --tlsverify service create --name redis redis:3.0.6

Response:

[Prisma Cloud] The command service_create denied for user admin by rule Default - deny all

**service_remove - Remove a service**

Affects docker service rm command to remove a service from the swarm.
Command:

docker -H 172.17.0.1:9998 --tlsverify service rm redis

Response:

[Prisma Cloud] The command service_remove denied for user admin by rule
Default - deny all

**service_inspect - Inspect one or more services**

Affects docker service inspect command to inspect a service

Command:

docker -H 172.17.0.1:9998 --tlsverify service inspect redis

Response:

[Prisma Cloud] The command service_inspect denied for user admin by rule
Default - deny all

**service_update - Update a service**

Affects docker service update command to Update the attributes of a service

Command:

docker -H 172.17.0.1:9998 --tlsverify service update --limit-cpu 2 redis

Response:

[Prisma Cloud] The command service_inspect denied for user admin by rule
Default - deny all

**Tasks**

For more information about the Docker API for tasks, see [https://docs.docker.com/engine/api/v1.30/#tag/Task](https://docs.docker.com/engine/api/v1.30/#tag/Task).

**task_list - List tasks**

Affects docker service where host is deployed. Relevant only for Swarm.

Command:

docker -H 10.0.0.1:9998 --tlsverify service ls

Response:

[Prisma Cloud] The command service_list denied for user admin by rule
Default - deny all

**task_inspect - Inspect a task**

Affects docker service inspect command.
Command:

docker -H 10.0.0.1 --tlsverify inspect redis

Response:

[Prisma Cloud] The command service_inspect denied for user admin by rule Default - deny all

Secrets

Secrets are added in Prisma Cloud 2.0 in accordance with Docker Engine API v1.26.

For more information about the Docker API for secrets, see https://docs.docker.com/engine/api/v1.30/#tag/Secret.

secret_list - List secrets

Affects docker secret ls command used to list secrets.

Command:

docker -H 10.0.0.1:9998 --tlsverify secret ls

Response:

[Prisma Cloud] The command secret_ls denied for user admin by rule Default - deny all

secret_create - Create secrets

Affects docker secret create command used to create secrets.

Command:

docker -H 10.0.0.1:9998 --tlsverify secret create my-secret ./aqsa.json

Response:

[Prisma Cloud] The command secret_create denied for user admin by rule Default - deny all

secretInspect - Inspect secrets

Affects docker secret inspect command used to inspect secrets.

Command:

docker -H 10.0.0.1:9998 --tlsverify secret inspect <id>

Response:

[Prisma Cloud] The command secret_inspect denied for user admin by rule Default - deny all
**secret_remove - Delete secrets**

Affects docker secret rm command used to remove one or more secrets.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify secret rm aqsa.json
```

Response:

```
[Prisma Cloud] The command secret_rm denied for user admin by rule Default - deny all
```

**secret_update - Update a secret**

Affects POST /secrets/[id]/update command used to remove one or more secrets.

Command:

Response:
Defender architecture

Customers often ask how Prisma Cloud Defender really works under the covers. Prisma Cloud leverages Docker’s ability to grant advanced kernel capabilities to enable Defender to protect your whole stack, while being completely containerized and utilizing a least privilege security design.

Defender design

Because we’ve built Prisma Cloud expressly for cloud native stacks, the architecture of our agent (what we call Defender) is quite different. Rather than having to install a kernel module, or modify the host OS at all, Defender instead runs as a Docker container and takes only those specific system privileges required for it to perform it’s job. It does not run as --privileged and instead takes the specific system capabilities of net_admin, sys_admin, sys_ptrace, audit_control, mknod, and setfcap that it needs to run in the host namespace and interact with both it and other containers running on the system. You can see this clearly by inspecting the Defender container:

```bash
# docker inspect twistlock_defender_<VERSION> | grep -e CapAdd -A 7 -e Priv
"CapAdd": ["NET_ADMIN", "SYS_ADMIN", "SYS_PTRACE", "AUDIT_CONTROL", "MKNOD", "SETFCAP" ],

"Privileged": false,
```

This architecture allows Defender to have a near real time view of the activity occurring at the kernel level. Because we also have detailed knowledge of the operations of each container, we can correlate the kernel data with the container data to get a comprehensive view of process, file system, network, and system call activity from the kernel and all the containers running on it. This access also allows us to take preventative actions like stopping compromised containers and blocking anomalous processes and file system writes.

Critically, though, Defender runs as a user mode process. If Defender were to fail (and if that were to happen, it would be restarted immediately), there would be no impact on the containers on the host, nor the host kernel itself. Additionally, we can and do apply cgroups to set hard limits on CPU and memory consumption, guaranteeing it will be a ‘good neighbor’ on the host and not interfere with host performance or stability.

In the event of a communications failure with Console, Defender continues running and enforcing the active policy that was last pushed by the management point. Events that would be pushed back to Console are cached locally until it is once again reachable.

Why not a kernel module?

Given the broad range of security protection Prisma Cloud provides, not just for containers, but also for the hosts they run on, you might assume that we use a kernel module - with all the associated baggage that goes along with that. However, that’s not actually how Prisma Cloud works.

Kernel modules are compiled software components that can be inserted into the kernel at runtime and typically provide enhanced capabilities for low level functionality like process scheduling or file monitoring. Because they run as part of the kernel, these components are very powerful and privileged. This allows them to perform a wide range of functions but also greatly increases the operational and security risks on a
given system. The kernel itself is extensively tested across broad use cases, while these modules are often created by individual companies with far fewer resources and far more narrow test coverage.

Because kernel modules have unrestricted system access, a security flaw in them is a system wide exposure. A single unchecked buffer or other error in such a low level component can lead to the complete compromise of an otherwise well designed and hardened system. Further, kernel modules can introduce significant stability risks to a system. Again, because of their wide access, a poorly performing kernel module that’s frequently called can drag down performance of the entire host, consume excessive resources, and lead to kernel panics. For these reasons, many modern operating systems designed for cloud native apps, like Google Container-Optimized OS, explicitly prevent the usage of kernel modules.

Blocking rules

Defender is responsible for enforcing vulnerability and compliance blocking rules. When a blocking rule is created, Defender moves the original runC binary to a new path and inserts a Prisma Cloud runC shim binary in its place.

When a command to create a container is issued, it propagates down the layers of the container orchestration stack, eventually terminating at runC. Regardless of your environment (Docker, Kubernetes, or OpenShift, etc) and underlying CRI provider, runC does the actual work of instantiating a container.
When starting a container in a Prisma Cloud-protected environment:

1. The Prisma Cloud runC shim binary intercepts calls to the runC binary.
2. The shim binary calls the Defender container to determine whether the new container should be created based on the installed policy.
   - If Defender replies affirmatively, the shim calls the original runC binary to create the container, and then exits.
   - If Defender replies negatively, the shim terminates the request.
   - If Defender does not reply within 60 seconds, the shim calls the original runC binary to create the container and then exits.

The last step guarantees that Defender always fails open, which is important for the resiliency of your environment. Even if the Defender process terminates, becomes unresponsive, or cannot be restarted, a failed Defender will not hinder deployments or the normal operation of a node.
Firewalls

Defender enforces WAF policies (WAAS) and monitors layer 4 traffic (CNNF). In both cases, Defender creates iptables rules on the host so it can observe network traffic. For more information, see CNNF architecture and WAAS architecture.
Telemetry

To drive product improvements, Prisma Cloud captures anonymous data about how our product is used. By default, telemetry is disabled.

No information that is collected can be used to uniquely identify you or your deployment.

Telemetry can be disabled any time.

We never collect IP addresses, host names, user names, container labels, or image tags.

Our telemetry is designed to help us understand how customers use our product at an aggregate level. For example, we detect which features are enabled, the number of containers and images in an environment, and so on.

The legal terms governing telemetry can be found in the Prisma Cloud License Agreement. The Prisma Cloud License Agreement is included with the installation package.

Disabling telemetry

Disable telemetry in Console’s settings page.

STEP 1 | Open Console.

STEP 2 | Go to Manage > System > Intelligence.

STEP 3 | Scroll down to the section Updates and threats feeds.

STEP 4 | Set Anonymously report threats and vulnerabilities to Prisma Cloud to Off.
Configure

After installing Prisma Cloud, configure it to meet your operational and security requirements.

> Rule ordering and pattern matching
> Disaster recovery
> Custom feeds
> Configuring Prisma Cloud’s proxy settings
> Custom certs for Console access
> Configure scanning
> User certificate validity period
> Enable HTTP access to Console
> Set different paths for Defender and Console (with DaemonSets)
> Authenticate to Console with certificates
> Customize terminal output
> Collections
> Tags
> Logon settings
> Reconfigure Prisma Cloud
> Subject Alternative Names
> Credentials store
Rule ordering and pattern matching

Prisma Cloud rules have granular pattern matching capabilities. For example, you could apply a rule to any image with the name *ubuntu*. Or you could apply a rule to all hosts, except those named “*test*”. This article describes how filtering and pattern matching works in Prisma Cloud.

Pattern matching

All rules in Prisma Cloud have resource filters that let you precisely target specific parts of your environment.

These fields let you apply rules based on container name, image name, host name, Docker labels, Kubernetes Pod template labels, Kubernetes Namespace labels, and Kubernetes Deployment labels. To use Kubernetes Namespace and Deployment labels, you must deploy defenders with the following setting enabled: Manage > Defenders > Deploy > DaemonSet > Collect Deployment and Namespace labels. By default, they’re populated with wildcards, which applies the rule to all objects.

You can customize how rules are applied with string matching. When Prisma Cloud encounters a wildcard in a resource name, it evaluates the resource name according to the position of the wildcard.

- If the string starts with a wildcard, it’s evaluated as *string-starts-with*.
- If the string terminates with a wildcard, it’s evaluated as *string-ends-with*.
- If a string is starts and terminates with a wildcard, it’s evaluated as *string-contains*.

For example, if you specify a resource filter of “*foo-resource*”, Prisma Cloud matches that resource to any value that contains the string, such as *example-foo-resource* and *foo-resource-1*. Matching logic is case insensitive.

Individual fields are combined using AND logic. For example, if a rule is configured to apply to hosts called *foo-hosts* and images called *foo-images*, it will apply only to images with names starting with *foo-images* on hosts with names starting with *foo-hosts*.

Wildcards apply rules to all objects of a given type. Constrain the scope of the rule by specifying filters.

In the following example, there are no filters for containers or labels (they’re wildcards). But there are filters for hosts named *foo-hosts* and images named *foo-images*. The effective result is apply this rule anytime the host name starts with foo-hosts and image name starts with foo-images, regardless of the container name or label.
If strings have no wildcards, Prisma Cloud exactly matches the value you enter against the resource string. This gives you precise control over which values match. For example:

- *:latest matches ubuntu:latest or debian:latest.
- If you want to explicitly target just ubuntu:latest from Docker Hub, use docker.io/library/ubuntu:latest.
- *
  _test matches host_sandbox_test and host_preprod_test but doesn’t match host_test_server.

For DNS filtering, Prisma Cloud doesn’t prevent you from entering multiple wildcards per string, but it’s treated the same as if you simply entered the right-most wildcard. The following patterns are equivalent:

```
*.*.b.a == *.b.a
```

Exemptions

While basic string matching makes it easy to manage rules for most scenarios, you sometimes need more sophisticated logic. Prisma Cloud lets you exempt objects from a rule with the minus (-) sign. From example, if you want a rule to apply to all hosts starting with foo-hosts*, except those starting with foo-hosts-exempt*, then you could create the following rule:

When Prisma Cloud evaluates an object against a rule that uses a minus sign, it first skips any object for which there is a match with the exempted object. So, from our example:

1. If the host name starts with foo-hosts-exempt, skip the rule.
2. If the host name starts with foo-hosts AND the image name starts with foo-images, apply the rule.

Rule ordering

For any given feature area, such as vulnerability management or compliance, you might have multiple rules, such as test 1 and test 2.
The entire set of rules in a given feature area is called the policy. The rules in the policy are evaluated from top to bottom, making it easy to understand how policy is applied. When evaluating whether to apply a rule to a given object, Prisma Cloud uses the following logic:

1. Does rule 1 apply to object? If yes, apply action(s) defined in rule and stop. If no, go to 2.
2. Does rule 2 apply to object? If yes, apply action(s) defined in rule and stop. If no, go to 3.
3. ...
4. Apply the built-in Default rule (unless it was removed or modified).

Prisma Cloud evaluates the rule list from top to bottom until it finds a match based on the object filters. When a match is found, it applies the actions in the rule and stops processing further rules. If no match is found, then no action is applied. Sometimes this could mean that an attempted action is blocked (e.g. if no access control rule is matched that allows a user to run a container).

To reorder rules, click on a rule’s hamburger button and drag it to a new position in the list.

### Disabling rules

If you want to test how the system behaves without a particular rule, you can temporarily disable it. Disabling a rule gives you a way to preserve the rule and its configuration, but take it out of service, so that it’s ignored when Prisma Cloud evaluates events against your policy.

To disable a rule, click **Actions > Disable**.
Image names

The canonical name of an image is its **full name** in a format like `registry/repo/image-name`. For example: `1234.dkr.ecr.us-east-1.amazonaws.com/morello/foo-images`. Within Docker itself, these canonical names can be seen by inspecting any given image, like this:

```bash
$ sudo docker inspect morello/foo-images | grep Repo -A 3
"RepoTags": [
  "1234.dkr.ecr.us-east-1.amazonaws.com/morello/foo-images",
]
```

However, there's a special case to be aware of with images sourced from Docker Hub. For those images, the Docker Engine and client do not show the full path in the canonical name; instead it only shows the 'short name' that can be used with Docker Hub and the full name is implied. For example, compare the previous example of an image on AWS ECR, with this image on Docker Hub:

```bash
$ sudo docker inspect morello/docker-whale | grep Repo -A 3
"RepoTags": [
  "morello/docker-whale:latest",
]
```

Note that when the image is from Hub, the canonical name is listed as just the short name (the same name you could use with the Docker client to issue a command like `docker run morello/docker-whale`). For images like this, Prisma Cloud automatically prepends the actual address of the Docker Hub registry (docker.io) and, if necessary, the library repo name as well, even though these values are not shown by Docker itself.

For example, you can run the Alpine image from Docker Hub simply by issuing a Docker client command like `docker run -ti alpine /bin/sh`. The Docker client automatically knows that this means to pull and run the image that has a canonical name of `docker.io/library/alpine:latest`. However, this full canonical name is not exposed by the Docker client when inspecting the image:

```bash
$ sudo docker inspect alpine | grep Repo -A 2
"RepoTags": [
  "alpine:latest"
],
"RepoDigests": [
  "alpine@sha256:1354db23ff5478120c980eca1611a51c9f2b88b61f24283ee8200bf9a54f2e5c"
]
```

But because Prisma Cloud automatically prepends the proper values to compose the canonical name, a rule like this blocks images from Hub running:

```bash
$ docker -H :9998 --tls run -ti alpine /bin/sh
docker: Error response from daemon: [Prisma Cloud] The command container_create denied for user admin by rule Deny - deny all docker.io images.
```
Disaster recovery

Prisma Cloud disaster recovery automatically backs up all data and configuration files periodically. You can view all backups, make new backups, and restore specific backups from the Console UI. You can also restore specific backups using the twistcli command line utility.

Prisma Cloud is implemented with containers that cleanly separate the application from its state and configuration data. To back up a Prisma Cloud installation, only the files in the data directory need to be archived. Because Prisma Cloud containers read their state from the files in the data directory, Prisma Cloud containers do not need to be backed up, and they can be installed and restarted from scratch.

When data recovery is enabled (default), Prisma Cloud archives its data files periodically and copies the backup file to a location you specify. The default path to the data directory is `/var/lib/twistlock`. You can specify a different path to the data directory in `twistlock.cfg` when you install Console.

Configuring automated backups

By default, automated backups are enabled. With automated backups enabled, Prisma Cloud takes a daily, weekly, and monthly snapshots. These are known as system backups.

To specify a different backup directory or to disable automated backups, modify `twistlock.cfg` and install (or reinstall) Prisma Cloud Console. The following configuration options are available:

<table>
<thead>
<tr>
<th>Configuration option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_RECOVERY_ENABLED</td>
<td>Enables or disables automated backups.</td>
</tr>
<tr>
<td>• <code>true</code> — Enables automated backups (default).</td>
<td></td>
</tr>
<tr>
<td>• <code>false</code> — Disables automated backups.</td>
<td></td>
</tr>
<tr>
<td>DATA_RECOVERY_VOLUME</td>
<td>Specifies the directory where backups are saved.</td>
</tr>
<tr>
<td></td>
<td>For example, archives could be saved on durable persistent storage, such as a volume from Amazon Elastic Block Storage (EBS).</td>
</tr>
<tr>
<td></td>
<td>The default value is <code>/var/lib/twistlock-backup</code>.</td>
</tr>
</tbody>
</table>

**STEP 1** | Open `twistlock.cfg` for editing.

**STEP 2** | Scroll down to the Data recovery section.

**STEP 3** | Enable (or disable) automated back up by setting `DATA_RECOVERY_ENABLED` to true (or false).

```
DATA_RECOVERY_ENABLED=true
```

**STEP 4** | Specify the location where backups should be stored.

```
DATA_RECOVERY_VOLUME=/PATH/TO/BACKUP/VOLUME
```

**STEP 5** | Load your new configuration settings.
If you have not installed Prisma Cloud Console yet, follow the regular installation procedure. For more information, see Install Prisma Cloud.

If Prisma Cloud has already been installed on your host, load your new `twistlock.cfg` file by re-running `twistlock.sh`. The following command assumes that `twistlock.sh` and your updated `twistlock.cfg` reside in the same directory.

```
$ sudo ./twistlock.sh console
```

### Making manual backups

Prisma Cloud automatically creates and maintains daily, weekly, and monthly backups. These are known as system backups. You can also make your own backups at any point in time. These are known as manual backups.

**STEP 1 |** Open Console.

**STEP 2 |** Go to Manage > System > Backup & Restore.

**STEP 3 |** Under Manual backups, click Create backup.

**STEP 4 |** Give your backup a name, then click Create.

Your backup file is stored in `/var/lib/twistlock-backup` in the storage volume allocated to Prisma Cloud Console. For a onebox installation, this would simply be the local file system of the host where Console runs. For a cluster, such as Kubernetes, this would be the persistent volume allocated to the Console service.

### Restoring backups from the Console UI

You can restore Console from a backup file directly from within the Console UI. The Console UI lists all available backups.

> You can only restore Console from a backup file whose version exactly matches the current running version of Console. Therefore, if the current running version of Console is 19.11.512, you cannot restore a backup whose version is 19.11.506. To restore a different version of Console, install the Prisma Cloud version that matches your backup version, then follow the procedure here to restore that backup. As long as the specified backup directory (by default, `/var/lib/twistlock-backup`) contains your backup file, you'll be able to restore it.

**STEP 1 |** Open Console.

**STEP 2 |** Go to Manage > System > Backup & Restore.

**STEP 3 |** Click Restore on one of the system or manual backups.

**STEP 4 |** After the database is reloaded from the backup file, restart Console.

For a onebox installation, ssh to the host where Console runs, then run the following command:

```
$ docker restart twistlock_console
```
For a Kubernetes installation, delete the Console pod, and the replication controller will automatically restart it:

// Get the name of Prisma Cloud Console pod:
$ kubectl get po -n twistlock | grep console

// Delete the Prisma Cloud Console pod:
$ kubectl delete po <TWISTLOCK_CONSOLE> -n twistlock

If any new Defenders were installed since the backup was created, restart those Defenders. Otherwise, they might not function properly.

If a Defender created any new runtime models since the backup was created, restart those Defenders. Otherwise, those models might not be visible.

**Restoring backups from twistcli**

You can restore Console from a backup using **twistcli**. Use this restore flow when Console is unresponsive and you cannot access the UI to force a restore to a known good state.

You can only restore Console from a backup file whose version exactly matches the current running version of Console. Therefore, if the current running version of Console is 2.5.88, you cannot restore a backup whose version is 2.5.50. To restore a different version of Console, install the Prisma Cloud version that matches your backup version, then follow the procedure here to restore that backup. As long as the specified backup directory (by default, /var/lib/twistlock-backup) contains your backup file, you’ll be able to restore it.

**Prerequisites:**

- Your host can access the volume where the Prisma Cloud backups are stored. By default, backups are stored in /var/lib/twistlock-backup, although this path might have been customized at install time.
- Your host can access the Prisma Cloud’s data volume. By default, the data volume is located in /var/lib/twistlock, although this path might have been customized at install time.
- Your version of **twistcli** matches the version of the backup you want to restore.

**STEP 1** | Go to the directory where you unpacked the Prisma Cloud release.

**STEP 2** | Run the **twistcli restore** command. Run **twistcli restore --help** to see all arguments.

1. List all available backups. To list all files in the default backup folder (/var/lib/twistlock-backup), run **twistcli restore** without any arguments:

   ```bash
   $ ./twistcli restore
   ```

   To list all backup files in a specific location, run:

   ```bash
   $ ./twistcli restore <PATH/TO/FOLDER>
   ```

2. Choose a file to restore by entering the number that corresponds with the backup file.

   For example:

   ```bash
   aqsa@aqsa-faith: ./twistcli restore --data-recovery-folder /var/lib/twistlock-backup/ 
   Please select from the following: 
   0: backup1 2.5.91 2018-08-07 15:10:10 +0000 UTC
   ```
STEP 3 | After the database is reloaded from the backup file, re-install/restart Console.

For a onebox installation, ssh to the host where Console runs, then rerun the installer:

```
$ sudo ./twistlock.sh -ys onebox
```

For a Kubernetes installation, delete the Console pod, and the replication controller will automatically restart it:

```
// Get the name of Prisma Cloud Console pod:
$ kubectl get po -n twistlock | grep console

// Delete the Prisma Cloud Console pod:
$ kubectl delete po <TWISTLOCK_CONSOLE> -n twistlock
```

- If any new Defenders were installed since the backup was created, restart those Defenders. Otherwise, they might not function properly.
- If a Defender created any new runtime models since the backup was created, restart those Defenders. Otherwise, those models might not be visible.

Downloading backup files

Prisma Cloud Compute lets you download backup files so that they can be copied to another location. Backup files can be downloaded from:

- **Console** — Go to Manage > System > Backup & Restore, and click Actions > Export to download a backup.
- **API** — Use the GET /api/v1/backups/{id} endpoint to download a specific backup, where {id} is the full name of the backup file. An example of a full file name is daily-20.03.140-1583278547.tar.gz. Full file names can be retrieved from Console’s data folder here: /var/lib/twistlock-backup.
Custom feeds

You can supplement the Prisma Cloud Intelligence Stream with your own custom data, including:

- Banned IP addresses.
- Malware signatures.
- Custom vulnerabilities for proprietary software components.
- Allowed CVEs.

For each data type, you can add individual entries to a table from the Console UI, bulk upload a list from a CSV file, or submit a JSON object via the Prisma Cloud API.

Supplementing the IP reputation list

You can supplement the Prisma Cloud Intelligence Stream with your own list of banned IP addresses. Update your custom IP reputation list from the Console UI. You can specify one entry at a time, or do a bulk upload from a CSV file. The maximum file size is 20MB.

The first line in your CSV file must be a header record that contains the field names. Specify one IP address per line. For example:

<table>
<thead>
<tr>
<th>ip</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.104.125.48</td>
</tr>
<tr>
<td>101.200.81.187</td>
</tr>
<tr>
<td>103.19.89.118</td>
</tr>
</tbody>
</table>

**STEP 1** | Open Console.

**STEP 2** | Go to Manage > System > Custom Feeds.

**STEP 3** | Click IP Reputation Lists, and either click Add or Import CSV.

Your list of banned IP addresses is immediately enforced when your data is imported. A default runtime defense rule, Default - detect suspicious runtime behavior, logs an alert when a container tries to connect to a banned IP address.

**STEP 4** | Review the default rule

Go to Defend > Runtime > [Container Policy | Host Policy], then click manage for the Default - detect suspicious runtime behavior rule. You should see that Prisma Cloud Advanced Threat Protection is set to On.

Supplementing the malware data feed

You can supplement the Prisma Cloud Intelligence Stream with your own custom malware data. Update your custom list of malware signatures from the Console UI. You can specify one entry at a time, or do a bulk upload from a CSV file. The maximum file size is 20MB.

*Malware scanning and detection is supported for Linux container images only. Windows containers are not supported.*
The first line in your CSV file must be a header record that contains the field names. For malware data, specify the MD5, followed by the malware name. Specify one entry per line. For example:

```
md5, name
194836f0be0f121a25b145e55e80ce22, evil-malware
0aeb0c816a81a6ac45776d6b56dd70, evil-binary
33cc273ae3aa8bce6a22c92e7d11f63a, bigevil
```

**STEP 1** | Open Console.

**STEP 2** | Go to Manage > System > Custom Feeds.

**STEP 3** | Click Upload Malware Data, and either click Add or Import CSV.

Your custom malware data is used in all subsequent image scans. It is also used immediately by the runtime defense file system sensor, which assesses all writes to the host and container file system.

**STEP 4** | Review the default rule.

A default runtime defense rule, Default - detect suspicious runtime behavior, logs an alert when malware is detected using signatures from Prisma Cloud’s data set or your custom data set.

To review the default rule, go to Defend > Runtime > [Container Policy | Host Policy], then click manage for the Default - detect suspicious runtime behavior rule. You should see that Prisma Cloud Advanced Threat Protection is set to On.

**Globally allowing CVEs**

Some organizations have very sophisticated CI pipelines that encompass many teams and products. When a security team concludes that a CVE doesn’t impact your organization, they want to dismiss it globally without having to manage individual rules. Managing exceptions on a per rule basis requires a lot of manual effort.

The CVE Allow List lets you globally allow CVEs system-wide. Any entry in the CVE Allow List affects all flows in the product, including twistcli, the Jenkins plugin, registry scanning, deployment blocking, Vulnerability Explorer, and so on. Adding a CVE to this list effectively filters it out from the data in the Prisma Cloud Intelligence Stream before it’s used by the scanner.

The CVE Allow List takes precedence over any rule that’s been created under Defend > Vulnerabilities menu. It is a feature designed to complement rules. Rules also let you allow a CVE, but more granularly, by scoping them to specific resources or parts of your environment.

**STEP 1** | Open Console.

**STEP 2** | Go to Manage > System > Custom Feeds.

**STEP 3** | Click CVE Allow List, and either click Add or Import CSV.

**Test Prisma Cloud’s malware detection capabilities**

Safely simulate malware in your environment to test Prisma Cloud’s malware detection capabilities.

**Configure a custom malware feed**

Set up a custom feed by uploading the provided CSV file to Prisma Cloud Console. This file specifies the MD5 signature for a file that will be considered malware for the purposes of this demo.
STEP 1 | Download *malware.csv*.

STEP 2 | In Console, go to **Manage > System > Custom Feeds > Malware Signatures**.

STEP 3 | Click **Import CSV**, and upload *malware.csv*.

**Detect malware at runtime**

Test how Prisma Cloud detects malware being downloaded into a container at runtime.

**Prerequisites:** The default runtime rule, Default - alert on suspicious runtime behavior under **Defend > Runtime > Container Policy** is in place. If you have deleted or changed the default rule, create a new one.

1. Go to **Defend > Runtime > Container Policy**, and click **Add rule**.
2. Enter a name for the rule.
3. In the **General** tab, verify Prisma Cloud Advanced Threat Protection is **On**.
4. In each of the **Process**, **Networking**, **File System**, and **System Calls** tabs, set **Effect** to **Alert**.

STEP 1 | Run a container and download malware into it.

```
$ docker run -ti alpine sh
  # wget https://cdn.twistlock.com/docs/attachments/evil
```

STEP 2 | Look at resulting audit. Open Console and browse to **Monitor > Events > Container Audits**. You will see a file system audit that says malware was detected.

![Audit Image]

---

Get created jevil, which is detected as evil-malware-demo malware in a custom malware feed.
Configuring Prisma Cloud’s proxy settings

In some environments, access to the Internet must go through a proxy. Prisma Cloud can be configured to route requests through your proxy. Proxy settings are applied to both Console and Defender containers. Proxy settings are configured in the UI after Console is installed. Console immediately starts using your settings after saving them. Any Defenders deployed after saving your settings will use your proxy settings. Any Defenders deployed before saving your settings must be redeployed.

**Console**

Console has a number of connections that might traverse a proxy.

- Retrieving Intelligence Stream updates.
- Connecting to services, such as Slack and JIRA, to push alerts.

**Defenders**

Defender has a number of connections that might traverse a proxy.

- Connecting to Console. If you deploy Defenders in a remote region, they might need to connect to Console through a proxy.
- Connecting to external systems, such as Docker Hub or Google Container Registry, for scanning.
- Connecting to your secrets store to retrieve secrets for injection into your containers.

**Settings**

A number of settings let you specify how Prisma Cloud interfaces with your proxy.
Proxy bypass
You can provide a list of addresses that Prisma Cloud can contact directly without connecting through the proxy. Specify DNS names, IP addresses, or a combination of both. Specifying a block of IP addresses in CIDR notation is currently not supported.

CA certificate
Console verifies server certificates for all TLS connections. With TLS intercept proxies, the connection from Console to the Internet passes through a proxy, which may be transparent. To facilitate traffic inspection, the proxy terminates the TLS connection and establishes a new connection to the final destination.

If you have a TLS intercept proxy, it will break the Console’s ability to connect to external services, because Console won’t be able to verify the proxy’s certificate. To get Console to trust the proxy, provide the CA certificates for Console to trust.

Proxy authentication
If egress connections through your proxy require authentication, you can provide the credentials in Prisma Cloud’s proxy settings. Prisma Cloud supports Basic authentication for the Proxy-Authenticate challenge-response framework defined in RFC 7235. When you provide a username and password, Prisma Cloud submits the credentials in the request’s Proxy-Authorization header.

Configuring proxy settings
Configure your proxy settings in Console.

STEP 1 | Open Console, and go to Manage > System > Proxy.

STEP 2 | In HTTP Proxy, enter the address of the web proxy. Specify the address in the following format: <PROTOCOL>://<IP_ADDR|DNS_NAME>:<PORT>, such as http://proxyserver.company.com:8080.

STEP 3 | (Optional) In No Proxy, enter addresses that Prisma Cloud can access directly without connecting to the proxy. Enter a list of individual IP addresses and fully-qualified domain names. IP blocks in CIDR notation are not supported.

STEP 4 | (Optional) For TLS intercept proxies, enter the root trusted authority certificate, in PEM format, that Console should trust.

STEP 5 | (Optional) If your proxy requires authentication, enter a username and password.

STEP 6 | Click Save.

STEP 7 | Redeploy your Defenders to propagate updated proxy settings to them.

Console does not need to be restarted. After proxy settings are saved, Console automatically uses the settings the next time it establishes a connection.

Any newly deployed Defenders will use your proxy settings.

Any already deployed Defenders must be redeployed. For single Container Defenders, uninstall then reinstall. For Defender DaemonSets, regenerate the DaemonSet YAML, then redeploy.

$ kubectl apply -f defender.yaml
Custom certs for Console access

You can secure access to Console with your own digital certificate. By default, Prisma Cloud secures access to Console’s web portal and API with a self-signed certificate.

The self-managed certificate generated by Console is valid for three years. A month prior to expiration, Prisma Cloud will let you rotate it (a banner will appear at the top of the UI). After rotating Console’s certificate, you must manually redeploy your Defenders.

When you access Console’s web portal with this setup, for example, the browser flags the portal as untrusted with a warning message. The following screenshot shows the warning message in Chrome:
You can resolve these warnings by installing your own certificate that proves your server’s identity to the client. With the proper certificate, users are taken directly to Console, and the green padlock in the address bar indicates that the site is trusted.

You can resolve these warnings by installing your own certificate that proves your server’s identity to the client. With the proper certificate, users are taken directly to Console, and the green padlock in the address bar indicates that the site is trusted.

Creating certificates is outside the scope of this article. For more information about how SSL and certificates secure a site, see How does HTTPS actually work.
Configuration options

Prisma Cloud secures the communication between various actors and entities with certificates. These certs are automatically generated and self-signed during the Prisma Cloud install process. They secure communication between:

- Users and the Console web portal.
- Users and the Console API.
- Console and Defenders.
- Console and the Prisma Cloud Intelligence Stream.

The following options control the properties of the certificates generated during the install process. The default values for these options are typically adequate.

Note that these settings only change the values used when creating self-signed certificates. Thus, users accessing the Console will still see warning messages because the certificates are not signed by a trusted certificate authority (CA). To configure the Console to use a certificate signed by a trusted CA, follow the steps later in this article.

These options can be found in `twistlock.cfg` under the General Configuration section:

<table>
<thead>
<tr>
<th>Configuration option</th>
<th>Description</th>
</tr>
</thead>
</table>
| **CONSOLE_CN**       | Specifies the Common Name to be used in the certificate generated by Prisma Cloud for the host that runs Console. The Common Name is typically your host name plus domain name. For example, it might be www.example.com or example.com.  
(Default) By default, the Common Name is assigned the output from the command `hostname --fqdn`. |
| **DEFENDER_CN**      | Specifies the Common Name to be used in the certificate generated by Prisma Cloud for the hosts that run Defender.  
(Default) By default, the Common Name is assigned the output from the command `hostname --fqdn`. |

You can also control the Subject Alternative Names (SANs) in Console’s certificate.

Securing access to Console with custom certs

Secure access to Console with your own custom certs.

Prerequisites:

- Your certs have been generated by a commercial Certificate Authority (CA) or with your own Public Key Infrastructure (PKI). You should have the following files on hand:
  - A `.pem` file, which contains your certificate and your Certificate Authority's intermediate certificates.
  - A `.key` file, which contains your private key.

**STEP 1** | Have your signed certificate (.pem file) and private key (.key file) ready to be accessed and uploaded to Console.

*Make sure that the private key starts and ends with:*

```
-----BEGIN PRIVATE KEY-----
```
STEP 2 | Open Prisma Cloud Console in a browser.

STEP 3 | Navigate to Manage > Authentication > System Certificates.

STEP 4 | Concatenate your public cert and private key into a single PEM file.

```bash
$ cat server.crt server.key > server-cert.pem
```

STEP 5 | Upload the PEM file into the TLS Certificate for Console section.

STEP 6 | Click Save

STEP 7 | Verify that your certs have been correctly installed.

Open your browser, and navigate to: https://<CONSOLE_HOSTNAME>:8083

If you see the locked padlock icon, you have installed your certs correctly.
Configure scanning

You can specify how often Prisma Cloud scans your environment for vulnerabilities and compliance issues. By default, Prisma Cloud scans your environment every 24 hours. Images are re-scanned when changes are detected. For example, pulling a new image triggers a scan.

Prisma Cloud scans for vulnerabilities and/or compliance issues in:

- Images
- PCF Blobstores
- Containers
- Serverless functions
- Hosts
- Cloud platforms
- Registries
- VM images

Scan intervals can be separately configured for each type of object.

Configuring scan intervals

The scan frequency is configurable. By default, Prisma Cloud scans your environment every 24 hours.

STEP 1 | Open Console.

STEP 2 | Go to Manage > System > Scan.

STEP 3 | Scroll down to the Scheduling section.

STEP 4 | Set the scan intervals for each type according to your requirements.

Scan intervals are specified in hours.

STEP 5 | Scroll to the bottom of the page, and click Save.

Last scan time

Console reports the last time Defender scanned your environment. Go to Manage > Defenders > Manage, and click a row in the table to get a detailed status report for each deployed Defender. The status column shows the last time Defender scanned the containers and images on the host where it runs. When Defender is delegated the registry scanner role, you can also see the last time your registry was scanned.
Defender roles are displayed in the Manage > Defenders > Manage table. The following screenshot shows that Defender on host ian-23 is a registry scanner.

Scan performance

Scanning for malware in archives in container images consumes a lot of resources. The scanner unpacks each archive to search for malicious software. Checksums must be individually calculated for each file. Because of the performance impact and the way containers tend to be used, malware in archives is an unlikely threat. As such, Scan for malware within archives in images is disabled by default.

If this option is enabled, Prisma Cloud supports the following archive file types.

- ZIP
- GZ
- TAR
- WAR
- JAR
- EAR

Note: If the archive is over 512Mb, Prisma Cloud will not scan it.

Scan JavaScript components in manifest but not on disk

The purpose of this option is to show vulnerabilities in dependencies that might not exist on disk (which are often development dependencies).

Most Node.js packages contain a package.json that lists all of its dependencies (both dependencies, and devDependencies). When parsing a Node.js package discovered during a scan, if this option is enabled, Prisma Cloud appends the all packages found in each package.json to the list of packages to be assessed for
vulnerabilities. This option isn't recommended for production scenarios because it can generate a significant number of false positives.

If this option is disabled (default), Prisma Cloud only evaluates the packages that are actually found on disk during scan. This is the recommended setting for production scenarios.

When scanning images with twistcli, use `--include-js-dependencies` to enable this option.

### Unrated vulnerabilities

When **Show vulnerabilities that are of negligible severity** is enabled, the scanner reports CVEs that aren't scored yet or have a negligible severity. Negligible severity vulnerabilities don't pose a security risk, and are often designated with a status of "will not fix" or similar labels by the vendor. They are typically theoretical, require a very special (unlikely) situation to be exploited, or cause no real damage when exploited.

By default, this setting is disabled to strip unactionable noise from your scan reports.

### Orchestration

Kubernetes and other orchestrators have control plane components implemented as containers. By default, Prisma Cloud doesn't scan orchestrator utility containers for vulnerability and compliance issues.
User certificate validity period

User certificates identify a user, and are used to enforce access control policies. You can control how long user certificates are valid. By default, user certificates are valid for 365 days.

Configuring the validity period of user certificates

Configure the validity period of user certs.

STEP 1 | Open Console.

STEP 2 | Go to Manage > Authentication > Certificates.

STEP 3 | Under Configuration, enter a new value for Number of days until expiration of certificate.

STEP 4 | Click Save.

Expired user certificates

The following message is printed when you try to authenticate with an expired certificate. This example command tries to run docker ps on a remote host named prod_host1.

```
$ docker --tlsverify -H prod_host1:9998 ps
The server probably has client authentication (--tlsverify) enabled.
Please check your TLS client certification settings
```

Generating new certificates

When your certificates expire, you can generate new ones.

STEP 1 | Go to Console.

STEP 2 | Log in with your credentials to reauthenticate with Console. This step generates fresh certificates.

- If you integrated Prisma Cloud with LDAP, log in with your LDAP credentials.
- If you integrated with SAML, log in with your SAML credentials.
- If you are using Prisma Cloud users, log in with your Prisma Cloud user credentials.

STEP 3 | On the left menu, click Manage > Authentication > Credentials. Non-admin users are taken directly to this page.

STEP 4 | Copy the installation script, and run it on your local machine.

The script installs fresh certificates on your machine.

STEP 5 | Verify that your certs are valid by running a Docker command on a host protected by Defender.

```
$ docker --tlsverify -H prod_host1:9998 ps
```
Enable HTTP access to Console

By default, Prisma Cloud only creates an HTTPS listener for access to Console. In some circumstances, you may wish to enable an HTTP listener as well. Notice that accessing Console over plain, unencrypted HTTP isn’t recommended, as sensitive information can be exposed.

Enabling an HTTP listener simply requires providing a value for it in twistlock.cfg. At first, your configuration file would look like this:

```
#############################################
# Network configuration
#############################################
# Each port must be set to a unique value (multiple services cannot share the same port)
##### Management console ports ####
# Sets the ports that the Prisma Cloud management website listens on
# The system that you use to configure Prisma Cloud must be able to connect to the Prisma Cloud Console on these ports
# To disable a listener, leave the value empty (e.g. MANAGEMENT_PORT_HTTP=)
# Accessing Console over plain, unencrypted HTTP isn't recommended, as sensitive information can be exposed
MANAGEMENT_PORT_HTTP=
MANAGEMENT_PORT_HTTPS=8083
```

To enable the HTTP listener, your configuration file should look like this:

```
#############################################
# Network configuration
#############################################
# Each port must be set to a unique value (multiple services cannot share the same port)
##### Management console ports ####
# Sets the ports that the Prisma Cloud management website listens on
# The system that you use to configure Prisma Cloud must be able to connect to the Prisma Cloud Console on these ports
# To enable the HTTP listener, set the value of MANAGEMENT_PORT_HTTP (e.g. MANAGEMENT_PORT_HTTP=8081)
# Accessing Console over plain, unencrypted HTTP isn't recommended, as sensitive information can be exposed
MANAGEMENT_PORT_HTTP=8081
MANAGEMENT_PORT_HTTPS=8083
```

After you’ve updated the configuration file, just rerun `twistlock.sh` for the changes to take effect. For example:

```
$ sudo ./twistlock.sh -s console
```
Set different paths for Defender and Console (with DaemonSets)

When using daemon sets, Console is set up to store the Prisma Cloud config under /opt/twistlock. By default, it uses this same config when installing the defenders. This article describes a work around solution to be able to set up different config paths for Console and Defenders using daemon sets

**STEP 1 | Download Daemonset configurations for Defender.**

The API to download Daemonset Configuration is:

```
/api/v1/defenders/daemonset.yaml?registry=${registry}&type=${DEFENDER_TYPE}&consoleaddr=${consoleaddr}&namespace=${namespace}&orchestration=${orchestration}&ubuntu=${os_ubuntu}"
```

The parameters are:

- **registry** --
  the registry from where Kubernetes gets the image, where you pushed the image. In the example above, the value will be "gcr.io/projectA/

- **type** --
  defender type - Daemon Set Docker on Linux or Daemon Set Kubernetes Node. (Daemon set Docker on Linux is the regular default Defender type, called in the UI Docker. Only difference being, unlike the default Defender, it does not listen to incoming traffic.

- **consoleaddr** --
  Name or IP address that Defenders use to connect to Console.

- **namespace** --
  the default when using the script is twistlock, but you can use whatever you want.

- **orchestration** --
  OpenShift or Kubernetes

- **ubuntu** --
  (ubuntu=true \ ubuntu=false), states if the cluster is running on ubuntu OS or not. If not provided, it’s assumed to be false.

**STEP 2 | Edit the yaml file.**

Make the necessary changes in this yaml file and upload this modified version of the yaml to the K8 controller.
Authenticate to Console with certificates

Prisma Cloud supports certificate-based authentication for the Console UI and the API. Prisma Cloud has always provided username / password based authentication. In addition to that, Prisma Cloud also supports certificate based authentication for the Console UI and the API. This is especially useful for those in government and financial services, who use multi-factor authentication technologies built on x.509 certificates. This is applicable to users authenticating via Active Directory accounts as well. This feature allows customers to be able to control the trusted CAs for signing certificates for authentication.

Setting up your certs

This procedure shows you how to set up Prisma Cloud for certificate-based authentication.

If you’re using certificates to authenticate against Active Directory accounts, Prisma Cloud uses the UserPrincipalName field in the SAN to match the certificate to the user in Active Directory. This is the same process used by Windows clients for authentication, so for most customers, the existing smart card certificates you’re already using can also be used for authentication to Prisma Cloud.

STEP 1 | Save the CA certificate(s) used to sign the certificates that you’ll use for authentication to Prisma Cloud.

The certificate has to be in PEM format. If you have multiple CAs that issue certificates to your users, concatenate their PEM files together. For example, if you have Issuing CA 1 and Issuing CA 2, create a combined PEM file like this:

```
$ cat issuing-ca-1.pem issuing-ca-2.pem > issuing-cas.pem
```

STEP 2 | Log into Console, and go to Manage > Authentication > System Certificates.

STEP 3 | Set Advanced certificate configuration to Show.

STEP 4 | Scroll down to Console Authentication, and upload your CA certificate(s) in PEM format.

STEP 5 | Click Save.

STEP 6 | Open Console login page in your browser. When prompted select your user certificate.
What’s next?

See Assigning roles to learn how to add users and assign roles to them.
Customize terminal output

Prisma Cloud lets you create rules that block access to resources or block the deployment of non-compliant containers.

For example, you might create a rule that blocks the deployment of any image that has critical severity vulnerabilities. By default, when you try to run non-compliant image, Prisma Cloud returns a terse response:

```
# docker -H :9998 --tls run -ti morello/docker-whale
docker: Error response from daemon: [Prisma Cloud] operation blocked by policy: (test-compliance), host has 19 compliance issues.
```

To help the operator better understand how to handle a blocked action, you can enhance Prisma Cloud's default response by

- Appending a custom message to the default message. For example, you could tell operators where to go to open a ticket.
- Configuring Prisma Cloud to return an itemized list of compliance issues rather than just a summary. This way, the operator does not need to contact the security team to determine which issues are preventing deployment. They are explicitly listed in the response.

Enhanced terminal output is available for rules created under:

- Defend > Vulnerabilities > Policy
- Defend > Compliance > Policy
- Defend > Access (Docker Engine, Docker Swarm, and Kubernetes access control rules).

Specifying a custom message

This procedure shows you how to create an access control rule that blocks all users from running the `container_create` operation. You will configure the rule to emit the following custom message when an action is blocked:

```
Contact admin@example.com to get additional privileges
```

Although this procedure is specific to access control rules, the process for configuring custom messages for vulnerability and compliance rules is the same.

**STEP 1 |** Open Console.

**STEP 2 |** Go to Defend > Access > Docker, then click New Docker rule.

**STEP 3 |** In the new rule dialog, enter the following information:

1. In Rule name, enter a name.
2. Set Effect to Deny.
3. In Show, uncheck All to deselect all actions.
4. In Actions, check container_create.
5. Click on the Advanced tab.

6. In Custom message for blocked requests, enter Contact admin@example.com to get additional privileges.

7. Click Save.

STEP 4 | Test your setup by running a command that violates your access control rule.

1. Install your client certs.

   For more information, see Configure Docker client variables.

2. Try to run a container on a host protected by Prisma Cloud:

   ```bash
   $ docker --tlsverify -H <HOST>:9998 run ubuntu:latest
docker: Error response from daemon: [Prisma Cloud] The command container_create denied for user aqsa by rule Block create. Contact admin@example.com to get additional privileges. See 'docker run --help'.
   ```

   Where <HOST> is the hostname or IP address for a host running Defender.

Output itemized list of compliance issues

You can configure vulnerability and compliance rules to return a detailed list of issues when Prisma Cloud blocks a deployment.

In this procedure, you create a vulnerability rule that prevents the deployment of any image that contains any type of vulnerable package.

Although this procedure is specific to vulnerability rules, the process for compliance rules is the same.

STEP 1 | Open Console.

STEP 2 | Create a new vulnerability rule (Defend > Vulnerabilities > Policy) or compliance rule (Defend > Compliance > Policy).

STEP 3 | In the new rule dialog, enter the following information:
1. Enter a rule name.
2. Specify conditions that trigger a block action.
   
   For example, for the Image contains vulnerable OS packages condition in a vulnerability rule, set the Action to Block and set the Severity threshold to Low.
3. Set Terminal output verbosity for blocked requests to Detailed.
4. Click Save.

**STEP 4 | Test your setup by deploying an image with vulnerabilities.**

On a host protected by Prisma Cloud, run an image with vulnerabilities.

```
$ docker run --rm -it ubuntu:14.04 sh
docker: Error response from daemon: [Prisma Cloud] Image operation blocked by policy: (sdf), has 44 vulnerabilities, [low:25 medium:19].

<table>
<thead>
<tr>
<th>Image</th>
<th>ID</th>
<th>CVE</th>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubuntu:14.04</td>
<td>4333f1</td>
<td>CVE-2017-2518</td>
<td>sqlite3</td>
<td>3.8.2-1ubuntu2.1</td>
</tr>
<tr>
<td>medium</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| ubuntu:14.04   | 4333f1   | CVE-2017-6512   | perl      | 5.18.2-2ubuntu1.1 |
| medium         |          |                 |           |         |

.| . | . |
Collections

Collections are predefined filters for segments of your environment. They are centrally defined, and can be used in rules and views across the product.

Collections can be used to:

- Partition views. They provide a convenient way to browse data from related resources.
- Optionally enforce which views specific users and groups can see. They can control access to data on a need-to-know basis. These are known as assigned collections.

While a single Console manages data from Defenders spread across all hosts, collections let you segment that data into different views based on attributes. Collections are created with pattern matching expressions that are evaluated against attributes such as image name, container name, host name, labels, function name, namespace, and more. AWS tags are supported as part of the Collection label attribute.

Collections are useful when you have large container deployments with multiple teams working on multiple apps all in the same environment. For example, you might have a single Kubernetes cluster that runs a shopping app, a travel app, and an expenses app. Different teams might be responsible for the development and operation of each app. An internal tools team might be responsible for the travel and expenses app, while the product team runs the shopping app.

Selecting a collection reduces the scope displayed in Console to just the relevant components. The developer for the travel app, for example, only cares about vulnerabilities in the images that make up the travel app. All other vulnerabilities are just noise. Collections help focus the data.

Creating collections

You can create as many collections as you like. Collections cannot be nested. When using Tenant Projects, Collections are created in each of the tenant Projects. When using Scale Projects, Collections are created in the Central Console.

Filtering by cloud account ID for Azure Container Instances isn’t currently supported.

STEP 1 | To create a collection, navigate to Manage > Collections and Tags > Collections.

Prisma Cloud ships with a built-in set called All collections that is not editable. The Default collection contains all objects in the system. It is effectively the same as manually creating a collection manually with a wildcard (*) for each resource type (Containers, Images, Hosts, Labels).

STEP 2 | Click Add collection.

STEP 3 | In the Create a new collection dialog, enter a name, then specify a filter.

For example, create a collection called Raspberry images that shows all raspberry images in the namespace fruit. Pick a color for easy visibility and differentiation.
Create a new collection

<table>
<thead>
<tr>
<th>Name</th>
<th>Raspberry images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Enter a description</td>
</tr>
<tr>
<td>Color</td>
<td></td>
</tr>
<tr>
<td>Containers</td>
<td>* x Add a container</td>
</tr>
<tr>
<td>Images</td>
<td>raspberry* x Add an image</td>
</tr>
<tr>
<td>Hosts</td>
<td>* x Add a host</td>
</tr>
<tr>
<td>Labels</td>
<td>* x Add a label</td>
</tr>
<tr>
<td>Apps</td>
<td>* x Add an app</td>
</tr>
<tr>
<td>Functions</td>
<td>* x Add a function</td>
</tr>
<tr>
<td>RASP Defenders</td>
<td>* x Add a RASP Defender ID</td>
</tr>
<tr>
<td>Namespaces</td>
<td>fruit x Add a namespace</td>
</tr>
</tbody>
</table>

This collection selects all images that start with the string *raspberry*. You can also create a collection to exclude a set of images. For more information on syntax that can be used inside the filter fields (Containers, Images, Hosts, and Labels), see Rule ordering and pattern matching.

You cannot have collections that specify both containers and images. You must leave a wildcard in one of the fields, or else the collection won't be applied correctly. If you want to create collections that apply to both a container and an image, create two separate collections. The first collection should only include the container name, the second should only include the image name. Filtering on both collections at the same time will yield the desired result.

**STEP 4 | Click Save.**

**Assigned collections**

Collections provide a light-weight mechanism to provision least-privilege access to the resources in your environment. You can assign collections to specific users and groups to limit their view of data and resources in the environment.

Projects is the other mechanism for partitioning your environment. Projects are Prisma Cloud’s solution for multi-tenancy. They let you provision multiple independent environments, and federate them behind a single Console URL, interface, and API. Projects take more effort to deploy than collections. Collections and Projects can work together. Collections can be utilized in both non-Project and Project-enabled environments.

By default, users and groups can access all collections and are not assigned with any collection.
Users with admin or operator roles can always see all resources in the system. They can also see all collections, and utilize them to filter views. When creating users or groups with the admin or operator role, there is no option for assigning collections.

When creating users or groups with any other role, admins can optionally assign one more collections. These users can only see the resources in the collections they've been assigned.

Collections cannot be deleted as long as they've been assigned to users or groups. This enforcement mechanism ensures that users and groups are never left stateless. Click on a specific collection to see who is using them.
Assigning collections

Assign collections to specific users and groups to restrict their view of data in the environment. Collections can be assigned to local users, LDAP users, and SAML users. Collections can also be assigned to LDAP and SAML groups. They cannot be assigned to local groups.

When using Projects, Collections can only be assigned to users on each project. Users of the Central Console have access to all projects, and cannot be limited with assigned collections.

Prerequisites:

- You’ve already created one or more collections.
- (Optional) You’ve integrated Prisma Cloud with a directory service or SAML IdP.

**STEP 1** | Open Console, and go to Manage > Authentication > {Users | Groups}.

**STEP 2** | Click Add users or Add group.

**STEP 3** | Select the Auditor or DevOps User role.

**STEP 4** | In Permissions, select one or more collections. If left unspecified, the default permissions is All collections.

**STEP 5** | Click Save.
Selecting a collection

Collections filter data in the Monitor section of Console.

When a collection (or multiple collections) are selected, only the objects that match the filter are shown in those views. When a collection is selected, it remains selected for all views until it is explicitly disabled.

To select a collection, go to any view under Monitor. In the Collections drop-down list in the top right of the view, select a collection. In the following screenshot, the view is filtered based on the collection named google images, which shows all images that contain the string google_containers.

![Monitor / Vulnerabilities](image)

When multiple collections are selected, the effective scope is the union of each individual query.

*Individual filters on each collection aren’t applicable to all views. For example, a collection created with only functions won’t include any resources when viewing hosts results. Similarly, a collection created with hosts won’t filter images by hosts when viewing image results.*

![Monitor / Vulnerabilities](image)

The Collections column shows to which collection a resource belongs. The color assigned to a collection distinguishes objects that belong to specific collections. This is useful when multiple collections are
Collections limitations

The different views under the Console are filtered by different resource types of the collections.

If your collection is created based on a resource that is not included in the resource types relevant to the view you wish to filter, filtering by this collection should yield empty results.

<table>
<thead>
<tr>
<th>Section</th>
<th>View</th>
<th>Supported resources in collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor/</td>
<td>Images</td>
<td>Images, Hosts, Namespaces, Clusters, Labels, Cloud Account IDs</td>
</tr>
<tr>
<td>Vulnerabilities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor/</td>
<td>Containers</td>
<td>Images, Containers, Hosts, Namespaces, Clusters, Labels, Cloud Account IDs</td>
</tr>
<tr>
<td>Compliance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor/</td>
<td>Hosts</td>
<td>Hosts, Clusters, Labels, Cloud Account IDs</td>
</tr>
<tr>
<td>Vulnerabilities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor/</td>
<td>VM images</td>
<td>VM images (under Images), Cloud Account IDs</td>
</tr>
<tr>
<td>Compliance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor/</td>
<td>Functions</td>
<td>Functions, Cloud Account IDs</td>
</tr>
<tr>
<td>Vulnerabilities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor/</td>
<td>Code repositories</td>
<td>Code repositories</td>
</tr>
<tr>
<td>Compliance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor/</td>
<td>PCF Blobstore</td>
<td>Hosts (of the scanner host), Cloud Account IDs</td>
</tr>
<tr>
<td>Vulnerabilities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor/</td>
<td>Vulnerability</td>
<td>Images, Hosts, Clusters, Labels, Functions, Cloud Account IDs</td>
</tr>
<tr>
<td>Explorer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor/</td>
<td>Cloud Discovery</td>
<td>Cloud Account IDs</td>
</tr>
<tr>
<td>Compliance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor/</td>
<td>Cloud Compliance</td>
<td>Cloud Account IDs</td>
</tr>
<tr>
<td>Compliance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>View</td>
<td>Supported resources in collection</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Monitor/Compliance</td>
<td>Compliance Explorer</td>
<td>Images, Hosts, Namespaces, Clusters, Labels, Cloud Account IDs</td>
</tr>
<tr>
<td>Monitor/Events</td>
<td>Container audits</td>
<td>Images, Containers, Namespaces, Container Deployment Labels (under Labels), Cloud Account IDs</td>
</tr>
<tr>
<td>Monitor/Events</td>
<td>Host audits</td>
<td>Hosts, Labels, Cloud Account IDs</td>
</tr>
<tr>
<td>Monitor/Events</td>
<td>Serverless audits</td>
<td>Functions, Cloud Account IDs</td>
</tr>
<tr>
<td>Monitor/Events</td>
<td>App Embedded audits</td>
<td>App IDs (App Embedded), Cloud Account IDs</td>
</tr>
<tr>
<td>Monitor/Runtime</td>
<td>Container incidents</td>
<td>Images, Containers, Hosts, Namespaces, Cloud Account IDs</td>
</tr>
<tr>
<td>Monitor/Runtime</td>
<td>Host incidents</td>
<td>Hosts, Cloud Account IDs</td>
</tr>
<tr>
<td>Monitor/Runtime</td>
<td>Serverless incidents</td>
<td>Functions, Cloud Account IDs</td>
</tr>
<tr>
<td>Monitor/Runtime</td>
<td>App Embedded incidents</td>
<td>App IDs (App Embedded), Cloud Account IDs</td>
</tr>
<tr>
<td>Monitor/Runtime</td>
<td>Container models</td>
<td>Images, Namespaces, Clusters, Cloud Account IDs</td>
</tr>
<tr>
<td>Monitor/Runtime</td>
<td>Host Observations</td>
<td>Hosts, Clusters, AWS tags (under Labels), OS tags (under Labels), Cloud Account IDs</td>
</tr>
<tr>
<td>Radar</td>
<td>Containers Radar</td>
<td>Images, Containers, Hosts, Namespaces, Clusters, Labels, Cloud Account IDs</td>
</tr>
<tr>
<td>Radar</td>
<td>Hosts Radar</td>
<td>Hosts, Clusters, AWS tags (under Labels), OS tags (under Labels), Cloud Account IDs</td>
</tr>
<tr>
<td>Radar</td>
<td>Serverless Radar</td>
<td>Functions</td>
</tr>
<tr>
<td>Manage</td>
<td>Defenders</td>
<td>Hosts, Clusters, Cloud Account IDs</td>
</tr>
</tbody>
</table>

**Using Collections**

After collections are created or updated, there are some views that require a rescan before you can see the change:

- Deployed Images vulnerabilities and compliance views
- Registry Images vulnerabilities and compliance views
- Code repositories vulnerabilities view
- Trusted images
- Cloud Discovery
- Cloud Compliance
- Vulnerability Explorer
- Compliance Explorer

After collections are created or updated, there are some views that are affected by the change only for future records. These views include historical records that keep their collections from creation time:

- Images and Functions CI results view
- Events views
- Incidents view
Tags

Tags are predefined labels that can help you manage the vulnerabilities in your environment. They are centrally defined and can be set to vulnerabilities and as policy exceptions.

Tags are used as:

- Vulnerability labels. They provide a convenient way to categorize the vulnerabilities in your environment.
- Policy exceptions. They can be a part of your rules in order to have a specific effect on tagged vulnerabilities.

Tags are useful when you have large container deployments with multiple teams working in the same environment. For example, you might have different teams handling different types of vulnerabilities. Then you can set tags in order to define responsibilities over vulnerabilities. Other uses would be to set the status of fixing the vulnerability, or to mark vulnerabilities to ignore when they are a known problem that can’t be fixed in the near future.

Creating tags

You can create as many tags as you like.

STEP 1 | To create a tag, navigate to Manage > Collections and Tags > Tags.

Prisma Cloud ships with a predefined set of tags: Ignored, In progress, For review, DevOps notes. The predefined tags are editable and you can use them according to your needs.

STEP 2 | Click Add Tag.

STEP 3 | In the Create a new tag dialog, enter a name and description.

STEP 4 | Pick a color for easy visibility and differentiation.

STEP 5 | Click Save.
Logon settings

You can control how users access Prisma Cloud with logon settings.

Setting Console’s token validity period

Prisma Cloud lets you set up long-lived tokens for access to the Console web interface and the API. For security, users are redirected to the login page when an inactive Console session exceeds a configurable timeout. By default, the timeout is 30 minutes. This configurable timeout value also controls the validity period for API tokens.

For Console web interface tokens:
- If a user explicitly logs out, the claim to access Console is revoked.
- If Console is restarted, all users are automatically logged out.

STEP 1 | Open Console.

STEP 2 | Go to Manage > Authentication > Logon.

STEP 3 | Specify a value for Timeout for inactive Console sessions.

This value controls:
- Time, in minutes, that a Console session can be inactive. After the timeout expires, the user is redirected to the login page. In an active session, the token is automatically renewed when the time elapsed is greater than or equal to half the timeout value.
- Time, in minutes, that an API token is valid. After the token expires, a new one must be retrieved.

The maximum value permitted for Timeout for inactive Console sessions is 71580 minutes.

STEP 4 | Click Save.

After you save your changes, Console redirects you to the login page for your changes to take effect.

Single sign-on to the Prisma Cloud Support

Prisma Cloud can allow single sign on and contextual help from the "?” button in the upper right hand corner of each Console page.

Our https://docs.twistlock.com site allows access when a valid token is issued from the Customer. Or in this case, the "?” contextual links can embed the token into the URL used to access the page.

STEP 1 | Open Console.

STEP 2 | Go to Manage > Authentication > Logon.

STEP 3 | Set the toggle for Enable context sensitive help and single sign on to the Twistlock Support site.
When set to on (default), the token will be embedded into the contextual help link. When set to off, it will not be and you will need to enter the token manually.

STEP 4 | Click **Save**.

After saving your changes, Console redirects you to the login page for your changes to take effect.

**Basic authentication to Console and API**

Twistlock lets you disable basic authentication to the Console and API. Basic authentication is used in connections from *twistcli*, the API, and Jenkins.

With *twistcli*, you need to use the ‘--token’ option to authenticate with the Console for image scanning and other operations that access Console. This is the same token you receive from the /api/v1/authenticate API endpoint. For more information, see Accessing the API.

With the API, you would have to use the authenticate endpoint to generate an authentication token to access any of the endpoints. Accessing the API with Basic Authentication would not be allowed.

With Jenkins, there is no option at this point to use the Jenkins plugin and have basic authentication disabled. An option would be to use *twistcli* within Jenkins. This would require a step in the pipeline to retrieve an authentication token from the API for the scan to be completed.

STEP 1 | Open Console.

STEP 2 | Go to Manage > Authentication > Logon.

STEP 3 | Set the toggle for **Disable basic authentication to Console and API**.

When set to on, basic authentication will be disabled for the Console and API. You will not lose access to the Console from the login page. All of your user account will still be active and will still have access to login to the Console.

STEP 4 | Click **Save**.

After saving your changes, Console redirects you to the login page for your changes to take effect.

**Strict certificate validation in Defender**

Twistlock Console provides Defender installation scripts which use *curl* to transfer data from Console. By default, scripts copied from Console append the ‘-k’ option, also known as ‘--insecure’, to curl commands. This option lets curl proceed even if server connections are otherwise considered insecure.

Console provides a global option to disable the ‘-k’ argument for curl commands.

STEP 1 | Open Console.

STEP 2 | Go to Manage > Authentication > Logon.

STEP 3 | Set the toggle for **Require strict certificate validation in Defender installation links**.

When set to **On**, Defender installation scripts copied from Console do not use the ‘-k’ option with curl when transferring data from Console. In addition, the piped *sudo bash* command passes the ‘-V’ option to defender.sh to secure secondary curl commands in the defender.sh script.

STEP 4 | Click **Save**.

After saving your changes, Console redirects you to the login page for your changes to take effect.
Strong passwords for local accounts

Twistlock can enforce the use of a strong password. A strong password has the following requirements:

- Cannot be the same as the username.
- Must be at least 12 characters.
- Must contain one of each of the following: uppercase character, lowercase character, number, special character.
- List of special characters: `~!@#$%^&*()-_=+|[]{};:'",<.>/?`

**STEP 1 |** Open Console.

**STEP 2 |** Go to Manage > Authentication > Logon.

**STEP 3 |** Set the toggle for **Require strong passwords for local accounts**.

When enabled, strong passwords are required for passwords of newly created accounts or when existing passwords are changed. Enabling this setting doesn't force existing accounts to change their password or disable access to any accounts.

**STEP 4 |** Click **Save**.

After saving your changes, Console redirects you to the login page for your changes to take effect.
Reconfigure Prisma Cloud

In many cases, you will set up `twistlock.cfg` before you install Prisma Cloud. However, in some cases, you might want to change some parameters in `twistlock.cfg` after Prisma Cloud has already been installed. To reconfigure Prisma Cloud with an updated `twistlock.cfg`, run the `twistlock.sh` installer script again.

**STEP 1** | Extract the release tarball to a new location on your host.

Make sure this location does not have any previous Prisma Cloud install files.

```
$ tar -xvf twistlock_<VERSION>.tar.gz
```

**STEP 2** | Update `twistlock.cfg` with your new settings.

```
$ vim twistlock.cfg
```

**STEP 3** | Reload `twistlock.cfg`.

```
$ sudo ./twistlock.sh onebox
```

This command assumes that both `twistlock.sh` and `twistlock.cfg` reside in the same directory. To specify a configuration file in a different directory, use the `-c` option.

The old configuration is stored in `/var/lib/twistlock/scripts/twistlock.cfg.old`
Subject Alternative Names

You can add or remove Subject Alternative Names (SANs) to Console's certificate. The subjectAltName extension is described in RFC6125. It defines a mechanism for adding identities to Public Key Infrastructure X.509v3 (PKIX) certificates.

Defender communicates with Console using Transport Layer Security (TLS). When Defender tries to establish a secure connection with Console, it must validate Console's identity. Defender checks a reference identity against the identifiers presented in Console's PKIX certificate, searching for a match. The reference identity is set when you deploy Defender. It's the name you configured Defender to use to connect to Console.

When deploying Prisma Cloud Console, setting up a DNS name for it is considered a best practice. RFC6125 says:

"IP addresses are not necessarily reliable identifiers for application services because of the existence of private internets [PRIVATE], host mobility, multiple interfaces on a given host, Network Address Translators (NATs) resulting in different addresses for a host from different locations on the network, the practice of grouping many hosts together behind a single IP address, etc. Most fundamentally, most users find DNS domain names much easier to work with than IP addresses, which is why the domain name system was designed in the first place."

Adding a SAN to Console's certificate

Add a SAN to Console's certificate directly from Console's web interface.

**STEP 1** | Open Console.

**STEP 2** | Go to Manage > Defenders > Names.

**STEP 3** | Click Add SAN.

**STEP 4** | Enter a DNS name or IP address.

**STEP 5** | Click Add.
Credentials store

Container environments tend to utilize many third party services across multiple cloud providers. To improve accessibility and reusability, Prisma Cloud manages all credentials in a central encrypted store. Credentials are used when setting up the following integrations:

- Scanning (container registries, serverless functions, etc).
- Alerting in third party services (email, Slack, ServiceNow, etc).
- Deploying and managing Defender DaemonSets from the Console UI.
- Injecting secrets from secret stores into containers at runtime.

The credential store can be found under Manage > Authentication > Credentials Store. Credentials cannot be deleted if they are currently in use. To see all the places where a credentials is being used, click on an entry in the credentials store table, and review the Usages list.

If a credential is being used by an integration, and you edit its parameters (e.g. username, password, etc), the new values are automatically propagated to the right places in the product. You don’t need to delete and set up the integration again to refresh the credential’s values.

**Edit Credential**

<table>
<thead>
<tr>
<th>Name</th>
<th>AWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>AWS</td>
</tr>
<tr>
<td>Access Key</td>
<td>AKIAYP7JO7DYCDIKPA</td>
</tr>
<tr>
<td>Secret Key</td>
<td>Secret is stored in encrypted format, click here to replace</td>
</tr>
</tbody>
</table>

**Usages**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Scan</td>
<td>Used for scanning cloud provider account</td>
</tr>
<tr>
<td>Serverless Scan</td>
<td>Used for scanning serverless functions in aws-us-east-1</td>
</tr>
</tbody>
</table>
AWS

Prisma Cloud lets you authenticate with AWS three different ways:

- IAM users
- IAM roles
- Security Token Service (STS) (Recommended)

IAM users

IAM users are entities that represent persons or applications. The Prisma Cloud credentials store lets you save credentials for IAM users. When creating a new credential, select the AWS type, and enter an access key ID and secret access key.

Create A New Credential

<table>
<thead>
<tr>
<th>Name</th>
<th>Example AWS IAM user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>AWS</td>
</tr>
<tr>
<td>Access Key</td>
<td>AKIAI63E17UIDVK6OD4A</td>
</tr>
<tr>
<td>Secret Key</td>
<td>************************</td>
</tr>
</tbody>
</table>

IAM roles

IAM roles are identities, associated with permissions policies, that can be assumed by the resources that need them. Setting up Prisma Cloud to authenticate with IAM roles is handled in the subsystem-specific configuration dialogs.

For example, when you configure Prisma Cloud to scan an ECR repository, go to Defend > Vulnerabilities > Registry, and click Add registry. In the Version drop-down list, select Amazon EC2 Container Registry, and set Use IAM role to On. No other credentials need to be provided.
How Prisma Cloud accesses IAM role credentials

Roles provide a way to grant credentials to applications that run on EC2 instances to access other AWS services, such as ECR. IAM dynamically provides temporary credentials to the EC2 instances, and these credentials are automatically rotated for you.

When you create an EC2 instance, you can assign it a role. When the instance is started, the AWS instance metadata service (IMDS) attaches your credentials to the running EC2 instance. You can access this metadata from within the instance using the following command:

```
```

Where `<POLICY_NAME>` is assigned to the EC2 instance when it is created or at some point during its life.

The following diagram shows all the pieces. Defender retrieves the credentials from the metadata service, then uses those credentials to retrieve and scan the container images in ECR.
AWS Security Token Service (STS) (Recommended)

AWS Security Token Service (STS) lets you request temporary, limited-privilege credentials for AWS IAM users or users that you authenticate (federated users). Like IAM roles, setting up Prisma Cloud to use STS is handled in the subsystem-specific configuration dialogs, such as registry scanning and serverless function scanning.

This method is recommended as a best practice per the AWS Well-Architected Framework.

When you configure integration with an AWS resource, you can pick an AWS credential from the central store, then use STS to change the role of the account. AWS STS lets you have a few number of IAM identities that can be used across many AWS accounts. For example, if you were setting up Prisma Cloud to scan an AWS ECR registry, you would select the AWS credentials from the central store. Then you would enable Use AWS STS, and enter the name of the STS role to assume in the target account.

When using AWS STS, ensure the following:

- The policy of the IAM user you use as credentials has **sts:AssumeRole** permission on the IAM role you're going to assume. Sample policy:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "sts:AssumeRole",
         "Resource": "arn:aws:iam::123456789123:role/stsIAMrole"
      }
   ]
}
```
The IAM role you’re going to assume has the IAM user mentioned above configured as a trusted entity. Sample trusted entity policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789123:user/prismaUser"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Azure

This section discusses Azure credentials.

**Creating an Azure Service Principal**

Create an Azure Service Principal so that Prisma Cloud Console can scan your Azure tenant for microservices. To get a service key:

**STEP 1 | Download and install the Azure CLI.**

**STEP 2 | Create a service principal and configure its access to Azure resources.**

```
$ az ad sp create-for-rbac \
  --name <user>.twistlock-azure-cloud-discovery-<contributor|reader> \
  --role <reader|contributor> \
  --sdk-auth
```

The `--role` value depends upon the type of scanning:

- contributor = Cloud Discovery + Azure Container Registry Scanning + Azure Function Apps Scanning
- reader = Cloud Discovery + Azure Container Registry Scanning

**STEP 3 | Copy the output of the command and set it aside. It will be used as the Service Key when creating an Azure credential.**

```
{
  "clientId": "bc968c1e-67g3-4ba5-8d05-f807abb54a57",
  "clientSecret": "5ce0f4ec-5291-42f8-gbe3-90bb3f42ba14",
  "subscriptionId": "ae01981e-e1bf-49ec-ad81-80rf157a944e",
  "tenantId": "d189c61b-6c27-41d3-9749-ca5c9cc4a622",
  "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",
  "resourceManagerEndpointUrl": "https://management.azure.com/",
  "activeDirectoryGraphResourceId": "https://graph.windows.net/",
  "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",
  "galleryEndpointUrl": "https://gallery.azure.com/",
  "managementEndpointUrl": "https://management.core.windows.net/
```
Storing the credential in Prisma Cloud

Store the service principal's credentials in Console so that Prisma Cloud can authenticate with Azure for scanning.

**STEP 1** | Open Console, and go to Manage > Authentication > Credentials Store.

**STEP 2** | Click Add credential, and enter the following values:
1. In the Name field, enter a label to identify the credential.
2. In the Type field, select Azure.
3. In the Service Key field, enter the value returned by the Azure CLI tool when you created the service principal.
4. Click Save.

Google Cloud Platform (GCP)

Accessing GCP to scan resources can be done in one of two ways. You can make use of a service account and create an account for that purpose, or you can use an API Key. Google recommends that you use a service account with a key and we document that here. More information is available here https://cloud.google.com/docs/authentication/api-keys

Creating a service account

Create a service account that Prisma Cloud can use to scan your resources in GCP.

**STEP 1** | Google provide a comprehensive guide for creating a service account - https://cloud.google.com/iam/docs/creating-managing-service-accounts

**STEP 2** | Create a key for this service account. The format of this key should be JSON. Google have a guide for this - https://cloud.google.com/iam/docs/creating-managing-service-account-keys

**STEP 3** | Copy the contents of the downloaded key, here is an example:

```json
{
    "type": "service_account",
    "project_id": "mycompany-project",
    "private_key_id": "abe29475a09fb22e709f0262f6258a7b2e68dfc",
    "private_key": "-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBBgwgSkAgaEAoIBAQCyBjgPechqxAK
Tzal77Aqgei147IbgWeqRg8JqgoQGERhBX8X41otaRNUN7fpTdh/JjRfJ0wydzu
nn6TlmMez+d/yZBztzuJ4koGTS0TybtckWKe254upri6RicM354nxSntwlQx
vicVDC13uDKlNyawmLfi1HlWZK8bOueU57hd3J9UXc+B+lDL9RYYz1qX+n/Nz1w
7n3DTPFxxy54Hg39DrRum0kO+AIRMA5vPFmgrWlyN6HylOxCU/D5NufExRo3aug
isVqG3QLYU4a5n9jsWPbjSGI+E/H/+zZlfze5p6prl1vKtbyvgJSFGpdsjs4EX
nbfqPYjajMBAAEcggEAU9fIayLYNIKIYrwsnpUQhkk/
OwSbhQf61VfqaAavnF0hFX4GGLpl2u6bmP1zoFoutWIhatagpBG91AU/
fi/cNi+iK0rZW0Muj8CQ0ttg/nQbQpZbP4Daxsg72NVH2hKjgK10VbW/
xSIMZswXwXnxqBF17qaN4Q5RFBsdqf+mpM4/\nJ23ZIKW5muF+5Svz4wLLS7Vmb1l/\nXr9eCepVQnZq70167VQa1Z8K1ot51eQ0d2\jnHJhN6XgAe/IKuP8CIxNwuo/
GbChCtctPr0BpTeatsOS0rلل7ntqFBGYx1SmgZ0/n13x43KuYacy2EycKEoaa9U
+k1KcXzF/CDQvd+AUQKBoQD25mVWFXQdHfQf1z2u/\nNeH20yrj3LbRe
+V3uQEyOJboKMfM3iH+N9Q9neF1n8qrHgO7KHEgjLav4ZGeC1n
\nxlNtMNAFPGRREGuozvzQQwBQgAzB0vI2gkITmcsQ7xzcGy1Xlm1xhpkqkoWFe5c
\nk253FyHmjuTTXisYv8Lb15xGQKQbQc41ER2AmTSvLe+4su1TeEOcMsp+G4j/
A\ne1Sn1m5e5yGUtudFp1nuY5uX3ERZvEDCR07B/osQ4ueAe1S1S3Zwv5QVt/S/
s1nFOJaiUJ/\nxKAGAaV6cAp1RgKJlYrbf/yoxeR1CQkQJcRd1S09D1CSwdSW1CpipaUN
\nfZzzW3iD78QKBqCWW7Lk3cMjQhH6Fjm1ylYRDIYHA1MkUf1Fga0Cuc7EDtyYicF
```
Storing the credential in Prisma Cloud

Store your GCP credential in Prisma Cloud.

STEP 1 | Open Console, and go to Manage > Authentication > Credentials Store.

STEP 2 | Click Add credential, and enter the following values:

1. In the Name field, enter a label to identify the credential.
2. In the Type field, select GCP.
3. In the Service Account field, copy and paste the entire JSON key that you downloaded.
4. Leave the API token blank
5. Click Save.

IBM Cloud

Prisma Cloud integrates with IBM Cloud Security Advisor. To enable the integration, you must provide credentials, which consist of an Account GUID and API Key.
Kubernetes stores cluster authentication information in a YAML file known as kubeconfig. The kubeconfig file grants access to clients, such as kubectl, to run commands against the cluster. By default, kubeconfig is stored in $HOME/.kube/config.

Prisma Cloud uses the kubeconfig credential to deploy and upgrade Defender DaemonSets directly from the Console UI. If you plan to manage DaemonSets from the command line with kubectl, you don’t need to create this credential type.

The user or service account in your kubeconfig must have permissions to create and delete the following resources:

- ClusterRole
- ClusterRoleBinding
- DaemonSet
- Secret
- ServiceAccount

Prisma Cloud doesn’t currently support kubeconfig credentials for Google Kubernetes Engine (GKE) or AWS Elastic Kubernetes Service (EKS) clusters. The kubeconfig for these requires an external binary (specifically the Google Cloud SDK and aws-iam-authenticator, respectively) for authentication, and the Prisma Cloud Console doesn’t ship with this binary.

STEP 1 | Open Console, and go to Manage > Authentication > Credentials Store.

STEP 2 | Click Add credential, and enter the following values:

1. In Name, enter a label to identify the credential.
2. In Type, select Kubeconfig.
3. In Kubeconfig, paste the contents of your kubeconfig file.
Authentication

Prisma Cloud provides broad enterprise identity support. It can integrate with a number of identity providers, including Active Directory, OpenLDAP, Ping, Okta, Shibboleth, Azure AD, and Google G Suite, so you can implement single sign-on for the Prisma Cloud Console. Prisma Cloud supports simultaneous integration with multiple identity providers. For example, you might want to integrate with both Okta and GitHub to support users that login from both platforms.

Prisma Cloud ships with prebuilt roles to provide least privilege access to your devops and security teams. Use assigned collections to precisely control what data teams can view or use built-in multi-tenancy to securely isolate entire business units or geographies within the same Console.

Pluggable cryptography lets you bring your own certificates, not just for TLS, but also for smart card authentication to Console.

A credentials store provides a single secure location to store service accounts for integration with the various cloud providers. Define them once in the credentials store, and then reuse them throughout Console for your various integrations.

- Logging into Prisma Cloud
- Integrate with Active Directory
- Integrate with OpenLDAP
- Integrate Prisma Cloud with Open ID Connect
- Integrate with Okta via SAML 2.0 federation
- Integrate Google G Suite via SAML 2.0 federation
- Integrate with Azure Active Directory via SAML 2.0 federation
- Integrate with PingFederate via SAML 2.0 federation
- Integrate with Windows Server 2016 & 2012r2 Active Directory Federation Services (ADFS) via SAML 2.0 federation
- Integrate Prisma Cloud with GitHub
- Integrate Prisma Cloud with OpenShift
- Non-default UPN suffixes
- Compute user roles
- Assign roles
- Use custom certificates for authorization
- Credentials store
Logging into Prisma Cloud

Prisma Cloud Console supports multiple authentication methods. Check with your administrator to see how sign-in has been implemented for your organization, then choose the appropriate method from the drop-down list.

The options are:

- **Local/ LDAP** – Users are evaluated against Console's database before the LDAP database. By default, initial admin users are created in Console's local database, so choose this option when you're logging in with your first user. If you integrate with a central identity provider, you can always delete the initial admin user, so that all users authenticate in compliance with your organization’s policy (e.g., 2FA).
  
  If the same username exists in both databases, it's not possible to login with the LDAP user.

- **SAML** – Security Assertion Markup Language (SAML) is an open standard that enables single sign-on. Prisma Cloud supports all standard SAML 2.0 providers.

- **OAuth** – Prisma Cloud currently supports GitHub and OpenShift for OAuth login. For the OAuth login flow, Prisma Cloud gets permission from the user to query their information (username and email) from GitHub or OpenShift, and then checks the local database to determine if the user is authorized to access Prisma Cloud Console. If so, Prisma Cloud issues a token to the user to access Console.

- **OpenID Connect** – OpenID Connect is a simple identity layer on top of the OAuth 2.0 protocol. Prisma Cloud supports all standard OpenID Connect providers.

Login flow

If you integrate Prisma Cloud with an identity provider (IdP), the user’s identity is verified by the IdP, and the role is mapped in Prisma Cloud Console.

If you don’t want to integrate with an IdP, Prisma Cloud lets you create "local" users and groups, where the Console itself both authenticates and authorizes users.
Direct login URL

Direct login URLs are supported for SAML, OAuth and OIDC. When you use the direct login URL, the client doesn’t need the extra step of selecting an auth provider from the Prisma Cloud login page.

Set type in the direct login URL:

Integrate with Active Directory

Prisma Cloud can integrate with Active Directory (AD), an enterprise identity directory service.

*If your AD environment uses alternative UPN suffixes (also referred to as explicit UPNs), see Non-default UPN suffixes to understand how to use them with Prisma Cloud.*

*LDAP group names are case sensitive in Prisma Cloud.*

With AD integration, you can reuse the identities and groups centrally defined in Active Directory, and extend your organization’s access control policy to manage the data users can see and the things they can do in the Prisma Cloud Console.

For more information about Prisma Cloud’s built-in roles, see User Roles.

**Configuration options**

The following configuration options are available:

<table>
<thead>
<tr>
<th>Configuration option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Enables or disables integration with Active Directory. In Console, use the slider to enable (ON) or disable (OFF) integration with AD. By default, integration with AD is disabled.</td>
</tr>
<tr>
<td>URL</td>
<td>Specifies the path to your LDAP server, such as an Active Directory Domain Controller. The format for the LDAP server path is: <code>&lt;PROTOCOL&gt;://&lt;HOST&gt;:&lt;PORT&gt;</code> Where <code>&lt;PROTOCOL&gt;</code> can be ldap or ldaps. For an Active Directory Global Catalog server, use ldap. For performance and redundancy, use a load balanced path. Example: ldap://ldapserver.example.com:3268</td>
</tr>
<tr>
<td>Search Base</td>
<td>Specifies the search query base path for retrieving users from the directory. Example: dc=example,dc=com</td>
</tr>
<tr>
<td>User identifier</td>
<td>User name format when authenticating: sAMAccountName = DOMAIN\sAMAccountName userPrincipalName = <a href="mailto:user@ad.example.com">user@ad.example.com</a> The Active Directory domain name must be provided when using sAMAccountName due to domain trust behavior.</td>
</tr>
<tr>
<td>Account UPN</td>
<td>Console Account UPN Specifies the username for the Prisma Cloud service account that has been set up to query Active Directory. Specify the username with the User Principal Name (UPN) format:</td>
</tr>
</tbody>
</table>
### Configuration option

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
</table>
| `<USERNAME>@<DOMAIN>`
Example: `twistlock_service@example.com` |

<table>
<thead>
<tr>
<th>Account Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the password for the Prisma Cloud service account.</td>
</tr>
</tbody>
</table>

### Integrating Active Directory

Integrate Active Directory after you have installed Prisma Cloud.

**STEP 1** | Open Console, then go to Manage > Authentication > Identity Providers.

**STEP 2** | Set Integrate LDAP users and groups with Prisma Cloud to Enabled.

**STEP 3** | Specify all the parameters for connecting to your Active Directory service.

1. For Authentication type, select Active Directory.
2. In Path to LDAP service, specify the path to your LDAP server.
   
   For example: `ldap://ldaps://ldapserver.example.com:3268`
3. In Search Base, specify the base path to the subtree that contains your users.
   
   For example: `dc=example,dc=com`
4. In Service Account UPN and Service Account Password, specify the credentials for your service account.
   
   Specify the username in UPN format: `<USERNAME>@<DOMAIN>`
   
   For example, the account UPN format would be: `twistlock_service@example.com`
5. If you connect to Active Directory with ldaps, paste your CA certificate (PEM format) in the CA Certificate field.
   
   This enables Prisma Cloud to validate the LDAPS certificate to prevent spoofing and man-in-the-middle attacks. If this field is left blank, Prisma Cloud will not perform validation of the LDAPS certificate.

**STEP 4** | Click Save.

### Adding Active Directory group to Prisma Cloud

To grant authentication to users in an Active Directory group, add the AD group to Prisma Cloud.

**STEP 1** | Navigate to Manage > Authentication > Groups and click Add group.

**STEP 2** | In the dialog, enter AD group name and select LDAP group.
STEP 3 | Grant a role to members of the group.

Verifying integration with Active Directory

Verify the integration with AD.

STEP 1 | Open Console.

STEP 2 | If you are logged into Console, log out.

STEP 3 | At Console's login page, enter the UPN and password of an existing Active Directory user.

If the log in is successful, you are directed to the view appropriate for the user's role. If you have the Access User role, you are directed to a single page, where you can download certs for Docker client role-based access control.
Integrate with OpenLDAP

Prisma Cloud can integrate with OpenLDAP, an open source implementation of the Lightweight Directory Access Protocol.

Integrating Prisma Cloud with OpenLDAP lets users access Prisma Cloud using their LDAP credentials, and lets admins define granular access control rules to Docker Engine, Docker Swarm, and Kubernetes using existing LDAP identities.

With OpenLDAP integration, you can:
- Re-use the identities and groups already set up in your OpenLDAP directory.
- Extend your organization’s access control logic to the management of Docker containers.

For example, you could specify that only members of the group Dev Ops Admins can start and stop containers in the production environment. For more information, see the article for setting up role-based access control for Docker Engine.

Integrating OpenLDAP

This procedure shows you how to integrate OpenLDAP with Prisma Cloud.

**Prerequisites:**
- You have installed OpenLDAP 2.4.44 or later. Prisma Cloud has been tested with version 2.4.44. Integration with older versions should work as well, but isn’t officially supported.

**STEP 1**

In your LDAP directory, create a service account that has admin privileges and that can run ldapsearch queries.

This admin account will be used by Prisma Cloud to authenticate users in your LDAP directory. It should be able to control the entire domain, and should therefore be created under the root OU.

**STEP 2**

Verify that the service account can query your LDAP directory.

Run ldapsearch, passing it the credentials for your service account, and query your directory for a user:

```
$ ldapsearch -x \
-b dc=example,dc=com \n-D "cn=<SA-CN>,dc=example,dc=com" \n-w <SA-PASS> 
"(cn=<some-user-cn>)"
```

Where:
- `<SA-CN>` -- Common name for the Prisma Cloud service account.
- `<SA-PASS>` -- Password for the Prisma Cloud service account.
- `<some-user-cn>` -- Common name for a user in your LDAP directory.

**STEP 3**

Open Console, and go to Manage > Authentication > Identity Providers > LDAP.

**STEP 4**

Set Integrate LDAP users and groups with Prisma Cloud to Enabled.
STEP 5 | For Authentication type, select OpenLDAP.

STEP 6 | For Path to LDAP service, enter the LDAP server and port number in the following format:
For secure connections over TLS: ldaps://<server-dns>[:port-number>.
For insecure connections: ldap://<server-dns>[:port-number>.

STEP 7 | For Search base, enter the base DN for your users and groups.

STEP 8 | (OPTIONAL) For User identifier, specify an attribute to be used to match users.
For example, enter uid to match users based on their user IDs.

STEP 9 | For Service account UPN, enter the DN for your Prisma Cloud service account.

STEP 10 | For Service account password, enter the password for the Prisma Cloud service account.

STEP 11 | For CA certificate, provide the CA certificate used to sign the LDAPS certificate on the server.
Prisma Cloud uses the CA certificate to validate the LDAPS certificate and prevent man-in-the-middle attacks. If you are using an insecure connection or do not wish to validate the LDAPS certificate, leave this field blank.

STEP 12 | Click Save.
Console verifies all your parameters with the server. If a connection cannot be established, an error message is shown and no parameters are saved.

Verifying integration with OpenLDAP

Verify the integration with OpenLDAP.

STEP 1 | Open Console.

STEP 2 | If you are logged into Console, log out.

STEP 3 | Log in to Console using the credentials of an existing OpenLDAP user.
If the log in is successful, you are directed to the view appropriate for the user's role. If you have the Access User role, you are directed to a single page, where you can download certs for Docker client role-based access control.
Integrate Prisma Cloud with Open ID Connect

OpenID Connect is an standard that extends OAuth 2.0 to add an identity layer. Prisma Cloud supports integration with any standard Open ID Connect (OIDC) provider that implements both OpenID connect core and OpenID connect discovery. Instructions for integrating with PingOne and Okta are shown here. Prisma Cloud supports the authorization code flow only.

PingOne

Integrate with PingOne.

You need to configure Compute as an OIDC app. When configuring your app:

- The Start SSO URL must point to https://<CONSOLE>:<PORT>/callback.
- The Redirect URI must point to https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oidc.
- UserInfo must include sub, idpid, name.
- All of the following scopes must be included for OpenID: OpenID Connect (openid), OpenID profile, OpenID Email, OpenID address, OpenID Phone, Groups.

Update Ping callback URL

Update the callback URL.

STEP 1 | Log into the Ping web portal.

STEP 2 | Click Applications, and then click the OIDC tab.

STEP 3 | Click on the arrow button nest for your app.

STEP 4 | Click on the pencil icon on the right side.

STEP 5 | Click on Authentication Flow.

STEP 6 | In REDIRECT URIS, enter the callback URL:


Create new user and join to group

STEP 1 | In the Ping web portal, click Users, and then click the Users tab.

STEP 2 | Click Add users, and choose the Create New User option.

STEP 3 | Fill the fields for Password, Username (should be your email), First Name, Last Name, and Email.

STEP 4 | In the Membership field, click Add, and choose a group.

STEP 5 | Click Save.

Okta

Integrate with Okta.
- Initiate Login URI (Okta) must point to https://<CONSOLE>:<PORT>/callback.
- Redirect URI must point to https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oidc.
- UserInfo must include sub, idpid, name.
- Scopes:
  - All of the following scopes must be included for OpenID: OpenID Connect (openid), OpenID profile, OpenID Email, OpenID address, OpenID Phone, Groups.
  - All of the following scopes must be included for Okta: okta.groups.manage, okta.groups.read.

**Update Okta callback URL**

Update the callback URL.

**STEP 1** | Log into Okta.

**STEP 2** | Click on Applications and click on your application.

**STEP 3** | Click the General tab, and then click Edit.

**STEP 4** | Update Login redirect URIs. Enter the following callback URL:
  - https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oidc, and then click Save.

**Configure Prisma Cloud**

Configure Prisma Cloud.

**STEP 1** | Log into Prisma Cloud Console.

**STEP 2** | Go to Manage > Authentication > Identity Providers > OpenID Connect.

**STEP 3** | Enable OpenID Connect.

**STEP 4** | Fill in the settings.
  1. For **Client ID**, enter the client ID.
  2. For **Client Secret**, enter the client secret.
  3. For **Issuer URL**, enter:
  4. For **Group scope**, select groups.
  5. (Optional) Enter your certificate.
  6. Click **Save**.

**Prisma Cloud to OIDC user identity mapping**

Create a Prisma Cloud user for every user that should have access to Prisma Cloud. Prisma Cloud uses the sub attribute that comes from OIDC to match the username configured in the Prisma Cloud database (as required by the OIDC spec). Whichever value the provider is configured to send to Prisma Cloud should be used to configure users.

**STEP 1** | Go to Manage > Authentication > Users.

**STEP 2** | Click Add User.

**STEP 3** | Set **Username** to the GitHub user name.
STEP 4 | Set Auth method to OpenID Connect.

STEP 5 | Select a role for the user.

STEP 6 | Click Save.

STEP 7 | Test logging into Prisma Cloud Console.
   1. Logout of Prisma Cloud.
   2. On the login page, select OpenID Connect, and then click Login.
   3. You’re redirected to your OIDC provider to authenticate.
   4. After successfully authenticating, you’re logged into Prisma Cloud Console.

Prisma Cloud to OIDC provider group mapping

When you use groups to assign roles in Prisma Cloud you don’t have to create individual Prisma Cloud accounts for each user. The group value configured on the Compute side should reflect the name of the group scope in the OIDC provider. It might be something different than groups.

Groups can be associated and authenticated with by multiple identity providers.

STEP 1 | Go to Manage > Authentication > Groups.

STEP 2 | Click Add Group.

STEP 3 | In Name, enter an OpenShift group name.

STEP 4 | In Authentication method, select External Providers.

STEP 5 | In Authentication Providers, select OpenID Connect group.

STEP 6 | Select a role for the members of the group.

STEP 7 | Click Save.

STEP 8 | Test logging into Prisma Cloud Console.
   1. Logout of Prisma Cloud.
   2. On the login page, select OpenID Connect, and then click Login.
3. You're redirected to your OIDC provider to authenticate.
4. After successfully authenticating, you're logged into Prisma Cloud Console.
Integrate with Okta via SAML 2.0 federation

Many organizations use SAML to authenticate users for web services. Prisma Cloud supports the SAML 2.0 federation protocol to access the Prisma Cloud Console. When SAML support is enabled, administrators can log into Console with their federated credentials. This article provides detailed steps for federating your Prisma Cloud Console with Okta.

The Prisma Cloud/Okta SAML federation flow works as follows:

1. Users browse to Prisma Cloud Console.
2. Their browsers are redirected to the Okta SAML 2.0 endpoint.
3. They enter their credentials to authenticate. Multi-factor authentication can be enforced at this step.
4. A SAML token is returned to Prisma Cloud Console.
5. Prisma Cloud Console validates the SAML token’s signature and associates the user to their Prisma Cloud account via user identity mapping or group membership.

Integrating Prisma Cloud with SAML consists of setting up your IdP, then configuring Prisma Cloud to integrate with it.

Setting up Prisma Cloud in Okta

Set up Prisma Cloud in Okta.

**STEP 1** | Log into the Okta admin dashboard.

**STEP 2** | On the right, click **Add Applications**.

**STEP 3** | On the left, click **Create new app**.
STEP 4 | Select SAML 2.0, and then click Create.

STEP 5 | In the App name field, enter Prisma Cloud Console, then click Next.
STEP 6 | In the SAML Settings dialog:

1. In the Single Sign On URL field, enter https://<CONSOLE_ADDR>:8083/api/v1/authenticate.

   Note that if you've changed the default port you use for the HTTPS listener, you'd need to adjust the URL here accordingly. Additionally, this URL must be visible from the Okta environment, so if you're in a virtual network or behind a load balancer, it must be configured to forward traffic to this port and its address is what should be used here.

2. Select Use this for Recipient URL and Destination URL.

3. In the field for Audience Restriction, enter twistlock (all lowercase).

4. Expand Advanced Settings.

5. Verify that Response is set to Signed.

6. Verify that Assertion Signature is set to Signed.
### STEP 7 | (Optional) Add a group.

Setting up groups is optional. If you set up group attribute statements, then permission to access Prisma Cloud is assessed at the group level. If you don't set up group attribute statements, permission to access Prisma Cloud is assessed at the user level.

1. Scroll down to the **GROUP ATTRIBUTE STATEMENTS** section.
2. In the **Name** field, enter **groups**.
3. In filter drop down menu, select **Regex** and enter a regular expression that captures all the groups defined in Okta that you want to use for access control rules in Prisma Cloud.

In this example, the regular expression `.*(t|T)wistlock.*` is used to include all groups prepended with either Prisma Cloud or twistlock. You should enter your own desired group name here. If you have just one group, such as **YourGroup**, then just enter **YourGroup**. Regular expressions are not required. If you have multiple groups, you can use a regular expressions, such as `(group1|group2|group3)`. 

![SAML Settings](image-url)
STEP 8 | Click Next, and then click Finish.

You are directed to a summary page for your new app.

STEP 9 | Click on the People tab, and add users to the Prima Cloud app.
STEP 10 | Click on the Groups tab, and add groups to the Prisma Cloud app.

STEP 11 | Click on the Sign On tab and click View setup instructions.

The following values are used to configure Prisma Cloud Console, so copy them and set them aside.

- Identity Provider Single Sign-On URL
- Identity Provider Issuer
- X.509 Certificate
Configuring Console

Configure Prisma Cloud Console.

STEP 1 | Open Console, and login as admin.

STEP 2 | Go to Manage > Authentication > Identity Providers > SAML.

STEP 3 | Set Integrate SAML users and groups with Prisma Cloud to Enabled.

STEP 4 | Set Identity provider to Okta.

STEP 5 | Copy the following values from Okta and paste them into their corresponding fields in Console:

- Identity Provider Single Sign-On URL
- Identity Provider Issuer
- X.509 Certificate

STEP 6 | In Audience, enter twistlock.

STEP 7 | Click Save.

Granting access by group

Grant access to Prisma Cloud Console by group. Each group must be assigned a role. You can optionally use these groups to define RBAC rules for controlling who can run which Docker Engine commands in your environment.
STEP 1 | Open Console.

STEP 2 | Define a SAML group.
   1. Go to Manage > Authentication > Groups.
   2. Click Add group.
   3. In the Name field, enter a group name.
      The group name must exactly match the group name in the SAML IdP. Console does not verify if that 
      the value entered matches a group name in the SAML IdP.
   4. Select the SAML group checkbox.
   5. Select a role.
   6. Select a project(s) - Optional.
   7. Click Save.

Granting access by user

Grant access to Prisma Cloud Console by user. Each user must be assigned a role. You can optionally use 
these user to define RBAC rules for controlling who can run which Docker Engine commands in your 
environment.

STEP 1 | Open Console.

STEP 2 | Define a SAML user.
   1. Go to Manage > Authentication > Users.
   2. Click Add user.
   3. In the Username field, enter a user name.
      The username must exactly match the username in the SAML IdP. Console does not verify if that the 
      value entered matches a user name in the SAML IdP.
   4. Select SAML as the Auth method
   5. Select a role.
   6. (Optional) Select a project(s).
   7. Click Save.
Integrate Google G Suite via SAML 2.0 federation

Many organizations use SAML to authenticate users for web services. Prisma Cloud supports the SAML 2.0 federation protocol to access the Prisma Cloud Console. When SAML support is enabled, users can log into Console with their federated credentials. This article provides detailed steps for federating your Prisma Cloud Console with Google G Suite.

The Prisma Cloud/G Suite SAML federation flow works as follows:
1. Users browse to Prisma Cloud Console.
2. Their browsers are redirected to the G Suite SAML 2.0 endpoint.
3. They enter their credentials to authenticate. Multi-factor authentication can be enforced at this step.
4. A SAML token is returned to Prisma Cloud Console.
5. Prisma Cloud Console validates the SAML token’s signature and associates the user to their Prisma Cloud account via user identity mapping or group membership.

Setting up Google G Suite

Prisma Cloud supports SAML integration with Google G Suite.

**STEP 1 |** Log into your G Suite admin console.

**STEP 2 |** Click on Apps.

**STEP 3 |** Click on SAML apps.

**STEP 4 |** Click the + button at the bottom to add a new app.

**STEP 5 |** Click SETUP MY OWN CUSTOM APP at the bottom of the dialog.
STEP 6 | Copy the SSO URL and Entity ID, and download the certificate. You will need these later for setting up the integration in Prisma Cloud Console. Click NEXT.

STEP 7 | Enter an Application Name, such as Prisma Cloud, then click NEXT.

STEP 8 | In the Service Provider Details dialog, enter the following details, then click NEXT.

2. In Entity ID, enter: `twistlock`.
3. Enable Signed Response.
STEP 9 | Click FINISH, then OK.

STEP 10 | Turn the application to on. Select either ON for everyone or ON for some organizations.
Setting up Prisma Cloud

Set up Prisma Cloud for G Suite integration.

**STEP 1** | Log into Console, then go to Manage > Authentication > Identity Providers > SAML.

**STEP 2** | Set Integrate SAML users and groups with Prisma Cloud to Enabled.

**STEP 3** | Set Identity provider to G Suite.

**STEP 4** | Set up the following parameters:
1. Paste the SSO URL, Entity ID, and certificate that you copied during the G Suite set up into the Identity Provider single sign-on URL, Identity provider issuer, and X.509 certificate fields.
2. Set Audience to match the application Entity ID configured in G Suite. Enter twistlock.
3. Click Save.

**STEP 5** | Go to Manage > Authentication > Users, and click Add user.

**STEP 6** | In the Username field, enter the G Suite email address the user you want to add. Select a role, then click Save. Be sure Create user in local Prisma Cloud account database is Off.

**STEP 7** | Log out of Console.

You will be redirected into G Suite and you might need to enter your credentials. After that, you will be redirected back into Prisma Cloud and authenticated as a user.
Integrate with Azure Active Directory via SAML 2.0 federation

Many organizations use SAML to authenticate users for web services. Prisma Cloud supports the SAML 2.0 federation protocol to access Prisma Cloud Console. When SAML support is enabled, users can log into Console with their federated credentials. This article provides detailed steps for federating your Prisma Cloud Console with your Azure Active Directory (AAD) tenant’s Identity Provider (IdP).

The Prisma Cloud/Azure Active Directory SAML federation flow works as follows:

1. Users browse to Prisma Cloud Console.
2. Their browsers are redirected to the AAD SAML 2.0 endpoint.
3. They enter their AAD credentials to authenticate. Multi-factor authentication can be enforced at this step.
4. An AAD SAML token is returned to Prisma Cloud Console.
5. Prisma Cloud Console validates the Azure Active Directory SAML token's signature and associates the user to their Prisma Cloud account via user identity mapping or group membership. Prisma Cloud supports SAML groups for Azure Active Directory federation.

The Azure Portal may change the Enterprise Application SAML federation workflow over time. The concepts and steps outlined in this document can be applied to any Non-gallery application.

The Prisma Cloud Console is integrated with Azure Active Directory as a federated SAML Enterprise Application. The steps to set up the integration are:

- Configure Azure Active Directory
  - Prisma Cloud User to AAD User Identity mapping
  - Prisma Cloud Groups to AAD Group mapping
  - Configure Prisma Cloud Console

Configure Azure Active Directory

Configure Azure Active Directory.

Prerequisites:

- Required Azure Active Directory SKU: Premium
- Required Azure Active Directory role: Global Administrator

STEP 1 | Log onto your Azure Active Directory tenant (https://portal.azure.com).

STEP 2 | Go to Azure Active Directory > Enterprise Applications

STEP 3 | On the top left of the window pane, click + New Application.

STEP 4 | Select Non-gallery application, from the Add your own app section.
STEP 5 | In the **Name** field, enter `jdong-console`, then click **Add**. In this example I am using "jdong-console"

STEP 6 | On the `jdong-console` menu select **Single sign-on** and choose **SAML**

STEP 7 | **Section #1 Basic SAML Configuration:**

1. **Identifier**: `jdong-console` (Set to your Console’s unique Audience value. You will configure this value within your Console at a later step.)

**Basic SAML Configuration**

**Identifier (Entity ID)***

*The default identifier will be the audience of the SAML response for IDP-initiated SSO*

**Reply URL (Assertion Consumer Service URL)***

*The default reply URL will be the destination in the SAML response for IDP-initiated SSO*

---

**STEP 8** | **Section #2 User Attributes & Claims:**

Select the Azure AD user attribute that will be used as the user account name within Prisma Cloud. This will be the NameID claim within the SAML response token. We recommend using the default value.

1. **Unique User Identifier (Name ID):** user.userprincipalname [nameid-format:emailAddress]
### User Attributes & Claims

<table>
<thead>
<tr>
<th>Claim name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique User Identifier (Name ID)</td>
<td>user.userprincipalname [nameid-f... ***</td>
</tr>
</tbody>
</table>

### Additional claims

<table>
<thead>
<tr>
<th>Claim name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress">http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress</a></td>
<td>user.mail ***</td>
</tr>
<tr>
<td><a href="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname">http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname</a></td>
<td>user.givenname ***</td>
</tr>
<tr>
<td><a href="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name">http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name</a></td>
<td>user.userprincipalname ***</td>
</tr>
<tr>
<td><a href="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname">http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname</a></td>
<td>user.surname ***</td>
</tr>
</tbody>
</table>

Even if you are using AAD Groups to assign access to Prisma Cloud set this value.

**STEP 9 | Section #3 SAML Signing Certificate:**

1. Select **Download: Certificate (Base64)**
2. Select the Pen icon.
3. Set **Signing Option**: Sign SAML Response and Assertion
STEP 10 | Section #4 **Set up jdong_console:**

Save the value of Login URL and Azure AD Identifier. We will use these later for configuration in the Prisma Cloud Console.

**Set up jdong-console**

You'll need to configure the application to link with Azure AD.

- **Login URL**: https://login.microsoftonline.com/147769cd-2e7f-43a7-b8...
- **Azure AD Identifier**: https://sts.windows.net/147769cd-2e7f-43a7-b8...
- **Logout URL**: https://login.microsoftonline.com/common/wsf...

View step-by-step instructions

STEP 11 | Copy the **Application ID**. You can find this going to Properties tab in the Manage section of the application.

STEP 12 | Click on **Users and Groups** within the Manage section of the application. Add the users and/or groups that will have the right to authenticate to Prisma Cloud Console.
Prisma Cloud User to AAD User Identity mapping

If you plan to map Azure Active Directory users to Prisma Cloud accounts go to Configure Prisma Cloud Console.

Prisma Cloud Groups to AAD Group mapping

When you use Azure Active Directory Groups to map to Prisma Cloud SAML Group, do not create users in Prisma Cloud Console. Configure the AAD SAML application to send AAD group membership (http://schemas.microsoft.com/ws/2008/06/identity/claims/groups) claims within the SAML response token. If you enable AAD Group authentication the Prisma Cloud User to AAD User Identity method of authentication will be ignored.

STEP 1 | Set Application permissions:

1. In Azure go to Azure Active Directory > Application Registrations > jdong-console
2. Under the Manage section, go to API Permissions
3. Click on Add a Permission
4. Click on Microsoft Graph
5. Select permissions: Application Permissions: Application.Read.All
6. Click **Add Permissions**

7. Click **Grant admin consent for Default Directory** within the Configured permissions blade.

**STEP 2 | Create Application Secret**

1. Under the Manage section, go to *Certificates & secrets*
2. Click on **New Client secret**
3. Add a *secret description*
4. *Expires: Never*
5. Click **Add**
6. Make sure to save the secret *value* that is generated before closing the blade.
Client secrets
A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

<table>
<thead>
<tr>
<th>Description</th>
<th>Expires</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>jdong-console</td>
<td>12/31/2299</td>
<td>$i***************</td>
</tr>
</tbody>
</table>

**STEP 3** Configure the application to send group claims within the SAML response token.

You can configure this setting either within the Azure portal or via powershell.

1. Azure AD Portal:
   1. Go to Azure Active Directory > App registrations > jdong-console
   2. Click Manifest
   3. Set "groupMembershipClaims": "SecurityGroup"
   4. Click Save
2. Powershell:

1. Use the Azure AD powershell commandlet `Set-AzureADApplication` to configure the application.
2. Run the following powershell commands:

```powershell
import-module AzureAD
Connect-AzureAD
$twistlock = Get-AzureADApplication | where-object {$__DISPLAYNAME -eq "jdong-console"}
$oid = $twistlock.ObjectId
Set-AzureADApplication -ObjectId $oid -GroupMembershipClaims 1
```

3. Confirm that the `GroupMembershipClaims` has been set to `SecurityGroup`

```powershell
$twistlock = Get-AzureADApplication | where-object {$__DISPLAYNAME -eq "jdong-console"}
$twistlock.GroupMembershipClaims
```

Allow several minutes for these permissions to propagate within AAD.
Configure Prisma Cloud Console

Configure Prisma Cloud Console.

**STEP 1** | Log into Prisma Cloud Console as an administrator.

**STEP 2** | Go to Manage > Authentication > Identity Providers > SAML.

**STEP 3** | Set Integrate SAML users and groups with Prisma Cloud to Enabled.

1. Set Identity Provider to Azure.
2. In Identity provider single sign-on URL, enter the Azure AD provided Login URL.
3. In Identity provider issuer, enter the Azure AD provided Azure AD Identifier.
4. In Audience, enter jdong-console.
5. In Application ID, enter jdong-console’s AAD Application ID.
6. In Tenant ID, enter AAD tenant ID that contains the jdong-console application.
7. In Application Secret enter jdong-console application keys (only required if using AAD Groups).
8. In X.509 certificate, paste the Azure AD SAML Signing Certificate Base64 into this field.
**SAML settings**

Integrate SAML users and groups with Prisma Cloud

<table>
<thead>
<tr>
<th>Identity provider</th>
<th>ADFS</th>
<th>Azure</th>
<th>G Suite</th>
<th>Okta</th>
<th>Ping</th>
<th>Shibboleth</th>
</tr>
</thead>
</table>

**Identity provider single sign-on URL**

https://login.microsoftonline.com/\[Blank\]/saml2

**Identity provider issuer**

https://sts.windows.net/f\[Blank\]/

**Audience**

jdong-console

**Console URL**

Optional. Used by the IDP for routing the browser after login. e.g., https://<console-IP>:

**Application ID**

\[Blank\]

**Tenant ID**

\[Blank\]

**Application Secret**

*Secret is stored in encrypted format, click here to replace*

**X.509 certificate**

-----BEGIN CERTIFICATE-----
MIIC8DCCAdgAwIBAgIBATUDBgkqhkiG9w0BAQsFADAOMTlwMAYD
VQQD
EylNaWNyb3NzQgQxP1cmUgRmVzd3JhdGVkIFNTy8BZXJ0aWZpY2FQTAeFw0yMDAyMj
Y2OTM2
NTlaFW0yMzAyMjYxOTM2NTlaMDQxMjAwNVRHAMTK1pY3Jvc29mdCB0YWxGbDGZGRl
cmFZQg
U1NPIhEnRmZmljXRIMIIBljANBgkqhkiiG9w0BAQEFAAOCAQ8AMICGCQxKCAQEAYQUnzj9P
-----END CERTIFICATE-----

**STEP 4 | Click Save**

**Prisma Cloud User to AAD User Identity mapping**

If you plan to map Azure Active Directory users to Prisma Cloud accounts perform the following steps.

**STEP 1 | Go to Manage > Authentication > Users.**

**STEP 2 | Click Add user.**

**STEP 3 | Create a New User.**

1. **Username**: Azure Active Directory userprincipalname
2. **Auth Method**: Select SAML
3. **Role**: Select the appropriate role for the user
4. Click Save.

**STEP 4 | Test logging into Prisma Cloud Console via Azure Active Directory SAML federation.**


**Prisma Cloud Groups to AAD Group mapping**

When you use AAD Groups to assign roles within Prisma Cloud you do not have to create a corresponding Prisma Cloud account.

**STEP 1 | Go to Manage > Authentication > Groups.**

**STEP 2 | Click Add Group.**

**STEP 3 | Enter the display name of the AAD group.**

**STEP 4 | Click the SAML group radio button.**

**STEP 5 | Select the Prisma Cloud role for the group.**

**STEP 6 | Click Save**
Azure Active Directory SAML response will send the user’s group membership as OIDs and not the name of the group. When a group is added, Prisma Cloud Console will query the Microsoft Azure endpoints to determine the OID of the group entered. Ensure your Prisma Cloud Console is able to reach https://login.windows.net/ and https://graph.windows.net

STEP 7 | Test logging into Prisma Cloud Console via Azure Active Directory SAML federation.

Leave your existing session logged into Prisma Cloud Console in case you encounter issues. Open a new incognito browser window and go to https://<CONSOLE>:8083.
Integrate with PingFederate via SAML 2.0 federation

Many organizations use SAML to authenticate users for web services. Prisma Cloud supports the SAML 2.0 federation protocol to access the Prisma Cloud Console. When SAML support is enabled, users can log into the Console with their federated credentials. This article provides detailed steps for federating your Prisma Cloud Console with your PingFederate v8.4 Identity Provider (IdP).

The Prisma Cloud/PingFederate SAML federation flow works as follows:

1. Users browse to Prisma Cloud Console.
2. Their browsers are redirected to the PingFederate SAML 2.0 endpoint.
3. They enter their credentials to authenticate. Multi-factor authentication can be enforced at this step.
4. A PingFederate SAML token is returned to Prisma Cloud Console.
5. Prisma Cloud Console validates the SAML token’s signature and associates the user to their Prisma Cloud account via user identity mapping or group membership.

Prisma Cloud Console is integrated with PingFederate as a federated SAML Service Provider. The steps to set up the integration are:

- Configure PingFederate
- Configure Prisma Cloud Console

Configure PingFederate

**STEP 1 |** Logon to PingFederate

**STEP 2 |** Go to **IdP Configuration > SP Connection > Connection Type**, and select **Browser SSO**.
STEP 3 | Go to IdP Configuration > SP Connection > Connection Options, and select Browser SSO Profiles SAML 2.0.
STEP 4 | Skip the Import Metadata tab.

STEP 5 | Go to IdP Configuration > SP Connection > General Info.

1. In Partner's Entity ID, enter `twistlock`.

   *By default, the Partner’s Entity ID is “twistlock”. When configuring the SAML Audience in the Prisma Cloud Console, the default value is “twistlock”. If you choose a different value here, be sure to set the same value in your Console.*

2. In Connection Name, enter `Prisma Cloud Console`.
3. Click Add.
STEP 6 | In Browser SSO > SAML Profiles, select both IDP-INITIATED SSO and SP-INITIATED SSO.
STEP 7 | Go to **Assertion Creation** and set **SAML_SUBJECT** to **SAML 1.1 nameid-format**.

In this example you mapped the user’s email address to the SAML_SUBJECT attribute which matches the user's Prisma Cloud account. If you are using group-to-Prisma Cloud-role associations, add **groups** to the list of attributes to be returned in the SAML token.
STEP 8 | In IdP Configuration > SP Connection > Browser SSO > Protocol Settings > Assertion Consumer Service URL, specify an assertion consumer URL.

1. Under Binding, select POST.
As the IdP, you send SAML assertions to the SP's Assertion Consumer Service. The SP may request that the SAML assertion be sent to one of the possible assertion consumer URLs below and select one to be the default.

<table>
<thead>
<tr>
<th>Default</th>
<th>Index</th>
<th>Binding</th>
<th>Endpoint URL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>POST</td>
<td><a href="https://pfox-tl.lab.twistlock.com">https://pfox-tl.lab.twistlock.com</a></td>
</tr>
</tbody>
</table>
### SP Connection | Browser SSO | Protocol Settings

<table>
<thead>
<tr>
<th>Assertion Consumer Service URL</th>
<th>Allowable SAML Bindings</th>
<th>Signature Policy</th>
<th>Encryption Policy</th>
<th>Summary</th>
</tr>
</thead>
</table>

Additional guarantees of authenticity may be agreed upon between you and your partner. For SP-initiated SSO, you can choose to require signed POST or redirect bindings. You can also choose to sign assertions sent to this partner, regardless of the binding used.

- [ ] REQUIRE AUTHN REQUESTS TO BE SIGNED WHEN RECEIVED VIA THE POST OR REDIRECT BINDINGS
- [ ] ALWAYS SIGN THE SAML ASSERTION

**STEP 10** | In IdP Configuration > SP Connection > Browser SSO > Protocol Settings, review the protocol settings.
STEP 11 | Click Done.

STEP 12 | Copy the PingFederate SAML token signing X.509 certificate as Base64 in Server Configuration. This certificate will be imported into Prisma Cloud Console.

Configure Prisma Cloud Console

Configure Prisma Cloud Console.

STEP 1 | Login to the Prisma Cloud Console as an administrator.

STEP 2 | Go to Manage > Authentication > Identity Providers > SAML.

STEP 3 | Set Integrate SAML users and groups with Prisma Cloud to Enabled.

STEP 4 | Set Identity Provider to Ping.

STEP 5 | In Identity provider single sign-on URL, enter your PingFederate IdP endpoint.

STEP 6 | In Identity provider issuer, enter your PingFederate Entity ID.
STEP 7 | In Audience, enter twistlock (default) or the value you set for Partner's Entity ID in PingFederate.

1. In X.509 certificate, paste your PingFederate X.509 Signing Certificate Base64.
Manage / Authentication

SAML settings

Integrate SAML users and groups with Twistlock: Enabled

Identity provider

Identity provider single sign-on URL: https://pingfederate.example.com:9031/idp/SSO.saml2

Identity provider issuer: your_PingFederate_entity_id

X.509 certificate

|-----BEGIN CERTIFICATE-----
MIIwDwTCCAgmgAwIBAgIUMJumXlk7qOgNMA0GCSqGSIb3DQEBCwUAMGgxCzAJBgNV
BAYTA1YMQswCQYDVQQIDAJJRDESMBAGA1UEFwwJUm9ja3ZpbGxhMRwEAYDVQK
DAIud2lxdzGxY2xOAAOAaNVBAzMAONUTzEWMBOGQ1UEAwN6WJbnRlYwhhbi
hDAeFwQxNTzA5MikwMTExNVAwFwQxODA5MikwMTE2NDVAMGgxCzAJBgNV
hDAeFwQxNTzA5MikwMTExNVAwFwQxODA5MikwMTE2NDVAMGgxCzAJBgNV
hDAeFwQxNTzA5MikwMTExNVAwFwQxODA5MikwMTE2NDVAMGgxCzAJBgNV
-----END CERTIFICATE-----

STEP 8 | Click Save.

PRISMA CLOUD COMPUTE EDITION ADMINISTRATOR’S GUIDE | Authentication 335

© 2020 Palo Alto Networks, Inc.
User account name matching

User account name matching.

**STEP 1** | Go to Manage > Authentication > Users.

**STEP 2** | Click Add user.

**STEP 3** | Create a new user:
1. In Username, enter the value returned within the SAML_SUBJECT attribute IdP user’s email address.
2. In Role, select the appropriate role.
3. Set Create user in local Prisma Cloud account database to Off.

Create A New User

<table>
<thead>
<tr>
<th>Username:</th>
<th><a href="mailto:paul@twistlock.com">paul@twistlock.com</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Role:</td>
<td>Administrator</td>
</tr>
<tr>
<td>Create user in local Twistlock account database:</td>
<td>Off</td>
</tr>
</tbody>
</table>

**STEP 4** | Click Save.

**STEP 5** | Test login into the Prisma Cloud Console via PingFederate SAML federation.

Leave your existing session logged onto the Prisma Cloud Console in case you encounter issues. Open a new incognito browser window and go to https://<CONSOLE>:8083.

Group name matching

Group name matching.

**STEP 1** | Go to Manage > Authentication > Groups.

**STEP 2** | Click the +Add Group button.

**STEP 3** | In the Name field, enter a group name.

The group name must exactly match the group name in the SAML IDP. Console does not verify if that the value entered matches a group name in the SAML IDP.

**STEP 4** | Select the SAML group checkbox.
Create A New Group

<table>
<thead>
<tr>
<th>Name</th>
<th>SAML_Twistlock_Admins</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAML group</td>
<td></td>
</tr>
<tr>
<td>Role</td>
<td>Administrator</td>
</tr>
</tbody>
</table>

**STEP 5** | Click **Save**

**STEP 6** | Test login into the Prisma Cloud Console via PingFederate SAML federation.

Leave your existing session logged onto the Prisma Cloud Console in case you encounter issues. Open a new incognito browser window and go to https://<CONSOLE>:8083.
Integrate with Windows Server 2016 & 2012r2 Active Directory Federation Services (ADFS) via SAML 2.0 federation

Many organizations use SAML to authenticate users for web services. Prisma Cloud supports the SAML 2.0 federation protocol for access to the Prisma Cloud Console. When SAML support is enabled, users can log into Console with their federated credentials. This article provides detailed steps for federating your Prisma Cloud Console with your Active Directory Federation Service (ADFS) Identity Provider (IdP).

Prisma Cloud supports SAML 2.0 federation with Windows Server 2016 and Windows Server 2012r2 Active Directory Federation Services via the SAML protocol. The federation flow works as follows:

1. Users browse to Prisma Cloud Console.
2. Their browsers are redirected to the ADFS SAML 2.0 endpoint.
3. Users authenticate either with Windows Integrated Authentication or Forms Based Authentication. Multi-factor authentication can be enforced at this step.
4. An ADFS SAML token is returned to Prisma Cloud Console.
5. Prisma Cloud Console validates the SAML token's signature and associates the user to their Prisma Cloud account via user identity mapping or group membership.

Prisma Cloud Console is integrated with ADFS as a federated SAML Relying Party Trust.

- Configure Active Directory Federation Services
- Configure the Prisma Cloud Console

The Relying Party trust workflows may differ slightly between Windows Server 2016 and Windows Server 2012r2 ADFS, but the concepts are the same.

Configure Active Directory Federation Services

This guide assumes you have already deployed Active Directory Federation Services, and Active Directory is the claims provider for the service.

STEP 1 | Log onto your Active Directory Federation Services server.

STEP 2 | Go to Server Manager > Tools > AD FS Management to start the ADFS snap-in.

STEP 3 | Go to AD FS > Service > Certificates and click on the Primary Token-signing certificate.

STEP 4 | Select the Details tab, and click Copy to File....
STEP 5 | Save the certificate as a Base-64 encoded X.509 (.CER) file. You will upload this certificate into the Prisma Cloud console in a later step.
STEP 6 | Go to AD FS > Relying Party Trusts.

STEP 7 | Click Add Relying Party Trust from the Actions menu.

1. Step Welcome: select Claims aware.

2. Step Select Data Source: select Enter data about the relying party manually.
3. Step Specify Display Name: In **Display Name**, enter **twistlock Console**.
5. Step Configure URL: select **Enable support for the SAML 2.0 WebSSO protocol**. Enter the URL for your Prisma Cloud Console `https://<FQDN_TWISTLOCK_CONSOLE>:8083/api/v1/authenticate/`. 
6. Step Configure Identifiers: for example enter **twistlock** all lower case and click **Add**.
7. Step Choose Access Control Policy: this is where you can enforce multi-factor authentication for Prisma Cloud Console access. For this example, select **Permit everyone**.
8. Step Ready to Add Trust: no changes, click Next.
9. Step Finish: select **Configure claims issuance policy for this application** then click Close.
10. In the Edit Claim Issuance Policy for Prisma Cloud Console click Add Rule.
11. Step Choose Rule Type: In Claim rule template, select Send LDAP Attributes as Claims.
12. Step Configure Claim Rule:

- Set Claim rule name to Prisma Cloud Console
- Set Attribute Store to Active Directory
- In Mapping of LDAP attributes to outgoing claim types, set the LDAP Attribute to SAM-Account-Name and Outgoing claim type to Name ID.
13. Click **Finish**.

**STEP 8 |** Configure ADFS to either sign the SAML response (-**SamlResponseSignature MessageOnly**) or the SAML response and assertion (-**SamlResponseSignature MessageAndAssertion**) for the Prisma Cloud Console relying party trust. For example to configure the ADFS to only sign the response, start an administrative PowerShell session and run the following command:

```powershell
set-adfsrelyingpartytrust -TargetName "Prisma Cloud Console" -SamlResponseSignature MessageOnly
```

**Active Directory group membership within SAML response**

You can use Active Directory group membership to assign users to Prisma Cloud roles. When a user’s group membership is sent in the SAML response, Prisma Cloud attempts to associate the user’s group to a Prisma Cloud role. If there is no group association, Prisma Cloud matches the user to an identity based on the NameID to Prisma Cloud username mapping. The SAML group to Prisma Cloud role association does not
require the creation of a Prisma Cloud user. Therefore simplify the identity management required for your implementation of Prisma Cloud.

**STEP 1** | In **Relying Party Trusts**, select the **Prisma Cloud Console** trust.

**STEP 2** | Click **Edit Claim Issuance Policy** in the right hand **Actions** pane.

**STEP 3** | Click **Add Rule**.

**STEP 4** | **Claim rule template**: Send Claims Using a Custom Rule.

**STEP 5** | Click **Next**.

**STEP 6** | **Claim rule name**: Prisma Cloud Groups.

**STEP 7** | Paste the following claim rule into the **Custom rule** field:

```c
[type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/windowsaccountname", issuer == "AD AUTHORITY"] => issue(store = "Active Directory", types = ["groups"], query = ";tokenGroups;{0}", param = c.Value);
```

---

**Configure the Prisma Cloud Console**

Configure the Prisma Cloud Console.

**STEP 1** | Login to the Prisma Cloud Console as an administrator.

**STEP 2** | Go to **Manage > Authentication > Identity Providers > SAML**.

**STEP 3** | Set **Integrate SAML users and groups with Prisma Cloud** to **Enabled**.

**STEP 4** | Set **Identity Provider** to ADFS.

**STEP 5** | In **Identity provider single sign-on URL**, enter your SAML Single Sign-On Service URL. For example **https://FQDN_of_your_adfs/adfs/ls**.

**STEP 6** | In **Identity provider issuer**, enter your SAML Entity ID, which can be retrieved from **ADFS > Service > Federation Service Properties : Federation Service Identifier**.

**STEP 7** | In **Audience**, enter the ADFS Relying Party identifier **twistlock**

**STEP 8** | In **X.509 certificate**, paste the ADFS **Token Signing Certificate Base64** into this field.
**STEP 9 |** Click **Save**.

**STEP 10 |** Go to **Manage > Authentication > Users**.

**STEP 11 |** Click **Add user**.

1. **Username**: Active Directory `samAccountName` must match the value returned in SAML token's Name ID attribute.

   *When federating with ADFS Prisma Cloud usernames are case insensitive. All other federation IdPs are case sensitive.*

2. **Auth method**: set to **SAML**.
3. **Role**: select an appropriate role.

**STEP 12** | Click **Save**.

**Active Directory group membership mapping to Prisma Cloud role**

Associate a user’s Active Directory group membership to a Prisma Cloud role.

**STEP 1** | Go to Manage > Authentication > Groups.

**STEP 2** | Click **Add group**.

**STEP 3** | **Group Name** matches the Active Directory group name.

**STEP 4** | Select the **SAML group** radio button.

**STEP 5** | Assign the **Role**.

---

The **SAML group to Prisma Cloud role association does not require the creation of a Prisma Cloud user.**
STEP 6 | Test login into the Prisma Cloud Console via ADFS SAML federation.

Leave your existing session logged onto the Prisma Cloud Console in case you encounter issues. Open a new incognito browser window and go to https://<CONSOLE>:8083.
Integrate Prisma Cloud with GitHub

Prisma Cloud supports OAuth 2.0 as an authentication mechanism. GitHub users can log into Prisma Cloud Console using GitHub as an OAuth 2.0 provider.

Prisma Cloud supports the authorization code flow only.

Configure Github as an OAuth provider

Create an OAuth App in your GitHub organization so that users in the organization can log into Prisma Cloud using GitHub as an OAuth 2.0 provider.

**STEP 1** | Log into GitHub as the organization owner.

**STEP 2** | Go to Settings > Developer Settings > OAuth Apps, and click New OAuth App (or Register an application if this is your first app).

**STEP 3** | In Application name, enter Prisma Cloud.

**STEP 4** | In Homepage URL, enter the URL for Prisma Cloud Console in the format https://<CONSOLE>:<PORT>.

**STEP 5** | In Authorization callback URL, enter https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oauth.

**STEP 6** | Click Register application.

**STEP 7** | Copy the Client ID and Client Secret, and set them aside setting up the integration with Prisma Cloud.
Integrate Prisma Cloud with GitHub

Set up the integration so that GitHub users from your organization can log into Prisma Cloud.

**STEP 1** | Log into Prisma Cloud Console.

**STEP 2** | Go to Manage > Authentication > Identity Providers > OAuth 2.0.

**STEP 3** | Set Integrate Oauth 2.0 users and groups with Prisma Cloud to Enabled.

**STEP 4** | Set Identity provider to GitHub.

**STEP 5** | Set Client ID and Client secret to the values you copied from GitHub.

**STEP 6** | Set Auth URL to `https://github.com/login/oauth/authorize`.

**STEP 7** | Set Token URL to `https://github.com/login/oauth/access_token`.

**STEP 8** | Click Save.
Prisma Cloud to GitHub user identity mappings

Create a Prisma Cloud user for each GitHub user that should have access to Prisma Cloud.

After the user is authenticated, Prisma Cloud uses the access token to query GitHub for the user’s information (user name, email). The user information returned from GitHub is compared against the information in the Prisma Cloud Console database to determine if the user is authorized. If so, a JWT token is returned.

**STEP 1** | Go to Manage > Authentication > Users.

**STEP 2** | Click Add User.

**STEP 3** | Set Username to the GitHub user name.

**STEP 4** | Set Auth method to OAuth.

**STEP 5** | Select a role for the user.

**STEP 6** | Click Save.

**STEP 7** | Test logging into Prisma Cloud Console.

1. Logout of Prisma Cloud.
2. On the login page, select OAuth, and then click Login.

3. Authorize the Prisma Cloud OAuth App to sign you in.
Prisma Cloud group to GitHub organization mappings

Use groups to streamline how Prisma Cloud roles are assigned to users. When you use groups to assign roles, you don’t have to create individual Prisma Cloud accounts for each user.

Groups can be associated and authenticated with by multiple identity providers.

**STEP 1** | Go to Manage > Authentication > Groups.

**STEP 2** | Click Add Group.

**STEP 3** | In Name, enter the the GitHub organization.

**STEP 4** | In Authentication method, select External Providers.

**STEP 5** | In Authentication Providers, select OAuth group.
STEP 6 | Select a role for the members of the organization.

STEP 7 | Click Save.

STEP 8 | Test logging into Prisma Cloud Console.
   1. Logout of Prisma Cloud.
   2. On the login page, select OAuth, and then click Login.
   3. Authorize the Prisma Cloud OAuth App to sign you in.
Authorize Prisma Cloud

Prisma Cloud by ficqco
wants to access your ficq account

Organizations and teams
Read-only access

Personal user data
Profile information (read-only)

Organization access
ficqco

Authorize ficqco

Authorizing will redirect to
https://34.237.91.210:8083
Integrate Prisma Cloud with OpenShift

OpenShift users can log into Prisma Cloud Console using OpenShift as an OAuth 2.0 provider.

The OpenShift master includes a built-in OAuth server. You can integrate OpenShift authentication into Prisma Cloud. When users attempt to access Prisma Cloud, which is a protected resource, they are redirected to authenticate with OpenShift. After authenticating successfully, they are redirected back to Prisma Cloud Console with an OAuth token. This token scopes what the user can do in OpenShift. Prisma Cloud only needs the auth token to get the user’s info (e.g. user name, email), and check the Prisma Cloud database to see if this user is authorized. If so, Prisma Cloud creates a JWT token, with a role claim, to complete the authentication process to Console. Roles are assigned based on users and group information specified in Console.

The following diagram shows the login flow when the auth provider is LDAP. With LDAP, users enter their credentials in Prisma Cloud Console, and Prisma authenticates with the LDAP server on the user’s behalf. With all other auth providers, Prisma isn’t part of verifying the user credentials. Instead Prisma redirects the client to the auth provider for authentication. Once the user successfully authenticates via the authentication provider, the client is redirected back to Prisma Cloud Console with an object (SAML assertion for SAML, JWT token for OIDC, Access token for OAuth 2.0) that proves a successful login or, in the OAuth 2.0 case, gives us access to the application to verify the user identity.

Prisma Cloud supports the authorization code flow only.

Integrate Prisma Cloud with OpenShift

Configure Prisma Cloud so that OpenShift users can log into Prisma Cloud with the same identity.

**STEP 1** | In OpenShift, register Prisma Cloud as an OAuth client. Set the redirect URL to: https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oauth.

**STEP 2** | Log into Prisma Cloud Console.

**STEP 3** | Go to Manage > Authentication > Identity Providers > OAuth 2.0.

**STEP 4** | Set Integrate OAuth 2.0 users and groups with Prisma Cloud to Enabled.

**STEP 5** | Set Identity provider to OpenShift.

**STEP 6** | Set Client ID to the name of the OAuth client you set up in OpenShift.
STEP 7 | Set Client secret to the secret in the OAuth client you set up in OpenShift.

STEP 8 | Set Auth URL to https://github.com/login/oauth/authorize.

STEP 9 | Set Token URL to https://github.com/login/oauth/access_token.

STEP 10 | In User Info API URL, enter the TCP endpoint for the OpenShift API server. For example, https://openshift.default.svc.cluster.local.

STEP 11 | Click Save.

Prisma Cloud to OpenShift user identity mappings

Create a Prisma Cloud user for every OpenShift user that should have access to Prisma Cloud.

After the user is authenticated, Prisma Cloud uses the access token to query OpenShift for the user’s information (user name, email). The user information returned from OpenShift is compared against the Prisma Cloud Console database to determine if the user is authorized. If so, a JWT token is returned.

STEP 1 | Go to Manage > Authentication > Users.

STEP 2 | Click Add User.

STEP 3 | Set Username to the OpenShift user name.

STEP 4 | Set Auth method to OAuth.

STEP 5 | Select a role for the user.

STEP 6 | Click Save.

STEP 7 | Test logging into Prisma Cloud Console.

1. Logout of Prisma Cloud.
2. On the login page, select OAuth, and then click Login.
3. Authorize the Prisma Cloud OAuth App to sign you in.
Prisma Cloud to OpenShift group mappings

Use groups to streamline how Prisma Cloud roles are assigned to users. When you use groups to assign roles, you don’t have to create individual Prisma Cloud accounts for each user.

Groups can be associated and authenticated with by multiple identity providers.

**STEP 1** | Go to Manage > Authentication > Groups.

**STEP 2** | Click Add Group.

**STEP 3** | In Name, enter an OpenShift group name.

**STEP 4** | In Authentication method, select External Providers.

**STEP 5** | In Authentication Providers, select OAuth group.
STEP 6 | Select a role for the members of the group.

STEP 7 | Click Save.

STEP 8 | Test logging into Prisma Cloud Console.
1. Logout of Prisma Cloud.
2. On the login page, select OAuth, and then click Login.
3. Authorize the Prisma Cloud OAuth App to sign you in.
Non-default UPN suffixes

Active Directory allows administrators to specify custom UPN suffixes that can be applied to user accounts. The default UPN suffix for a user account is the Domain Name System (DNS) domain name of the domain that contains the user account. Microsoft refers to this as the implicit UPN. Administrators may choose to add additional suffixes to shorten user names or provide consistent UPNs across a forest composed of multiple domains; these are known as explicit UPNs.

For example, if a domain is named domain.directory.company.com, the default UPN suffix would be domain.directory.company.com and users could logon with username@domain.directory.company.com. However, an admin may want to simplify this and provide an alternative UPN suffix like @company.com that would apply to all users across a forest. Users could then logon with this explicit UPN of username@company.com instead.

Within the directory service, the userPrincipalName attribute is updated to reflect whatever username + UPN suffix the administrator applies to a given account. In Windows systems, the implicit UPN can be used in addition to whatever explicit UPN may be set. However, for non-Windows LDAP systems, the explicit UPN is the only valid UPN that can be used with the user object.

Thus, understanding the UPN assigned to a user account is critical to Prisma Cloud integration with Active Directory. Even if the domain name and the search path may use one set of names (such as dc=domain,dc=directory,dc=company,dc=com in our above example), the actual (explicit) UPN must be used for all actions within Prisma Cloud, such as adding users to the system or logging on. From our above example, this means that if the user in Active Directory has a UPN of username@domain.directory.company.com set on their account, this UPN must be used with Prisma Cloud. Alternatively, if an Active Directory admin has set another UPN, such as username@company.com, that UPN must be used instead.

Any attempts to use a UPN not directly found in the userPrincipalName field on the user object will result in ‘user not found’ errors.
Compute user roles

You can assign roles to users to control their level of access to Prisma Cloud. Roles determine what a user can do and see in Console, and the APIs he or she can access.

Summary of available roles

The following table summarizes the roles available in Prisma Cloud.

<table>
<thead>
<tr>
<th>Role</th>
<th>Access level</th>
<th>Typical use case(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator</td>
<td>Full read-write access to all Prisma Cloud settings and data.</td>
<td>Security administrators.</td>
</tr>
<tr>
<td>Operator</td>
<td>Read-write access to all rules and data. Read-only access to user and group management, role assignments, and the global scan settings under Manage &gt; System &gt; Scan.</td>
<td>Security operations teams.</td>
</tr>
<tr>
<td>Auditor</td>
<td>Read-only access to all Prisma Cloud rules and data.</td>
<td>Auditors and compliance staff that need to verify settings and monitor compliance.</td>
</tr>
<tr>
<td>DevSecOps User</td>
<td>Read-only access to all results under Radar and Monitor, but no access to view or change policy or settings. Read-only access to User certificates and Downloads.</td>
<td>DevSecOps personnel.</td>
</tr>
<tr>
<td>Vulnerability Manager</td>
<td>Define policy and monitor vulnerabilities and compliance.</td>
<td>DevOps users that also need to define policy and monitor vulnerabilities and compliance.</td>
</tr>
<tr>
<td>DevOps User</td>
<td>Read-only access to the Prisma Cloud CI vulnerability, compliance scan reports, User certificates, and Downloads.</td>
<td>Developer, Operations, and DevOps personnel that need to know about and/or address the vulnerabilities in your environment.</td>
</tr>
<tr>
<td>Defender Manager</td>
<td>Install, manage, and remove Defenders from your environment.</td>
<td>DevOps team members that need to manage Defender deployments without sysadmin privileges.</td>
</tr>
<tr>
<td>Access User</td>
<td>Install personal certificates only.</td>
<td>Developers (and others) that use the nodes that Prisma Cloud protects.</td>
</tr>
<tr>
<td>CI User</td>
<td>Run the Continuous Integration plugin only.</td>
<td>CI Users can only run the plugin and have no other access to configure Prisma Cloud.</td>
</tr>
</tbody>
</table>
Let's look at how two roles at the opposite end of the spectrum differ: Administrator and User. Administrators set the security policy. They decide who can run what Docker commands, and where they can be run. Users need to run Docker commands to do their job. Testers, for example, run Docker commands in the staging environment to test containers under development. Testers, however, have no business starting containers in the production environment. Administrators set a policy to assign testers the user role that lets testers run Docker commands in staging, but restricts their access to production.

If a user is assigned multiple roles, either directly or through group inheritance, then he is granted the rights of the highest role. For example, assume Bruce is part of GroupA and GroupB in Active Directory. In Console, you assign the Administrator role to GroupA and the Auditor role to GroupB. When Bruce logs into Prisma Cloud, he will have Administrator rights.

Roles are enforced the same way for both the Prisma Cloud UI and the Prisma Cloud API.

To learn how to assign roles to users and groups, see Assigning roles.

Roles

This section describes all the roles Prisma Cloud supports.

Administrator

The Administrator can manage all aspects of your Prisma Cloud installation. They have full read-write access to all Prisma Cloud settings and data.

Administrators can:

- Create and update security policies.
- Create and update access control policies.
- Create and update the list of users and groups that can access Prisma Cloud.
- Assign roles to users and groups.
- The Admin role is reserved for security administrators.

When Administrators log into Console, they have access to the full dashboard. If you click on the profile button on the top right of the dashboard, you get the details of the currently logged in user (admin) and associated role (Administrator).
Operator

Operators can create and update all Prisma Cloud settings. This role lets you view audit data and manage the rules that define your policies.

Operators cannot:

- Create, update, or delete users or groups.
- Assign or reassign roles to any user or group.
- Change the global scan settings under Manage > System > Scan.

The Operator role is designed for members of your Security Operations team.

Auditor

Auditors get read-only access to all Prisma Cloud data, settings, and logs.

Auditors are typically members of your compliance team. They verify that your Prisma Cloud setup meets your organization’s security requirements. To verify compliance, they must be able to see your settings, but they do not need to make changes to them.

Auditors have access to the downloads page (Manage > System > Downloads), but cannot download Defender images.

DevSecOps User

DevSecOps Users get access to all views under Radar and Monitor. Access to the Actions menu in these views is disabled. The Actions menu lets you do things such as relearn models, protect services found by Cloud Discovery, and so on.

DevSecOps Users can’t access the views under Defend.

Under Manage, they only get access to:

- Manage > Authentication > User Certificates
- Manage > System > Downloads. This page lets you download various Prisma Cloud components. DevSecOps Users can download all files, except Defender images, which are disabled for this role.

Vulnerability Manager

Vulnerability Managers define and monitor vulnerabilities and compliance policy. Vulnerability Managers gain the following permissions:

- Read-write access to Defend > Vulnerabilities and Defend > Compliance.
- Read-write access to Monitor > Vulnerabilities, Monitor > Compliance and Monitor > Events > Trust Audits.
- Read-only access to Manage > Authentication > User certificates and Manage > System > Downloads. The Downloads page lets you download various Prisma Cloud components. Vulnerability Managers can download all files, except Defender images, which are disabled for this role.

DevOps User

DevOps Users get read-only access to the Jenkins Jobs and Twistcli Scans tabs under Monitor > Vulnerabilities and Monitor > Compliance. Each tab contains scan reports for images scanned using these tools. DevOps Users can use Prisma Cloud scan reports and tools, for example, to determine why the CI/CD pipeline is stalled.
Additionally, DevOps users can use the CVE Viewer to query the Prisma Cloud CVE database.

Under **Manage**, they only get access to:

- **Manage > Authentication > User Certificates**
- **Manage > System > Downloads.** This page lets you download various Prisma Cloud components. DevOps Users can download all files, except Defender images, which are disabled for this role.

**Defender Manager**

Defender Managers get read-write access to **Manage > Defenders, Manage > Authentication > User certificates** and **Manage > System > Downloads**.

Defender Managers can install, manage, and remove Defenders from your environment. The Defender Manager role was designed to let members of your DevOps team manage the hosts that Prisma Cloud protects without requiring Administrator-level privileges. To help debug Defender deployment issues, Defender Managers get read-only access to Prisma Cloud settings and log files.

Defender Managers are typically members of your DevOps team. They need to manage the hosts that Prisma Cloud protects, but they never need to alter any security policies.

Defender Managers are also used to automate the installation of Defenders. If you use the API to programmatically install new Defenders in an environment that is not orchestrated by Kubernetes or Swarm, create a service account with the Defender Manager role, then follow the instructions in **Automate Defender install**.

Access to **Manage > Defenders > Names** is blocked as it requires API access for certificate management.

> This role can see view the secrets that Defenders use to do their job, such as cloud credentials for registry scanning.

**Access User**

Users work with Docker containers. They run Docker client commands on the hosts that are protected by the Defender. The commands they run include:

- Pulling an image from a registry.
- Starting a container on a host.
- Stopping a container.

Users are typically members of your engineering team. For example, all members of your test team would be assigned the User role.

The following screenshot shows the view that user with the User role see when they log into Console.

Users log into Console to get their client certificates, which they install on their machines to identify them. When a user runs a Docker command on a host, Defender checks that the user has the right permissions to run that command on that host.
The CI user role can be assigned to users that should only be able to run the plugin but have no other access to configure Prisma Cloud or view the data that we have. It is designed to only provide the minimal amount of access required to run the plugins.

A CI user cannot log into the Console or even view the UI Dashboard.
Assign roles

After creating a user or group, you can assign a role to it. Roles determine the level of access to Prisma Cloud’s data and settings.

Prisma Cloud supports two types of users and groups:

- Centrally managed users and groups, defined in your organization’s directory service. With directory services such as Active Directory, OpenLDAP, and SAML providers, you can re-use the identities set up in these systems.
- Prisma Cloud users and groups, created and managed from Console. For centrally managed users groups, roles can be assigned after you integrate your directory service with Prisma Cloud. Roles can be assigned to individual users or to groups. When you assign a role to a group, all members of the group inherit the role. Managing role assignments at the group level is considered a best practice. Groups provide an easier way to manage a large user base, and simpler foundation for building your access control policies.

For Prisma Cloud users and groups, roles are assigned at the user level when the user is created. When you create a Prisma Cloud group, you add Prisma Cloud users to it. Users in this type of group always retain the role they were assigned when they were created.

Assigning roles to Prisma Cloud users

If you do not have a directory service, such as Active Directory (AD) or Lightweight Directory Access Protocol (LDAP), Prisma Cloud lets you create and manage your own users and groups. When you create a Prisma Cloud user, you can assign it a role, which determines its level of access.

To create a user and assign it a role:

**STEP 1 |** Open Console, and log in with your admin credentials.

**STEP 2 |** Go to Manage > Authentication > Users.

**STEP 3 |** Click Add user.

1. Enter a username.
2. Enter a password.
3. Assign a role.
4. Click Save.

Assigning roles to Prisma Cloud groups

Collecting users into groups makes it easier to manage your access control rules.

> Each user in the group retains his own role to prevent erroneous privilege escalation.

To create a Prisma Cloud group and add users to it:

**STEP 1 |** Open Console and log in with your admin credentials.

**STEP 2 |** Go to Manage > Authentication > Groups.

**STEP 3 |** Click Add group.

1. Enter a name for your group.
2. In the drop down list, select a user.
3. Click +.
4. Repeat steps b to c until your group contains all the members you want.
5. Click *Save:

Assigning roles to AD/OpenLDAP/SAML users

By default, AD/OpenLDAP/SAML users have the very basic Access User role. You can grant users a different level of access to Console by assigning them roles.

*If a user is a part of an AD, OpenLDAP, or SAML group, and you have assigned a role to the group, the user inherits the group’s role.*

Prerequisites: You have integrated Prisma Cloud with Active Directory, OpenLDAP, or SAML.

STEP 1 | Open Console.

STEP 2 | Log in with your admin credentials.

STEP 3 | Go to Manage > Authentication > Users.

STEP 4 | Click Add user.

1. Enter the username for the user whose role you want to set. For example, if you have integrated Prisma Cloud with Active Directory, enter a UPN.
2. In the Role drop-down menu, select a role.
3. Click Save.

Assigning roles to AD/OpenLDAP/SAML groups

You can assign an AD/OpenLDAP/SAML group a role. Members of the group inherit the group’s role. When a user from a group tries to access a resource protected by Prisma Cloud, Prisma Cloud resolves the member’s role on the fly.

*If a user is assigned multiple roles, either directly or through group inheritance, then he is granted the rights of the highest role. For example, assume Bruce is part of GroupA and GroupB in Active Directory. In Console, you assign the Administrator role to GroupA and the Auditor role to GroupB. When Bruce logs into Prisma Cloud, he will have Administrator rights.*

The following procedure shows you how to assign a role to an existing AD/OpenLDAP/SAML group:

Prerequisites: You have integrated Prisma Cloud with Active Directory, OpenLDAP, or SAML.

STEP 1 | Open Console, and log in with your admin credentials.

STEP 2 | Go to Manage > Authentication > Groups.

STEP 3 | Click Add group.

1. Specify the name of the group. It should match the group name specified in your directory service.
2. Check LDAP group.
3. Select a role.
4. Click Save.
Use custom certificates for authorization

By default, Prisma Cloud uses an internal, self-managed certificate authority for all x.509 cryptographic functions. Prisma Cloud self-managed certificates are used for TLS access to Console’s web UI, for Defenders to authenticate to Console, and for users to authenticate to Defenders.

The self-managed certificate generated by Console is valid for three years. A month prior to expiration, Prisma Cloud will let you rotate it (a banner will appear at the top of the UI). After rotating Console’s certificate, you must manually redeploy your Defenders.

Prisma Cloud lets you use your own certificates for some functions. For example, if you want to make access to the web UI more seamless, you can use certificates from an implicitly trusted CA for securing the TLS connection. You can also use custom certificates to authenticate to Defenders using the Docker and Kubernetes clients. Your custom certificates are used in place of those generated by Prisma Cloud.

Setting up your custom certs

To set up your custom certs:

**STEP 1** | Open Console, and go to Manage > Authentication > Certificates.

**STEP 2** | Set Advanced certificate configuration to Show.

**STEP 3** | Under Console Authentication (section 2), upload the CA certificate(s) in PEM format, then click Save.

If you have multiple CAs, such as a root CA and several issuing CAs, you must add all these certificates into the PEM file. The order of certificates in the PEM file should be from the lowest tier of the hierarchy to the root. For example, if you have a 3 tier hierarchy that looks like this:

```
->RootCA
   ->IntermediateCA
   ->IssuingCA1
   ->IssuingCA2
```

Your PEM file should be ordered as IssuingCA1, IssuingCA2, IntermediateCA, RootCA. To create such a PEM file, you'd get the public keys of each CA in PEM format and concatenate them together:

```
$ cat IssuingCA1.pem IssuingCA2.pem IntermediateCA.pem RootCA.pem > CAs.pem
```

Once this configuration is enabled, users must copy their keys (both public and private) to the host they’re using to run commands with docker or kubectl. Though the path can be referenced in each command, it’s usually simpler to place them in the default directory that docker looks in for certificates (~/.docker).

Each user certificate used with Prisma Cloud must have the user’s CN embedded in the Subject field of the certificate. You can validate these settings by running the following command against the certificate:

```
$ openssl x509 -in .docker/cert.pem -text | grep Subj
```
Finally, Docker requires that the CA certificate used to sign the server certificate on the nodes Prisma Cloud is protecting must also be in the ~/.docker folder, in a file called ca.pem. Because the 'server' certificate used in this deployment model is still generated by Prisma Cloud, this means that on each host where you’re running docker or kubectl commands, you must also add the CA certificate to this folder.

Explicit certificate trust list

You also have an option to create a list of explicitly trusted custom certificates. A typical use case of this feature would be when you have multiple certificates issued to a given user but only want specific ones to be available for use with Prisma Cloud. By adding an explicit trust list, you can control what certificates can be used because Prisma Cloud compares any certificates presented to it against the allowed trusted-certificates-list. This way, a user having certificate not in the explicitly allowed list will not be able to use the certificate with Prisma Cloud, even if it was issued by a trusted CA. Note that this feature is valid only when custom CA is configured. When enabled, it allows users to add new certificates to a table by uploading entire public certificates in PEM format.

NOTES:

- External certification authority section will be visible only to an Admin role user.
- All trusted certs information will be retrieved from the certificate itself, so the user doesn’t have to manually add info such as CN, issuer etc.
- Only the public portion of a user certificate should be added to the explicit trust list. Private keys are not required and should be excluded from this process.

Setting up

When a custom cert is provided to authenticate to Prisma Cloud, it first checks the certificate against this list. If the cert is matched to an entry in the list, then the previously existent flow continues. If the cert is not in the trusted list, then the authentication fails with an error ‘Certificate not in certificate trust list configured in Prisma Cloud’.

**STEP 1** | Under **Authentication to Defenders** (section 1a), upload CA certificate to trust.
STEP 2 | Set Enable authentication with only an explicit list of trusted certificates to ON.

STEP 3 | Click Add certificate, copy the PEM-formatted public certificate which was issued by the trusted CA, then click Add.
## Credentials store

Container environments tend to utilize many third party services across multiple cloud providers. To improve accessibility and reusability, Prisma Cloud manages all credentials in a central encrypted store. Credentials are used when setting up the following integrations:

- Scanning (container registries, serverless functions, etc).
- Alerting in third party services (email, Slack, ServiceNow, etc).
- Deploying and managing Defender DaemonSets from the Console UI.
- Injecting secrets from secret stores into containers at runtime.

The credential store can be found under **Manage > Authentication > Credentials Store**. Credentials cannot be deleted if they are currently in use. To see all the places where a credentials is being used, click on an entry in the credentials store table, and review the **Usages** list.

If a credential is being used by an integration, and you edit its parameters (e.g. username, password, etc), the new values are automatically propagated to the right places in the product. You don't need to delete and set up the integration again to refresh the credential’s values.

### Edit Credential

<table>
<thead>
<tr>
<th>Name</th>
<th>AWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>AWS</td>
</tr>
<tr>
<td>Access Key</td>
<td>AKIAIYP7JO7DYCDIKPA</td>
</tr>
<tr>
<td>Secret Key</td>
<td>Secret is stored in encrypted format, click here to replace</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usages</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cloud Scan</td>
<td>Used for scanning cloud provider account</td>
</tr>
<tr>
<td></td>
<td>Serverless Scan</td>
<td>Used for scanning serverless functions in aws-us-east-1</td>
</tr>
</tbody>
</table>
AWS

Prisma Cloud lets you authenticate with AWS three different ways:

- IAM users
- IAM roles
- Security Token Service (STS) (Recommended)

IAM users

IAM users are entities that represent persons or applications. The Prisma Cloud credentials store lets you save credentials for IAM users. When creating a new credential, select the AWS type, and enter an access key ID and secret access key.

Create A New Credential

<table>
<thead>
<tr>
<th>Name</th>
<th>Example AWS IAM user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>AWS</td>
</tr>
<tr>
<td>Access Key</td>
<td>AKIAI63E17UDV6OD4A</td>
</tr>
<tr>
<td>Secret Key</td>
<td>........................</td>
</tr>
</tbody>
</table>

IAM roles

IAM roles are identities, associated with permissions policies, that can be assumed by the resources that need them. Setting up Prisma Cloud to authenticate with IAM roles is handled in the subsystem-specific configuration dialogs.

For example, when you configure Prisma Cloud to scan an ECR repository, go to Defend > Vulnerabilities > Registry, and click Add registry. In the Version drop-down list, select Amazon EC2 Container Registry, and set Use IAM role to On. No other credentials need to be provided.
How Prisma Cloud accesses IAM role credentials

Roles provide a way to grant credentials to applications that run on EC2 instances to access other AWS services, such as ECR. IAM dynamically provides temporary credentials to the EC2 instances, and these credentials are automatically rotated for you.

When you create an EC2 instance, you can assign it a role. When the instance is started, the AWS instance metadata service (IMDS) attaches your credentials to the running EC2 instance. You can access this metadata from within the instance using the following command:

```
```

Where `<POLICY_NAME>` is assigned to the EC2 instance when it is created or at some point during its life.

The following diagram shows all the pieces. Defender retrieves the credentials from the metadata service, then uses those credentials to retrieve and scan the container images in ECR.
AWS Security Token Service (STS) (Recommended)

AWS Security Token Service (STS) lets you request temporary, limited-privilege credentials for AWS IAM users or users that you authenticate (federated users). Like IAM roles, setting up Prisma Cloud to use STS is handled in the subsystem-specific configuration dialogs, such as registry scanning and serverless function scanning.

This method is recommended as a best practice per the AWS Well-Architected Framework.

When you configure integration with an AWS resource, you can pick an AWS credential from the central store, then use STS to change the role of the account. AWS STS lets you have a few number of IAM identities that can be used across many AWS accounts. For example, if you were setting up Prisma Cloud to scan an AWS ECR registry, you would select the AWS credentials from the central store. Then you would enable Use AWS STS, and enter the name of the STS role to assume in the target account.

When using AWS STS, ensure the following:

- The policy of the IAM user you use as credentials has `sts:AssumeRole` permission on the IAM role you're going to assume. Sample policy:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::123456789123:role/stsIAMrole"
        }
    ]
}
```
• The IAM role you’re going to assume has the IAM user mentioned above configured as a trusted entity. Sample trusted entity policy:

```json
{  
    "Version": "2012-10-17",
    "Statement": [  
        {  
            "Effect": "Allow",
            "Principal": {  
                "AWS": "arn:aws:iam::123456789123:user/prismaUser"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

## Azure

This section discusses Azure credentials.

### Creating an Azure Service Principal

Create an Azure Service Principal so that Prisma Cloud Console can scan your Azure tenant for microservices. To get a service key:

**STEP 1 |** Download and install the Azure CLI.

**STEP 2 |** Create a service principal and configure its access to Azure resources.

```
$ az ad sp create-for-rbac \
   --name <user>.twistlock-azure-cloud-discovery-<contributor|reader> \
   --role <reader|contributor> \
   --sdk-auth
```

The `--role` value depends upon the type of scanning:

- **contributor** = Cloud Discovery + Azure Container Registry Scanning + Azure Function Apps Scanning
- **reader** = Cloud Discovery + Azure Container Registry Scanning

**STEP 3 |** Copy the output of the command and set it aside. It will be used as the **Service Key** when creating an Azure credential.

```json
{
    "clientId": "bc968c1e-6793-4ba5-8d05-f807abb54a57",
    "clientSecret": "5ce0f4ec-5291-42f8-gbe3-90bb3f42ba14",
    "subscriptionId": "ae01981e-e1bf-49ec-ad81-80rf157a944e",
    "tenantId": "d189c61b-6c27-41d3-9749-ca5c9cc4a622",
    "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",
    "resourceManagerEndpointUrl": "https://management.azure.com/",
    "activeDirectoryGraphResourceId": "https://graph.windows.net/",
    "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",
    "galleryEndpointUrl": "https://gallery.azure.com/",
    "managementEndpointUrl": "https://management.core.windows.net/
```

Storing the credential in Prisma Cloud

Store the service principal's credentials in Console so that Prisma Cloud can authenticate with Azure for scanning.

STEP 1 | Open Console, and go to Manage > Authentication > Credentials Store.

STEP 2 | Click Add credential, and enter the following values:
1. In the Name field, enter a label to identify the credential.
2. In the Type field, select Azure.
3. In the Service Key field, enter the value returned by the Azure CLI tool when you created the service principal.
4. Click Save.

Google Cloud Platform (GCP)

Accessing GCP to scan resources can be done in one of two ways. You can make use of a service account and create a key for that account or you can use an API Key. Google recommends that you use a service account with a key and we document that here. More information is available here https://cloud.google.com/docs/authentication/api-keys

Creating a service account

Create a service account that Prisma Cloud can use to scan your resources in GCP.

STEP 1 | Google provide a comprehensive guide for creating a service account - https://cloud.google.com/iam/docs/creating-managing-service-accounts

STEP 2 | Create a key for this service account. The format of this key should be JSON. Google have a guide for this - https://cloud.google.com/iam/docs/creating-managing-service-account-keys

STEP 3 | Copy the contents of the downloaded key, here is an example:

```
{"type": "service_account",
"project_id": "mycompany-project",
"private_key_id": "abe29a57a09f2e2a070f6a60741f83d1c",
"private_key": "-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyB6fS7mX

,password: "your_password"}
```
Storing the credential in Prisma Cloud

Store your GCP credential in Prisma Cloud.

STEP 1 | Open Console, and go to Manage > Authentication > Credentials Store.

STEP 2 | Click Add credential, and enter the following values:

1. In the Name field, enter a label to identify the credential.
2. In the Type field, select GCP.
3. In the Service Account field, copy and paste the entire JSON key that you downloaded.
4. Leave the API token blank
5. Click Save.

IBM Cloud

Prisma Cloud integrates with IBM Cloud Security Advisor. To enable the integration, you must provide credentials, which consist of an Account GUID and API Key.
Kubeconfig

Kubernetes stores cluster authentication information in a YAML file known as kubeconfig. The kubeconfig file grants access to clients, such as kubectl, to run commands against the cluster. By default, kubeconfig is stored in $HOME/.kube/config.

Prisma Cloud uses the kubeconfig credential to deploy and upgrade Defender DaemonSets directly from the Console UI. If you plan to manage DaemonSets from the command line with kubectl, you don’t need to create this credential type.

The user or service account in your kubeconfig must have permissions to create and delete the following resources:
- ClusterRole
- ClusterRoleBinding
- DaemonSet
- Secret
- ServiceAccount

Prisma Cloud doesn’t currently support kubeconfig credentials for Google Kubernetes Engine (GKE) or AWS Elastic Kubernetes Service (EKS) clusters. The kubeconfig for these services require external binaries for authentication (Google Cloud SDK and aws-iam-authenticator respectively), and Prisma Cloud Console doesn’t ship with these binaries.

**STEP 1** | Open Console, and go to Manage > Authentication > Credentials Store.

**STEP 2** | Click Add credential, and enter the following values:

1. In Name, enter a label to identify the credential.
2. In Type, select Kubeconfig.
3. In Kubeconfig, paste the contents of your kubeconfig file.
Vulnerability management

Identify and prevent vulnerabilities across the entire application lifecycle while prioritizing risk for your cloud native environments. Integrate vulnerability management into any CI process, while continuously monitoring, identifying, and preventing risks to all the hosts, images, and functions in your environment. Prisma Cloud combines vulnerability detection with an always up-to-date threat feed and knowledge about your runtime deployments to prioritize risks specifically for your environment.

- Vulnerability Explorer
- Vulnerability management rules
- Search CVEs
- Scan reports
- Customize image scanning
- Configure registry scans
- Configure VM image scanning
- Configure code repository scanning
- Malware scanning
- Vulnerability risk tree
- Detect vulnerabilities in unpackaged software
- CVSS scoring
- Google Cloud Container Builder
- Windows container image scanning
- Serverless function scanning
- VMWare Tanzu blobstore scanning
Vulnerability Explorer

Most scanners find and list vulnerabilities, but Vulnerability Explorer takes it a step further by analyzing the data within the context of your environment. Because Prisma Cloud can see how the containers run in your environment, we can identify the biggest risks and prioritize them for remediation.

To view Vulnerability Explorer, open Console, then go to Monitor > Vulnerabilities > Vulnerability Explorer.

Roll-ups

The charts at the top of the Vulnerability Explorer helps you answer two questions:

1. How many CVEs do you have?

For each object type (image, host, function), the chart reports a count of vulnerabilities in each object class in your environment as a function of time. Consider an environment that has just a single image, where that image has three vulnerabilities: one high, one medium, and one low. Then at time=today on the Images vulnerabilities chart, you could read the following values:

- Critical - 0
- High - 1
- Medium - 1
- Low - 1

2. How many images do you need to fix?
For each object type (image, host, function), the chart reports a count of the highest severity vulnerability in each object class in your environment as a function of time. Consider an environment that has just a single image, where that image has three vulnerabilities: one high, one medium, and one low. Then at time=today on the Impacted images chart, you could read the following values:

- Critical: 0
- High: 1
- Medium: 0
- Low: 0

Let’s look at it another way with a different set of data. Assume the reading at t=today reports the following values, where t is some point on the x-axis of the chart.

- Critical: 1
- High: 1
- Medium: 0
- Low: 2

If your policy calls for addressing all critical vulnerabilities, then the chart tells you that there is precisely one image in your environment that has at least one critical vulnerability. Therefore, your work for today is to fix one image. That image might also have two high vulnerabilities and twenty low vulnerabilities, which you will see when you open the image’s scan report, but this chart is not designed to give you a count of total number of vulnerabilities.

Search tool

The search tool at the top of the page lets you determine if any image or host in your environment is impacted by a specific vulnerability (whether it is in the top ten list or not).

Top ten lists

Vulnerability Explorer gives you a ranked list of the most critical vulnerabilities in your environment based on a scoring system. There are separate top ten lists for the container images, hosts, and functions in your environment.

The top ten table is driven by a risk score. The most important factor in the risk score is the vulnerability's severity. But additional factors are taken into account, such as:

- Is a fix available from the vendor?
- Is the container exposed to the Internet?
- Are ingress ports open?
- Is the container privileged?
- Is an exploit available?

The underlying goal of the risk score is to make it actionable (should you address the vulnerability, and with what urgency). Factors that contribute to the risk score are shown in the Risk Factors column.

Running containers can introduce additional environmental factors that increase the calculated score for a vulnerability. For example, when the container runs as root, it could exacerbate the problem. A list of aggravating container traits are listed in the detailed information dialog when you click on a row in the top ten table.
The risk score and risk factors listed in the CVE details dialog are the ones of the image, host, or function with the highest risk score.

Risk factors

Risk factors are combined to determine a vulnerability’s risk score. Vulnerabilities with the highest risk scores are surfaced in the top ten lists.

Risk factors can also be used to prioritize individual vulnerabilities for mitigation. For example, if your cluster runs containers from disparate business groups, a major concern might be container breakouts. DoS vulnerabilities would likely be much less important than remote code execution vulnerabilities, particularly if exploit code were available, you were running as root, and you didn’t have AppArmor or SELinux applied.

To filter vulnerabilities based on risk factors: open the image, host, or function scan report; open the Vulnerabilities tab; and select one or more risk factors.
Prisma Cloud supports the following risk factors:

- **[Critical | High | Medium] severity** — Vulnerability severity.
- **Has fix** — Fix is available from the distro, vendor, or package maintainer.
- **Remote execution** — Vulnerability can be exploited to run arbitrary code.
- **DoS** — Component is vulnerable to denial of service attacks, such as buffer overflow attacks, ICMP floods, and so on.
- **Recent vulnerability** — Vulnerability was reported in the current or previous year.
- **Exploit exists** — Code and procedures to exploit the vulnerability are publicly available.
- **Attack complexity: low** — Vulnerability is easily exploited.
- **Attack vector: network** — Vulnerability is remotely exploitable. The vulnerable component is bound to the network, and the attacker's path is through the network.
- **Reachable from the internet** — Vulnerability exists in a container exposed to the internet.
- **Listening ports** — Vulnerability exists in a container that is listening on network ports.
- **Container is running as root** — Vulnerability exists in a container running with elevated privileges.
- **No mandatory security profile applied** — Vulnerability exists in a container running with no security profile.
- **Running as privileged container** — Vulnerability exists in a container running with --privileged flag.
• **Package in use** – Vulnerability exists in a component that is actually running. For example, if Redis is running in a container or on a host as a service, then all the following (hypothetical) vulnerabilities could be surfaced by filtering on this risk factor:

<table>
<thead>
<tr>
<th>Component</th>
<th>CVE IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>redis (main process)</td>
<td>CVE-XXX, CVE-XXX</td>
</tr>
<tr>
<td>- libssl (dependent package)</td>
<td>CVE-XXX, CVE-XXX</td>
</tr>
<tr>
<td>- libzip (dependent package)</td>
<td>CVE-XXX, CVE-XXX</td>
</tr>
</tbody>
</table>

**Risk trees**

Risk trees list all the images, namespaces, containers, and hosts that are vulnerable to a specific CVE. Risk trees are useful because they show you how you are exposed to a given vulnerability. Because Prisma Cloud already knows which vulnerabilities impact which packages, which packages are in which images, which containers are derived from which images, which containers run in which namespaces, and which hosts run which containers, we can show you the full scope of your exposure to a vulnerability across all objects in your environment.

For each top ten vulnerability, Prisma Cloud shows you a vulnerability risk tree. To see the vulnerability tree for a given CVE, click on the corresponding row in the top ten table to open a detailed CVE assessment dialog.

You can also generate a risk tree for any arbitrary CVE in your environment by entering the CVE ID into the search bar at the top of the page, then clicking on the result in the table to open a detailed CVE assessment dialog.
Recalculating statistics

Statistical data is calculated every 24 hours. You can force Console to recalculate the statistics for the current day with the current data by clicking the Refresh button in the top left of Vulnerability Explorer. The Refresh button has a red marker when new data is available to be crunched.
Vulnerability management rules

Vulnerability policies are composed of discrete rules. Rules declare the actions to take when vulnerabilities are found in the resources in your environment. They also control the data surfaced in Prisma Cloud Console, including scan reports and Radar visualizations.

Rules let you target segments of your environment and specify actions to take when vulnerabilities of a given type are found. For example:

*Block images with critical severity vulnerabilities from being deployed to prod environment hosts*

There are separate vulnerability policies for containers, hosts, and serverless functions. Host and serverless rules offer a subset of the capabilities of container rules, the big difference being that container rules support blocking.

Creating vulnerability rules

Prisma Cloud ships with a simple default vulnerability policy for containers, hosts, and serverless functions. These policies have a rule named *Default - alert all components*, which sets the alert threshold to low. With this rule, all vulnerabilities in images, hosts, and functions are reported.

As you build out your policy, you'll create rules that filter out insignificant information, such as low severity vulnerabilities, and surface vital information, such as critical vulnerabilities.

*Rule order* is important. Prisma Cloud evaluates the rule list from top to bottom until it finds a match based on the object filters.

By default, Prisma Cloud optimizes resource usage by only scanning images with running containers. Therefore, you might not see a scan report for an image when it's first pulled into your environment unless it's been run. To scan all images on the hosts in your environment, go to *Manage > System > Scan*, set *Only scan images with running containers* to *Off*, and click *Save*.

To create a vulnerability rule:

**STEP 1 |** Open Console.

**STEP 2 |** Go to *Defend > Vulnerabilities > [Images | Hosts | Functions]*.

**STEP 3 |** Click *Add rule*.

**STEP 4 |** Enter a rule name and configure the rule. Configuration options are discussed in the following sections.

**STEP 5 |** Click *Save*.

**STEP 6 |** View the impact of your rule. Go to *Monitor > Vulnerabilities* to view the scan reports.

Severity-based actions

Vulnerability rules let you specify trigger thresholds for alerting and blocking. Alert and block actions let you establish quality gates in the CD segment of your continuous integration (CI) continuous deployment (CD) pipeline.

Alert and block thresholds can be set to different values. The block threshold, however, must always be equal or greater than the alert threshold.
Setting the alert threshold to off allows all vulnerabilities for the resources in scope (as defined by your filters). Practically, resource nodes in Radar turn green (no issues to report), and scan reports are empty (no issues to report).

When you create a blocking rule, Defender automatically installs itself as the final arbiter of all container lifecycle commands. This way, Defender can assess a Docker command, your current policy, and the status of an image before either forwarding the command to runC for execution, or blocking it all together.

Scope

Scope filters let you target specific resources in your environment. By default, filters are set to wildcards, which applies the rule globally. For more information about supported pattern matching syntax, see Rule ordering and pattern matching.

Vendor fixes

Rules can be applied conditionally depending on whether vendor fixes are available. For example, you could tune your policy to block the deployment of containers with a critical vulnerability *only if* the vulnerable package has an update that resolves the issue. Otherwise, the deployment would be allowed to proceed.

Some vulnerabilities have a vendor status of "Will not fix". This status is applied when vendors don’t intend to resolve a vulnerability because it poses no significant risk to your environment.

Rule exceptions

You can configure Prisma Cloud to:

- Alert or block on specific CVEs or tags (deny).
- Ignore specific CVEs or tags (allow).

Under Advanced settings, create a list of vulnerabilities and tags, and specify how the scanner should handle them. Leaving the expiration date blank enforces the action until the CVE or tag is removed from the list. If you set an expiration date, and the current date is later than the expiration date, the scanner ignores the directive. The CVE or tag remains in the list even if its expired. It must be manually removed. Notice that for tag exceptions, in case of a conflict (a vulnerability with two tags or more that have different actions in the rule exceptions) there’s no guarantee what action will apply.
Prisma Cloud lets you create rules that block access to resources or block the deployment of vulnerable containers. For example, you might create a rule that blocks the deployment of any image that has critical severity vulnerabilities. By default, when you try to run a vulnerable image, Prisma Cloud returns a terse response:

```
$ docker run -it ubuntu:14.04 sh
docker: Error response from daemon: [Prisma Cloud] Image operation blocked by policy: (sdf), has 44 vulnerabilities, [low:25 medium:19].
```

To help the operator better understand how to handle a blocked action, you can enhance Prisma Cloud's default response by:

- Appending a custom message to the default message. For example, you could tell operators where to go to open a ticket.
- Configuring Prisma Cloud to return an itemized list of compliance issues rather than just a summary. This way, the operator does not need to contact the security team to determine which issues are preventing deployment. They are explicitly listed in the response.

When terminal output verbosity is set to **Detailed**, the response looks as follows:

```
$ docker run -it ubuntu:14.04 sh
docker: Error response from daemon: [Prisma Cloud] Image operation blocked by policy: (sdf), has 44 vulnerabilities, [low:25 medium:19].

<table>
<thead>
<tr>
<th>Image</th>
<th>ID</th>
<th>CVE</th>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubuntu:14.04</td>
<td>4333f1</td>
<td>CVE-2017-2518</td>
<td>sqlite3</td>
<td>3.8.2-1ubuntu2.1</td>
</tr>
<tr>
<td>deferred</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ubuntu:14.04</td>
<td>4333f1</td>
<td>CVE-2017-6512</td>
<td>perl</td>
<td>5.18.2-2ubuntu1.1</td>
</tr>
<tr>
<td>needed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Grace period

Grace periods temporarily override the blocking action of a rule when new vulnerabilities are found. Grace periods give you time to address a vulnerability without compromising the availability of your app.

When grace periods are configured, alerts trigger as normal, notifying you that the vulnerability exists in your environment. The block action is suppressed for the number of days specified, giving you time to mitigate the vulnerability.

The start time for the grace period is the date the vulnerability report was published. The end time is the publish date plus the number of days configured for the grace period.

Blocking based on vulnerability severity

This example shows you how to create and test a rule that blocks the deployment of images with critical or high severity vulnerabilities.

**STEP 1 |** In Console, go to **Defend > Vulnerabilities > Images**.

**STEP 2 |** Click Add rule.

1. Enter a rule name, such as my-rule.
2. In the Severity based actions table, set both the Alert threshold and Block threshold to High.
3. Target the rule to a very specific image. In the Images filter, delete the wildcard, and enter nginx*.
4. Click Save.

**STEP 3 |** Validate your policy by pulling down the nginx image and running it.

1. SSH to a host protected by Defender.
2. Pull the nginx:1.14 image.

   $ docker pull nginx:1.14
3. Run the nginx image.

```bash
$ docker run -it nginx:1.14 /bin/sh
```

docker: Error response from daemon: oci runtime error: [Prisma Cloud] Image operation blocked by policy: my-rule, has 7 vulnerabilities, [high:7].


By default, Prisma Cloud optimizes resource usage by only scanning images with running containers. Therefore, you won’t see a scan report for nginx until it’s run.

## Blocking specific CVEs

This example shows you how to create and test a rule that blocks images with a specific CVE.

**STEP 1 |** In Console, go to Defend > Vulnerabilities > Images.

**STEP 2 |** Click Add rule.

1. Enter a rule name, such as `my-rule2`.
2. Click Advanced settings.
3. In Exceptions, click Add Exception.
You can find specific CVE IDs in the image scan reports. Go to Monitor > Vulnerabilities > Images, select an image, then click Show details in each row.

5. In Effect, select Block.
6. Click Add.
7. Click Save.

STEP 3 | Try running an image with the CVE that you've explicitly denied.

```bash
$ docker run -it imiell/bad-dockerfile:latest /bin/sh
docker: Error response from daemon: oci runtime error: [Prisma Cloud] Image operation blocked by policy: my-rule2, has specific CVE CVE-2018-8014
```

Ignoring specific CVEs

Follow the same procedure as above, but set the action to Ignore instead of Block. This will allow any CVE ID that you’ve defined in the rule, and lets you run images containing those CVEs in your environment.
Search CVEs

You can determine if Prisma Cloud offers coverage for a specific CVE by using the search interface in Console.

The CVE ID syntax is:

```
CVE--YYYY--NNNN
```

Where:
- **CVE** -- CVE-ID prefix.
- **YYYY** -- Calendar year.
- **NNNN** -- Numeric digits. This field has a variable length, but the minimum length is four digits.

Searching for a specific CVE

To search for a specific vulnerability:

**STEP 1** | Open Console, then go to **Monitor > Vulnerabilities > CVE Viewer**.

**STEP 2** | In the query text box in the top right, enter a CVE ID.

For example, enter **CVE-2015-1345**.

If Prisma Cloud has coverage for the queried vulnerability, details are listed in the results table.
Allow a CVE

Allowing CVEs is done directly as a policy.
Scan reports

Prisma Cloud scans all Docker images on all hosts that run Defender. After Defender is installed, it automatically starts scanning images on the host. After the initial scan, subsequent scans are triggered:

- Periodically, according to the scan interval configured in Console. By default, images are scanned every 24 hours.
- When new images are created, pushed, or pulled onto the host.
- When images change.
- When scans are forced with the Scan button in Console.

Defender scans Docker images for:

- Published Common Vulnerabilities and Exposures (CVEs).
- Vulnerabilities from misconfigurations.
- Malware
- Zero day vulnerabilities
- Compliance issues
- Secrets

The Prisma Cloud Intelligence Stream keeps Console up to date with the latest vulnerabilities. The data in this feed is distributed to your Defenders, and employed in subsequent scans.

Through Console, Defender can be extended to scan images for custom components. For example, you can configure Defender to scan for an internally developed library named libexample.so, and set a policy to block a container from running if version 1.9.9 or earlier at installed. For more information, see Scanning custom components.

View image scan reports

Review the health of all images in your environment.

Sorting the table on vulnerability severity as based on data from the last scan. If you update your vulnerability policy with a different alert threshold, recan your images if you want to be able to sort based on your new settings.

STEP 1 | Open Console, then go to Monitor > Vulnerabilities > Images.

The table summarizes the state of each image in your environment.

All vulnerabilities identified in the last image scan can be exported to a CSV file by clicking the CSV button in the top left of the page.
### Monitor / Vulnerabilities

<table>
<thead>
<tr>
<th>Registry</th>
<th>Repository</th>
<th>Tag</th>
<th>Hosts</th>
<th>Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>neilcar/struts2_demo</td>
<td>latest</td>
<td>demo-htmicinco-lab-t...</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>weaveworksdemos/queue...</td>
<td>0.3.1</td>
<td>demo-node-htmicinco...</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>weaveworksdemos/shipp...</td>
<td>0.4.8</td>
<td>demo-node-htmicinco...</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>weaveworksdemos/orders</td>
<td>0.4.7</td>
<td>demo-htmicinco-lab-t...</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>weaveworksdemos/carts</td>
<td>0.4.8</td>
<td>demo-node-htmicinco...</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>demo.htmicinco.lab.twistl...</td>
<td></td>
<td>demo-htmicinco-lab-t...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>infoslack/dvwa</td>
<td>latest</td>
<td>demo-htmicinco-lab-t...</td>
<td>499</td>
</tr>
<tr>
<td></td>
<td>infra/my_jenkins</td>
<td></td>
<td>demo-htmicinco-lab-t...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>demo.htmicinco-lab-twistl...</td>
<td></td>
<td>demo-htmicinco-lab-t...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>infra/portal_httpd</td>
<td>latest</td>
<td>demo-htmicinco-lab-t...</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>weaveworksdemos/catal...</td>
<td>0.3.0</td>
<td>demo-node-htmicinco...</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>weaveworksdemos/user...</td>
<td>0.4.0</td>
<td>demo-htmicinco-lab-t...</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>rabbitmq</td>
<td>3.6.8</td>
<td>demo-htmicinco-lab-t...</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>splunk/splunk</td>
<td>latest</td>
<td>demo-htmicinco-lab-t...</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>weaveworksdemos/front...</td>
<td>0.3.12</td>
<td>demo-htmicinco-lab-t...</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>weaveworksdemos/catal...</td>
<td>0.3.5</td>
<td>demo-node-htmicinco...</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>weaveworksdemos/paym...</td>
<td>0.4.3</td>
<td>demo-htmicinco-lab-t...</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>gcr.io</td>
<td></td>
<td>demo-htmicinco-lab-t...</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>google_containers/kube...</td>
<td>v1.9.11</td>
<td>demo-htmicinco-lab-t...</td>
<td>2 hosts</td>
</tr>
<tr>
<td></td>
<td>openebs/jiva</td>
<td>0.6.0</td>
<td>demo-htmicinco-lab-t...</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>registry</td>
<td>2</td>
<td>demo-htmicinco-lab-t...</td>
<td>6</td>
</tr>
</tbody>
</table>

**STEP 2** | Click on an image report to open a detailed report.

**STEP 3** | Click on the **Vulnerabilities** tab to see all CVE issues.
CVE vulnerabilities are accompanied by a brief description. Click Show details for more information, including a link to the report on the National Vulnerability Database.

The Vendor Status column contains terms such as ‘deferred’, ‘fixed in…’, and ‘open’. These strings are imported directly from the vendors’ CVE databases. They are not Prisma Cloud-specific.

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Highest Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>411</td>
<td>Binary</td>
<td>critical</td>
<td>node version 0.10.41 has 18 vulnerabilities. Show details</td>
</tr>
<tr>
<td>49</td>
<td>nodejs</td>
<td>critical</td>
<td>tar version 1.0.1 has 1 vulnerability. Hide details</td>
</tr>
</tbody>
</table>

### Tagging vulnerabilities

To help you manage and fix the vulnerabilities in your environment, you can set tags on each vulnerability. Setting a tag on a vulnerability will apply to the CVE ID and package across the product. The list of available tags is defined under Manage > Collections and Tags > Tags. See Configure Tags. To add a tag to a vulnerability, click on the Add tags to CVE action in the Tags column.

<table>
<thead>
<tr>
<th>Type</th>
<th>Highest Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>critical</td>
<td>python3.7 (used in libpython3.7-minimal, libpython3.7-stdlib, python3.7-minimal, python3.7) version 3.7.3-2 has 7 vulnerabilities.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Severity</th>
<th>Package</th>
<th>CVE</th>
<th>Fix Status</th>
<th>Risk Factors</th>
<th>Tags</th>
<th>Description</th>
</tr>
</thead>
</table>
| critical | python3.7 | CVE-2019-10160 | fixed in 3.7.3-2+deb10u1 | 5 | [In progress](#) | Impacted versions: <3.7.3-2+deb10u1  
Discovered: 5 days ago  
Published: > 8 months ago  
A security regression of CVE-2019-9636 was discovered in python since commit d537a00f9767ef024f26246899728f01161ec3 affecting versions 2.7, 3.5, 3.6, 3.7 and from v3.8.0a4 through v3.8.0b1, which still allows an attacker to exploit CVE-2019-9636 by... |

Setting tags on CVEs is allowed for all user roles that can view the scan results. Setting tags that have policy rules exceptions is allowed only for Admin, Operator, and Vulnerability Manager user roles.
You can also add comments to each tag you apply to the CVE, for example, to explain the reason this tag was added. Do it by clicking the comment icon on the left side of the tag.

<table>
<thead>
<tr>
<th>Type</th>
<th>Highest Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>critical</td>
<td>python3.7 (used in libpython3.7-minimal, libpython3.7-standard, python3.7) version 3.7.3-2 has 7 vulnerabilities.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Severity</th>
<th>Package</th>
<th>CVE</th>
<th>Fix Status</th>
<th>Risk Rating</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>critical</td>
<td>python3.7</td>
<td>CVE-2019-10160</td>
<td>fixed in 3.7.3-2+deb10u1</td>
<td>5</td>
<td>In progress</td>
</tr>
</tbody>
</table>

Impacted versions: <3.7.3-2+deb10u1
Discovered: 5 days ago
Published: >8 months ago
A security regression of CVE-2019-9636 was discovered in python since commit
d537ab0f9767ef024f26246899728f0116b1ec3 affecting versions 2.7, 3.5, 3.6, 3.7 and from v3.8.0a4 through v3.8.0b1, which still allows an attacker to exploit CVE-2019-9636 by

By default, all vulnerabilities, according to your policy, are listed. However, you can also examine vulnerabilities only with specific tags. Use the drop-down list to filter by tags.

Per-layer vulnerability analysis

To make it easier to understand how images are constructed and what components have vulnerabilities, Prisma Cloud correlates vulnerabilities to layers. This tool helps you assess how vulnerabilities were introduced into an image, and pick a starting point for remediation.

To see the layer analysis, click on an image to open the scan report, then click the Layers tab.
### RHEL images

The Prisma Cloud layers tool shows the instructions used to create each layer in an image. RHEL images, however, don’t contain the necessary metadata, so the Prisma Cloud layers tool shows an empty black box.
To validate the required metadata is absent, run `docker history IMAGE-ID` on a non-RHEL image. The `CREATED BY` column is fully populated.
Next, run `docker history IMAGE-ID` on a RHEL image. Notice that the `CREATED BY` column is empty.

### Packages in use

Prisma Cloud uses risk scores to calculate the severity of vulnerabilities in your environment. One of the factors in the risk score is called "Package in use", which indicates a package is utilized by running software.

Scan reports have a **Package info** tab, which lists all the packages installed in an image or host. It also shows all active packages, which are packages used by running software.

To see these active packages, open a scan report, open the **Package info** tab, and look at the **Binaries** column (see the **App** column in host scan reports). This column shows what's actually running in the container. For example, the fluent/fluentd:latest container in the following screenshot runs `/usr/bin/ruby`.

One of the packages utilized by the Ruby runtime is the bigdecimal gem. If you were prioritizing mitigation work, and there were a severe vulnerability in bigdecimal, bigdecimal would be a good candidate to address first.
Vulnerability management

Per-finding timestamps

Prisma Cloud’s image scan reports show the following per-vulnerability timestamps:

- Age of the vulnerability based on the discovery date. This is the first date that the Prisma Cloud scanner found the vulnerability.
- Age of the vulnerability based on its published date. This represents the date the vulnerability was announced to the world.

Host scan reports and registry scan reports show the published date only.
Timestamps are per-image, per-vulnerability. For example, if CVE-2019-1234 was found in image foo/foo:3.1 last week and image bar/bar:7.8 is created from foo/foo:3.1 today, then the scan results for foo show the discovery date for CVE-2019-1234 to be last week and for bar it shows today.

Timestamped findings are useful when you have time-based SLAs for remediating vulnerabilities (e.g. all critical CVEs must be fixed within 30 days). Per-finding timestamp data makes it possible to track compliance with these SLAs.

Host and VM image scanning

Prisma Cloud also scans your hosts and VM images for vulnerabilities. To see the scan report for your hosts and VM images, go to Monitor > Vulnerabilities > Hosts.

By default, all vulnerable packages, according to your policy, are listed. However, you can also examine vulnerabilities specific to an app (systemd service). Use the drop-down list to select an app. Clear the selection to see all vulnerabilities for a host/VM image.
The **Package Info** tab lists all packages installed on the host/VM image. If a package has a component utilized by a running app, the affected running apps are listed in the **Apps** column.

Prisma Cloud also collects and displays package license details. License information is available at all places where package details are displayed, such as **Monitor > Vulnerabilities > Images** (under the **Package Info** tab), **Monitor > Vulnerabilities > Hosts** and **Monitor > Vulnerabilities > Registry**, as well as the corresponding API endpoints.
Licensing compliance is currently supported only for viewing purposes and cannot be included in policies for alert/block capabilities.

Scan status

The initial scan can take substantial time when you have a large number of images. Subsequent scans are much faster.

To see the status of the image scans, go to Monitor > Vulnerabilities > Images.

Each row in the table represents an image in your environment.

If an image is being scanned, a progress bar shows the status of the scan. If there is no progress bar, the scan has completed.

Package types

Prisma Cloud uses compliance identification numbers to designate the package type when reporting vulnerabilities in images. Compliance IDs can be found in the CSV export files and API responses.

To download image reports in CSV format, go to Monitor > Vulnerabilities > Images, and click the CSV button at the top of the table. The Compliance ID, Type, and Packages fields report the package ID, package type, and package name respectively. The API output reports compliance IDs only.

The following table shows how compliance IDs map to package type.
<table>
<thead>
<tr>
<th>Compliance ID number</th>
<th>Package type</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>Operating system/distro packages</td>
</tr>
<tr>
<td>47</td>
<td>JAR files</td>
</tr>
<tr>
<td>48</td>
<td>Gem files</td>
</tr>
<tr>
<td>49</td>
<td>Node.js</td>
</tr>
<tr>
<td>410</td>
<td>Python</td>
</tr>
<tr>
<td>411</td>
<td>ie. MySgl</td>
</tr>
<tr>
<td>412</td>
<td>Custom (set by customer)</td>
</tr>
<tr>
<td>415</td>
<td>Nuget</td>
</tr>
</tbody>
</table>
Customize image scanning

You can customize how Prisma Cloud scans images and reports data.

Configuring the severity of reported CVEs

By default, Prisma Cloud reports all vulnerabilities. Setting the minimum reported severity lets you clean up the reported vulnerabilities to an actionable set.

To configure a minimum severity, install a new vulnerability rule, which overrides the default rule. Note that Prisma Cloud maps the Common Vulnerability Scoring System (CVSS) to a grading system that ranges from Low to Critical.

STEP 1 | Open Console, and go to Defend > Vulnerabilities > Policy.

STEP 2 | Click Add rule.

STEP 3 | Give your rule a name.

STEP 4 | In the table of Severity based actions, set the Severity in each row to an appropriate level. For example, if you want to concentrate on just the most severe issues, set every row to Critical.

STEP 5 | Click Save.

STEP 6 | View the scan reports for all the entities in your system.

Go to Monitor > Vulnerabilities. All reported vulnerabilities match or exceed the severity setting in your custom rule.

Scanning custom components

Prisma Cloud lets you scan for insecure versions of proprietary software components.

First, augment Prisma Cloud’s Intelligence Stream with your own custom data that specifies a package type, name, and version number. Then configure Prisma Cloud to take action (alert, block) when the scanner finds this package in an image. By default, Prisma Cloud raises an alert when it detects a vulnerability in a custom component.

Prisma Cloud supports the following package types:
- Distro packages (deb, rpm).
- Binaries.
- Nodejs packages.
- Python packages.
- Ruby gems.
- Java artifacts (JAR files).

For cases where Prisma Cloud does not offer built-in support for a package type, you can specify an MD5 hash for the file.

Defining a custom vulnerability

Define a custom vulnerability.

STEP 1 | Open Console.
**STEP 2** | Go to Manage > System > Custom Feeds.

**STEP 3** | Click on Custom Vulnerabilities.

**STEP 4** | Click Add.

1. Enter a name for your vulnerability.
2. From the drop-down list, select a package type.
   
   For Debian packages, RPM packages, and shared libraries, select **package**.

   If your package type is not supported, select **binary**.
3. Enter the name of your package/binary.
4. Specify the range of package versions for which your rule applies.

   The following formats can be used to specify versions:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specific version</td>
<td>Enter a single multi-dot number.</td>
<td>1.1</td>
</tr>
<tr>
<td>Range of versions:</td>
<td>Min and max are known.</td>
<td>5.4-5.5</td>
</tr>
<tr>
<td>Only min version is known.</td>
<td>Enter two multi-dot numbers, separated by a dash.</td>
<td>0.22.4.1-*</td>
</tr>
<tr>
<td>Only max version is known.</td>
<td>Specify a multi-dot number for the minimum version, followed by a dash, then a wild card.</td>
<td>0.22.4.1-*</td>
</tr>
<tr>
<td></td>
<td>Specify a wild card (*) for the minimum version, followed by a dash, then a multi-dot number for the maximum version.</td>
<td>*-0.22.4.1</td>
</tr>
</tbody>
</table>

   If package type is set to binary, the version fields are not visible. Instead, enter the MD5 hash for your file or binary.

**STEP 5** | Click **Save**.

Your custom vulnerability is now available to the scanner.

By default, an alert is logged if an image scan detects a component that you have designated as vulnerable. To see the default rule, go to Defend > Vulnerabilities > Images, and click on the Default - **alert all components** rule. To change the default rule, select a different Alert or block threshold. To take a different action, create a new vulnerability rule.
Configure registry scans

Prisma Cloud can scan container images in both public and private repositories on both public and private registries.

The registry is a system for storing and distributing container images. The most well-known public registry is Docker Hub, although there are also registries from Amazon, Google, and others. Organizations can also set up their own internal private registries. Prisma Cloud can scan container images on all of these types of registries.

After repository scanning is configured, Prisma Cloud automatically scans images for vulnerabilities. Periodic scans are run at an interval specified in Configure > System > Scan (by default, once every 24 hours).

Deployment patterns

Registry scanning is handled by Defender. When you configure Prisma Cloud to scan a registry, you can either:

- Let Prisma Cloud automatically distribute the scan job across a pool of available Defenders, or
- Explicitly specify the Defender to do the job.

Any Container Defender running on a host with the Docker Engine container runtime or container runtime interface (CRI) can scan a registry, and any number of them can simultaneously operate as registry scanners. This gives you a lot of options when you're trying to determine how to cover disparate environments.

In general, you should configure Prisma Cloud to automatically distribute scan jobs because it reduces operational complexity and improves resiliency. At scan-time, Prisma Cloud enumerates the available Defenders, manages the resource pool, and handles issues such as restarting partially completed jobs. If you explicitly select a specific Defender to handle scanning, the host where Defender runs is a single point of failure. If the host fails, or gets destroyed, you have to manually reconfigure your scan configuration with a different Defender.

When selecting the automatic algorithm for registry scanning, you can specify the number of Defenders in the resource pool. For large registries or aggressive scan intervals, increase the number of Defenders to improve throughput and reduce scan time.

Registry scanning is scoped by OS type. Windows Defenders can only scan Windows images, and Linux Defenders can only scan Linux images.

If you remove an image from the registry, or the registry becomes unavailable, Prisma Cloud maintains the scan results for 30 days. After 30 days, the scan results are purged.

Large-scale registries

When you have very large registries, you must optimize your scan configuration to maximize throughput and minimize scan time. The first obvious optimization is to set the scan algorithm to automatic and increase the number of scanners. The second optimization is to specify a version matching pattern in your registry scan configuration.

Optimizing registry scans with version pattern matching is only necessary for very large registries with tens of thousands of repositories and millions of images.
The scanner makes many API calls to the registry to retrieve metadata for the registry, repos, and images. All metadata must be collected, collated, and sorted before scanning can start. Consider the normal flow for collecting metadata:

<table>
<thead>
<tr>
<th>Get a list of all repos in the registry</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each repo:</td>
</tr>
<tr>
<td>Get a list of all image tags</td>
</tr>
<tr>
<td>For each image tag:</td>
</tr>
<tr>
<td>Get the image manifest (which contains the last modified date)</td>
</tr>
<tr>
<td>Sort, Cap, Scan</td>
</tr>
</tbody>
</table>

After fetching all metadata, the scanner sorts the images by last modified date, and caps the list if a cap value is specified in the scan configuration. The default cap value is 5. With a cap of 5, the scanner fetches the five most recently modified images from the registry for scanning.

If you specify a version matching pattern, the scanner looks to the image tag for sort order. Without a version matching pattern, the sort order is last modified date. With a version matching pattern, you customize how the scanner interprets image tags for sorting. For example, if you utilize semantic versioning in your image names, you could specify the following version pattern:

`*-%d.%d.%d`

The scanner parses each image tag, extracts the pattern from the tag, and splits it into its constituent parts. After all tags are parsed, they are sorted, and capped according to your configuration. The optimized flow for collecting metadata eliminates the inner loop, substantially reducing the number of requests to the registry so scanning can start sooner.

<table>
<thead>
<tr>
<th>Get a list of all repos in the registry</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each repo:</td>
</tr>
<tr>
<td>Get a list of all images tags</td>
</tr>
<tr>
<td>Sort, Cap, Scan</td>
</tr>
</tbody>
</table>

If your repo had three images, and your scan configuration specified a cap of 2 and version pattern of `*-%d.%d.%d`, you'd get the following result:

```
myimage-3.0.0 <<<--- Scan
myimage-2.0.1 <<<--- Scan
myimage-2.0.0 (Not scanned)
```

### Registry scan settings

Each rule has the following parameters, although the parameters can vary according to registry type. For step-by-step instructions for a registry from a specific vendor, see the appropriate registry-specific guide.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Specify the type of registry to scan.</td>
</tr>
<tr>
<td></td>
<td>• If you do not find your vendor's registry in the drop-down list, try Docker Registry v2. Most vendors comply with the Docker Registry version 2 API.</td>
</tr>
<tr>
<td>Registry</td>
<td>Specify the URL for the registry.</td>
</tr>
<tr>
<td></td>
<td>Docker Hub: leave this field blank.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Harbor</strong></td>
<td>Specify the FQDN of your Harbor registry (https://).</td>
</tr>
<tr>
<td>Repository name</td>
<td>Specify the repository to scan. This field supports pattern matching. To scan all repositories, simply leave this field blank or enter a wildcard (*).</td>
</tr>
<tr>
<td><strong>Docker Hub</strong></td>
<td>To specify an official Docker repository, enter library/, followed by the short string used to designate the repo. For example, to scan the images in the official Alpine Linux repository, enter library/alpine.</td>
</tr>
<tr>
<td></td>
<td>To specify non-official repositories, enter the user name or organization name, followed by a slash, followed by the name of the repo. For example, to specify the alpine repository in onescience’s account, enter onescience/alpine.</td>
</tr>
<tr>
<td></td>
<td>To scan all repos from a user or organization, simply enter the user or organization name, followed by a wildcard (<em>). For example, to scan all repos created by onescience, enter onescience</em>.</td>
</tr>
<tr>
<td><strong>Google Cloud Platform Container Registry</strong></td>
<td>Enter your project ID and image name in the following format: project-id/image-name.</td>
</tr>
<tr>
<td><strong>Harbor</strong></td>
<td>Enter the name of the repository, followed by a wildcard (<em>). For example, to scan repository library, enter library</em>.</td>
</tr>
<tr>
<td><strong>Any Docker Registry version 2 API compliant registry</strong></td>
<td>Docker Hub, Docker Registry, and Alibaba Container Registry all support the Docker Registry version 2 API.</td>
</tr>
<tr>
<td><strong>Tag</strong></td>
<td>Specify an image tag. Leave this field blank to scan all tags (limited by the value in Cap).</td>
</tr>
<tr>
<td><strong>Credentials</strong></td>
<td>Specify the credentials required to access the registry. If the credentials have already been created in the Prisma Cloud credential store, select it. If not, click Add New.</td>
</tr>
<tr>
<td><strong>Public repositories on public registries (such as Docker Hub)</strong></td>
<td>Leave this field blank. No credentials are required.</td>
</tr>
<tr>
<td><strong>AWS EC2 Container Registry</strong></td>
<td>Use the IAM access keys for authentication. For more information, see Amazon EC2 Container Registry (ECR).</td>
</tr>
<tr>
<td><strong>Google Container Registry</strong></td>
<td>Use the service account and JSON token. For more information, Google Container Registry (GCR).</td>
</tr>
<tr>
<td><strong>Harbor Registry</strong></td>
<td>Create a Basic authentication credential. Credentials for Harbor can be a Limited Guest.</td>
</tr>
<tr>
<td><strong>Registries that support token authentication (e.g. Quay, GitLab)</strong></td>
<td>Create a Basic authentication credential. Username is the name of the token and the token value is entered into the password field.</td>
</tr>
<tr>
<td><strong>OS Type</strong></td>
<td>Specify whether the image is built on a Windows or Linux base OS.</td>
</tr>
<tr>
<td><strong>Scanner</strong></td>
<td>Specify the Defender to scan this registry.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Automatic</strong> <em>(default, recommended)</em>: Prisma Cloud Console automatically schedules the scan job across a pool of available Defenders.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Explicitly selected Defender</strong>: Prisma Cloud Console sends the scan job to a specific Defender.</td>
</tr>
<tr>
<td></td>
<td>Only Linux Defenders can scan Linux container images, and only Windows Defenders can scan Windows container images.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Number of scanners</td>
<td>Number of Defenders across which the scan job can be distributed. Increase the number of Defenders to increase throughput and reduce scan time.</td>
</tr>
<tr>
<td>Cap</td>
<td>Specify the maximum number of images to scan in the given repository, sorted according to last modified date. That is, the most recently modified image in the repository is scanned first, followed by the image next most recently modified, and so on. The Docker Registry API does not support directly querying for the most recently updated images. To handle your CAP setting, Prisma Cloud first polls the registry for all tags and manifests in the given repository to discover the last updated dates. This is a low overhead operation because images do not need to be downloaded. Prisma Cloud then sorts the results by date, and then scans the most recently updated images up to the limit specified by CAP. Even when CAP is set to a low number, you might still notice the Prisma Cloud UI polling the registry for data about the images in the repository. To scan all images in a repository, set CAP to 0.</td>
</tr>
<tr>
<td>Version matching</td>
<td>Customize sort order by values in the image tag. Specify a pattern from which a version or date can be extracted from the image tag. There are two use cases for specifying version matching patterns:</td>
</tr>
<tr>
<td>matching pattern</td>
<td>• You want to reduce the total time it takes to complete the scan for very large registries. Rather than fetching the metadata from the registry required to sort images, you specify how the scanner can extract the metadata directly from the image tag.</td>
</tr>
<tr>
<td></td>
<td>• You want to order and cap the images to be scanned by some value other than last modified date.</td>
</tr>
<tr>
<td></td>
<td>Specify patterns with strings, wildcards, timedate elements, and integers.</td>
</tr>
<tr>
<td></td>
<td>• %d - version number</td>
</tr>
<tr>
<td></td>
<td>• %Y - 4 digit year</td>
</tr>
<tr>
<td></td>
<td>• %M - 2 digit month</td>
</tr>
<tr>
<td></td>
<td>• %D - 2 digit day</td>
</tr>
<tr>
<td></td>
<td>• %H - 2 digit hour</td>
</tr>
<tr>
<td></td>
<td>• %m - 2 digit minute</td>
</tr>
<tr>
<td></td>
<td>• %s - 2 digit second</td>
</tr>
<tr>
<td></td>
<td>For image tags that match the pattern, the tag is split into its constituent parts. After all image tags are parsed, they're ordered and capped according to the value set in Cap. Ordering is best-effort. Tags that don't conform to the pattern are ignored. If both date and version are specified in your pattern, date takes precedence. If the version matching pattern is left unspecified, Prisma Cloud orders images by last modified date.</td>
</tr>
</tbody>
</table>

Configure Prisma Cloud to scan a registry

To scan images in a registry, create a new registry scan rule.

**Prerequisites:** You have deployed at least one Defender in your environment.

**STEP 1** | Open Console.
STEP 2 | Go to Defend > Vulnerabilities > Registry.

STEP 3 | Click Add registry settings.

Additional scan settings

Additional scan settings can be found under Manage > System > Scan, where you can set the registry scan interval.

The Manage > System > Scan page has an option called Only scan images with running containers. This option does NOT apply to registry scanning; all images targeted by your registry scanning rule will be scanned regardless of how Only scan images with running containers is set.

CRI/containerd-only environments

Prisma Cloud fully supports scanning CRI/containerd-only environments.

Defenders that run on DC/OS cannot scan registries.
Registry scanning

Configure Prisma Cloud to scan your registries.

Scan images in Alibaba Cloud Container Registry

Configure Prisma Cloud to scan your Alibaba Cloud Container Registry. First, create a service account, and then specify the scan parameters.

Create a service account

Create a service account so Prisma Cloud can access your registry. Prisma Cloud needs the AliyunContainerRegistryReadOnly permission policy to query, download, and scan the images in your registry.

STEP 1 | In Alibaba Cloud, create a RAM account.

Go to RAM > Users, and click Create User.

STEP 2 | Click Add Permissions.
STEP 3 | Search for registry, and then select AliyunContainerRegistryReadOnly.

Scan images in Alibaba Cloud Container Registry

To scan a repository in Alibaba Cloud Container Registry, create a new registry scan setting.

Prerequisites:
- You’ve installed a Container Defender somewhere in your environment.
- You’ve already created an Alibaba Cloud Container Registry.
- You have the service account credentials.

STEP 1 | Open Console, and go to Defend > Vulnerabilities > Registry.

STEP 2 | Click Add registry.

STEP 3 | In the Add New Registry Setting Specification dialog, enter the following values:
1. In the Version drop-down list, select Docker Registry v2.
2. In the Registry field, enter the Fully Qualified Domain Name (FQDN) for the registry. For example, registry-intl.cn-hangzhou.aliyuncs.com.
3. In the Repository field, enter the name of the repository to scan. Example: library/alpine.
4. In the Tag field, enter an image tag. Leave this field blank to scan all images, regardless of their tag.
5. In the Credential field, configure how Prisma Cloud authenticates with Alibaba Cloud Container Registry.
   Select a credential from the drop-down list. If there are no credentials in the list, click Add new, and create a Basic authentication credential with the service account username and password.
6. In the OS type field, specify whether the repo holds Linux or Windows images.
7. In Scanner, select Automatic.
   Console automatically selects an available Defender to execute the scan job. Alternatively, you can explicitly select a Defender from the drop-down list. Defenders are listed according to the hosts where they run. For more information, see deployment patterns.
8. In Number of scanners, enter the number of Defenders across which scan jobs can be distributed.
9. In Cap, limit the number of images to scan.

   Set Cap to 5 to scan the five most recent images, or enter another value to increase or decrease the limit. Set Cap to 0 to scan all images.
10. Click Add.

STEP 4 | Click the yellow save button.

STEP 5 | Verify that the images in the repository are being scanned.
1. Go to Monitor > Vulnerabilities > Images > Registries.
A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

2. To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

**Scan images on Amazon EC2 Container Registry (ECR)**

To scan a repository, Prisma Cloud has to authenticate with ECR using either an IAM user (service account) or IAM role. The minimum permissions policy required is **AmazonEC2ContainerRegistryReadOnly**. It is a managed, predefined policy. AWS managed policies grant the minimum set of permissions required for common use cases so you don’t need to spend a lot of time investigating permissions yourself.

The **AmazonEC2ContainerRegistryReadOnly** permissions policy is currently defined as follows:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "ecr:GetAuthorizationToken",
            "ecr:BatchCheckLayerAvailability",
            "ecr:GetDownloadUrlForLayer",
            "ecr:GetRepositoryPolicy",
            "ecr:DescribeRepositories",
            "ecr:ListImages",
            "ecr:DescribeImages",
            "ecr:BatchGetImage"
         ],
         "Resource": "*"
      }
   ]
}
```

**Prerequisites:** You have installed a Defender somewhere in your environment.

**STEP 1 |** Open Console and go to **Defend > Vulnerabilities > Registry**.

**STEP 2 |** Click **Add registry**.

**STEP 3 |** In the dialog, enter the following information:

1. In **Version**, select **Amazon EC2 Container Registry**.
2. In **Registry**, enter the URL for the registry.
3. In **Repository**, enter the name of the repository to scan.
4. In the **Tag** field, enter an image tag. Leave this field blank to scan all tags.
5. Configure how Prisma Cloud authenticates with AWS. You can use an IAM user, IAM role, or AWS STS.

   To authenticate with an IAM role, set **Use IAM role** to **On**.

   To authenticate with an IAM user, click **Add new** in the **Credential** drop-down list, and create a new AWS credential with an access key ID and secret access key.

6. In **OS type**, specify whether the repo holds **Linux** or **Windows** images.
7. In **Scanner**, select **Automatic**.
Console automatically selects an available Defender to execute the scan job. Alternatively, you can explicitly select a Defender from the drop-down list. Defenders are listed according to the hosts where they run. For more information, see deployment patterns.

8. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
9. Set **Cap** to the number of most recent images to scan. Leaving **Cap** set to 5 will scan the 5 most recent images. Setting this field to 0 will scan all images.
10. Click **Add**.

**STEP 4** | Click the yellow save button.

**STEP 5** | Verify that the images in the repository are being scanned.

1. Go to Monitor > Vulnerabilities > Images > Registries.

   A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

2. To get details about the vulnerabilities in an image, click on it.

   To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

**Scan images in Azure Container Registry (ACR)**

To scan a repository in Azure Container Registry (ACR), create a new registry scan setting.

**Prerequisites:** You have installed a Defender somewhere in your environment.

**STEP 1** | Open Console, and go to Defend > Vulnerabilities > Registry.

**STEP 2** | Click **Add registry**.

**STEP 3** | In the **Add New Registry Setting Specification** dialog, enter the following values:

1. In the **Version** drop-down list, select Azure Container Registry.
2. In the **Registry** field, enter the Fully Qualified Domain Name (FQDN) for the registry’s ACR login server.

   The format for the FQDN is `<REGISTRY_NAME>.azurecr.io`, where `<REGISTRY_NAME>` is a unique value specified when the registry was created. Example: `example.azurecr.io`.
3. In the **Repository** field, enter the name of the repository to scan. Example: `library/alpine`.
4. In the **Tag** field, enter an image tag. Leave this field blank to scan all images, regardless of their tag.
5. In the **Credential** field, configure how Prisma Cloud authenticates with ACR.

   Select a credential from the drop-down list. If there are no credentials in the list, click **Add new**, and create an Azure credential.
6. In the **OS type** field, specify whether the repo holds Linux or Windows images.

   Console automatically selects an available Defender to execute the scan job. Alternatively, you can explicitly select a Defender from the drop-down list. Defenders are listed according to the hosts where they run. For more information, see deployment patterns.
8. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
9. In **Cap**, limit the number of images to scan.
Set Cap to 5 to scan the five most recent images, or enter a different value to increase or decrease the limit. Set Cap to 0 to scan all images.

10. Click Add.

**STEP 4** | Click the yellow save button.

**STEP 5** | Verify that the images in the repository are being scanned.

1. Go to **Monitor > Vulnerabilities > Images > Registries**.

   A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

2. To get details about the vulnerabilities in an image, click on it.

   To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

---

**Scan images on Docker Registry v2 (including Docker Hub)**

Most vendors’ registries comply with the Docker Registry version 2 API, including Docker Hub. To scan a Docker Registry v2 repository, create a new registry scan rule.

For Docker Hub repositories:

- To specify an official Docker Hub repository, enter library/, followed by the short string used to designate the repo. For example, to scan the images in the official Alpine Linux repository, enter library/alpine.
- To specify non-official repositories, enter the user name or organization name, followed by a slash, followed by the name of the repo. For example, to specify the alpine repository in onescience’s account, enter onescience/alpine.
- To scan all repos from a user or organization, simply enter the user or organization name, followed by a wildcard (*). For example, to scan all repos created by onescience, enter onescience*.

Scanning insecure registries (http) is not supported.

---

**Prerequisites:** You have installed a Defender somewhere in your environment.

**STEP 1** | Open Console, and then go to **Defend > Vulnerabilities > Registry**.

**STEP 2** | Click **Add registry settings**.

**STEP 3** | In the dialog, enter the following information:

1. In the **Version** drop-down list, select **Docker Registry v2**.
2. Leave the **Registry** field blank. An empty field specifies Docker Hub (hub.docker.com).
3. In **Repository name**, enter the name of the repo to scan. For example, enter library/alpine to scan the official Alpine image. If the repo is part of an organization, use the organization/repository format. For example, bitnami/nginx.
4. In **Credential**, select the credentials to use.

   If you are scanning a public repository, leave this field blank.
If you are scanning a private repository, and Console doesn’t have your credentials yet, click **Add New**. Select either **Basic authentication** or **Certificate-based authentication**, and fill out the rest of the fields. For certificate-based authentication, provide a client certificate with private key, and an optional CA certificate.

5. In **OS type**, specify whether the repo holds **Linux** or **Windows** images.

6. In **Scanner**, select **Automatic**.

   Console automatically selects an available Defender to execute the scan job. Alternatively, you can explicitly select a Defender from the drop-down list. Defenders are listed according to the hosts where they run. For more information, see deployment patterns.

7. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.

8. Set **Cap** to the number of most recent images to scan. Leaving **Cap** set to the default value of 5 will scan the most recent 5 images. Setting this field to 0 will scan all images.

9. Click **Add**.

**STEP 4 |** Click the yellow save button.

**STEP 5 |** Verify that the images in the repository are being scanned.

1. Go to **Monitor > Vulnerabilities > Images > Registries**.

   A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

2. To get details about the vulnerabilities in an image, click on it.

   To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

### Scan images on Google Container Registry

**Prerequisites:**

- You have installed a Defender somewhere in your environment.

- GCR access is governed by Google’s storage permissions. For Prisma Cloud to scan GCR, your service account must have the GCP IAM **Storage View** role.

- You must grant Prisma Cloud access to your registry with a service account JSON key file. Your JSON token blob will look something like this:

```json
{
  "type": "service_account",
  "project_id": "my_project_id",
  "private_key_id": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "private_key": "-----BEGIN PRIVATE KEY-----
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX==


-----END PRIVATE KEY-----
"client_email": "XXXXXXXXXXXXXXX@XXXXXXXXXXXXXX.iam.gserviceaccount.com",
"client_id": "XXXXXXXXXXXXXXXXXXXXXXXXXX",
"auth_uri": "https://accounts.google.com/o/oauth2/auth",
"token_uri": "https://oauth2.googleapis.com/token",
"auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
"client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.iam.gserviceaccount.com"
}
STEP 1 | Open Console, then go to **Defend > Vulnerabilities > Registry**.

STEP 2 | Click **Add registry**.

STEP 3 | Enter the registry address in the **Registry** field (e.g. `gcr.io`).

STEP 4 | Enter the repository name followed by `/` in the **Repository** field (e.g. `company-sandbox/`).

STEP 5 | Click in the **Credential** field, then click **Add new**.

1. Select the **GCP** credential type, then paste the JSON token blob from your service account into the **Service Account** field. Leave the **API Token** field blank.
2. Save your credentials.
3. In **OS type**, specify whether the repo holds **Linux** or **Windows** images.
4. In **Scanner**, select **Automatic**.

   Console automatically selects an available Defender to execute the scan job. Alternatively, you can explicitly select a Defender from the drop-down list. Defenders are listed according to the hosts where they run. For more information, see deployment patterns.

5. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
6. Set **Cap** to the number of most recent images to scan. Leaving **Cap** set to 5 will scan the 5 most recent images. Setting this field to 0 will scan all images.

7. Click **Add**.

**STEP 6** | Click the yellow save button.

**STEP 7** | Verify that the images in the repository are being scanned.
1. Go to Monitor > Vulnerabilities > Images > Registries.
   A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

2. To get details about the vulnerabilities in an image, click on it.
   To force a specific repository to be scanned again, select Scan from the top right of the results table, then click on the specific registry to rescan.

Scan images on Harbor

Configure Prisma Cloud to scan your Harbor registry. To scan a repository in Harbor, create a new registry scan setting.

STEP 1 | Open Console

STEP 2 | Go to Defend > Vulnerabilities > Images > Registry Settings.

STEP 3 | Click Add Registry.

STEP 4 | In the dialog, enter the following information:
   1. In the Version drop-down list, select Harbor.
   2. In the Registry field, enter the FQDN of your Harbor registry (https://).
   3. In Repository name, enter the name of the repository to scan, or leave this blank to scan all repositories.
   4. In Credential, select the credentials to use.
      If Console doesn't have a copy of your credentials yet, click Add New. Select either Basic authentication, and fill out the rest of the fields. The minimum required credentials for each repository is Limited Guest.
   5. In OS type, specify whether the repo holds Linux or Windows images.
   6. In Scanner, select Automatic.
      Console automatically selects an available Defender to execute the scan job. Alternatively, you can explicitly select a Defender from the drop-down list. Defenders are listed according to the hosts where they run. For more information, see deployment patterns.
   7. In Number of scanners, enter the number of Defenders across which scan jobs can be distributed.
   8. Set Cap to the number of most recent images to scan. Leaving Cap set to the default value of 5 will scan the most recent 5 images. Setting this field to 0 will scan all images.
   9. Click Add.

STEP 5 | Click the yellow save button.

STEP 6 | Verify that the images in the repository are being scanned.
   1. Go to Monitor > Vulnerabilities > Images > Registries.
      A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.
   2. To get details about the vulnerabilities in an image, click on it.
      To force a specific repository to be scanned again, select Scan from the top right of the results table, then click on the specific registry to rescan.
Scan the IBM Cloud Container Registry

To scan a repository on IBM Cloud Container Registry, create a new registry scan setting.

**Prerequisites:** You have installed a Defender somewhere in your environment.

**STEP 1 | Open Console**

**STEP 2 | Set up credentials so that Prisma Cloud can access the images in your registry.**

1. Go to Manage > Authentication > Credentials Store.
2. Click Add credential.
3. Enter a name.
4. In Type, select IBM Cloud.
5. In Account GUID, enter the GUID for your IBM Cloud account. See the IBM Cloud Docs to learn how to get the GUID of an account.
6. In API Key, enter your API key. See the IBM Cloud Docs to learn how to create a service ID for Prisma Cloud, and then create an API key for the service ID.
7. Click Save.

**STEP 3 | Go to Defend > Vulnerabilities > Registry.**

**STEP 4 | Click Add registry.**

**STEP 5 | In the dialog, enter the following information:**

1. From the Version drop-down list, select IBM Cloud Container Registry.
2. In Registry, enter the registry address for your region.
   
   For example, if you use the us-south registry, enter registry.ng.bluemix.net.
3. In Namespace, enter the namespace for your image.
   
   For images in private registries, this field is mandatory. For images in IBM’s public registry, leave this field blank. Wildcards are not supported for this field.
   
   IBM provides namespaces to help you organize your registries. Namespaces are appended to the registry URL as follows: registry.<REGION>.bluemix.net/<NAMESPACE>
4. In Repository name, specify the repository to scan.
   
   If you leave this field blank or enter a wildcard, Prisma Cloud finds and scans all repositories in the registry.
   
   If you specify a partial string that ends with a wildcard, Prisma Cloud finds and scans all repositories that start with the partial string.
   
   If you specify an exact match, Prisma Cloud scans just the specified repository.
5. In Tag, enter an image tag.
   
   If you leave this field blank or enter a wildcard, Prisma Cloud finds and scans all images in the repository.
   
   If you specify a partial string that ends with a wildcard, Prisma Cloud finds and scans all images that match the partial tag.
   
   If you specify an exact match, Prisma Cloud scans just the specified image with specified tag.
6. In Credential, select the credential you just created.
7. In OS type, specify whether the repo holds Linux or Windows images.
8. In Scanner, select Automatic.
Console automatically selects an available Defender to execute the scan job. Alternatively, you can explicitly select a Defender from the drop-down list. Defenders are listed according to the hosts where they run. For more information, see deployment patterns.

9. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.

10. **Cap** the number of images to scan.

Specify the maximum number of images to scan in the given repository, sorted according to last modified date. To scan all images in a repository, set **Cap** to 0. For a complete explanation of **Cap**, see the table in registry scan settings.

11. Click **Add**.

**STEP 6** | Click the yellow save button.

**STEP 7** | Verify that the images in the repository are being scanned.

1. Go to **Monitor > Vulnerabilities > Images > Registries**.

   A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

2. To get details about the vulnerabilities in an image, click on it.

   To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

### Scan images on Artifactory Docker Registry

Artifactory is a service for hosting and distributing container images. Artifactory lets you segment the service by repository key, so that you can allocate dedicated registries per project, team, or any other facet. Repositories can be accessed with the Docker client. A repository is a collection of related images, versioned by tag.

Artifactory lets you configure how images in the repository are accessed with a setting called the **Docker Access Method**. Prisma Cloud supports the subdomain method and the repository method. The port method is not supported.

In the subdomain model, the repository is accessed through a reverse proxy. Each Docker repository is individually addressed by a unique value, known as the repository key, positioned in subdomain of the registry's URL.

$ docker {pull|push} <REPOSITORY_KEY>.art.example.com/<IMAGE>:<TAG>

In the repository path model, each repository can be directly addressed. The repository key is part of the path to the image repo.

$ docker {pull|push} art.example.com:443/<REPOSITORY_KEY>/<IMAGE>:<TAG>

Artifactory recommends that the subdomain method be used for production environments. The repository model is suitable for small test setups and proof of concepts.

**Configuring Prisma Cloud to scan images in your registry**

To scan images in a JFrog Artifactory Docker registry, create a new registry scan setting. You have a couple of options for setting up your scan.
1) Prisma Cloud can autodiscover and scan all images in all repos across the Artifactory service for versions of Artifactory greater than or equal to 6.2.0. In the registry scan settings, set the version to **JFrog Artifactory** and set the registry address to your reverse proxy.

2) Scan all repositories under a registry key. Registry keys effectively subdivide the Artifactory service into stand-alone fully-compliant Docker v2 registries. In the registry scan settings, set the version to **Docker Registry v2**, and set the registry address to the full path to the "sub-registry". For example: https://<REPOSITORY_KEY>.example.com/.
Prerequisites: You have installed a Defender somewhere in your environment.

Grant Prisma Cloud access to your repo

To scan Artifactory registries, Prisma Cloud requires a privileged service account.

STEP 1 | Log in Prisma Cloud Console, then go to Manage > Authentication > Credentials Store.

STEP 2 | Click Add credential.

STEP 3 | Enter a credential name, such as JFrog Artifactory.

STEP 4 | In Type, select Basic authentication.

STEP 5 | In Username, enter a username.

STEP 6 | In Password, enter a password.

STEP 7 | Click Save.

Configure the scan

After you set up your credentials, create a new registry scan setting.

STEP 1 | Open Console, then go to Defend > Vulnerabilities > Registry.
STEP 2 | Click **Add registry**.

STEP 3 | In the dialog, enter the following information:

1. From the **Version** drop-down list, select one of:
   - **JFrog Artifactory** — Autodiscover and scan all images in all repos across the Artifactory service.
   - **Docker Registry v2** — Scan all images in all repos under a specific repository key.
2. In **Registry**, specify the address to scan.
   - If you selected **JFrog Artifactory**, enter the FQDN of the reverse proxy.
   - If you selected **Docker Registry v2**, enter the FQDN, including subdomain, of the sub-registry.
3. In **Repository**, specify the repository to scan.
   - If you leave this field blank or enter a wildcard, Prisma Cloud finds and scans all repositories in the registry.
   - If you specify a partial string that ends with a wildcard, Prisma Cloud finds and scans all repositories that start with the partial string.
   - If you specify an exact match, Prisma Cloud scans just the specified repository.
4. In **Repository types**, select the repository types that Prisma Cloud should scan.
   - This setting is available only when **Version** is set to **JFrog Artifactory**. Specify at least one registry type (local, remote, virtual).
5. Do the same with the **Tag** field.
6. In **Credential**, select the JFrog Artifactory credentials you created.
7. In **OS type**, specify whether the repo holds **Linux** or **Windows** images.
8. In **Scanner**, select **Automatic**.
   - Console automatically selects an available Defender to execute the scan job. Alternatively, you can explicitly select a Defender from the drop-down list. Defenders are listed according to the hosts where they run. For more information, see deployment patterns.
9. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
10. **Cap** the number of images to scan.
   - **Cap** specifies the maximum number of images to scan in the given repository, sorted according to last modified date. To scan all images in a repository, set **Cap** to 0. For a complete explanation of **Cap**, see the table in registry scan settings.
11. Click **Add**.

STEP 4 | Click the yellow save button.

STEP 5 | Verify that the images in the repository are being scanned.

1. Go to **Monitor > Vulnerabilities > Images > Registries**.
   - A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.
2. To get details about the vulnerabilities in an image, click on it.
   - To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.
Troubleshooting

If Artifactory is deployed as an insecure registry, Defender cannot pull images for scanning without first configuring an exception in the Docker daemon configuration. Specify the URL of the insecure registry on the machine where the registry scanning Defender runs, then restart the Docker service. For more information, see the Docker documentation.

Scan images on OpenShift’s integrated Docker registry

To scan an OpenShift integrated registry, create a new registry scan setting.

Prerequisites:
- You’ve installed a Defender somewhere in your environment.
- You’ve obtained a service account token.

STEP 1 | Open Console, then go to Defend > Vulnerabilities > Registry.

STEP 2 | Click Add registry.

STEP 3 | In Version, select Red Hat OpenShift.

STEP 4 | Enter the registry address in the Registry field.

STEP 5 | In Repository, specify the repository to scan.

If you leave this field blank or enter a wildcard, Prisma Cloud finds and scans all repositories in the registry.

If you specify a partial string that ends with a wildcard, Prisma Cloud finds and scans all repositories that start with the partial string.

If you specify an exact match, Prisma Cloud scans just the specified repository.

STEP 6 | Click in the Credential field, then click Add new.

1. Select the Basic authentication credential type
2. In Username, enter any arbitrary value.
3. In Password, enter your service account token.
4. Save your credentials.

STEP 7 | In OS type, specify whether the repo holds Linux or Windows images.

STEP 8 | In Scanner, select Automatic.

Console automatically selects an available Defender to execute the scan job. Alternatively, you can explicitly select a Defender from the drop-down list. Defenders are listed according to the hosts where they run. For more information, see deployment patterns.

STEP 9 | In Number of scanners, enter the number of Defenders across which scan jobs can be distributed.

STEP 10 | Set Cap to the number of most recent images to scan. Leaving Cap set to 5 will scan the 5 most recent images. Setting this field to 0 will scan all images.

STEP 11 | Click Add.
STEP 12 | Click the yellow save button.

STEP 13 | Verify that the images in the repository are being scanned.

1. Go to Monitor > Vulnerabilities > Images > Registries.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

2. To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, select Scan from the top right of the results table, then click on the specific registry to rescan.

Trigger registry scans with webhooks

You can use webhooks to trigger a scan when images in your registry’s repositories are added or updated.

Prisma Cloud supports webhooks for:

- Docker Hub
- Docker Registry
- Azure Registry
- Nexus Repository

*Prisma Cloud requires Docker Registry 2.4 or later.*

*Google Container Registry and Amazon EC2 Container Registry do not currently support webhooks.*

For Docker Hub, you must have Automated Builds enabled for your repository. Docker Hub webhooks are called when an image is built or a new tag is added to your automated build repository.

For Docker Private Registry, webhooks are called when manifests are pushed or pulled, and layers are pushed or pulled. Prisma Cloud scans images in response to layer push events.

For Azure Registry, you can configure webhooks for your container registry that generate events when certain actions are performed against it. See Azure’s documentation for more information.

The benefit of webhook-initiated scans is that they are triggered as soon as images change, but support is limited to Docker Hub, Docker Registry, and Azure Registry. Prisma Cloud also supports scheduled registry scans, with support for almost all registry types, including Google Container Registry and Amazon EC2 Container Registry.

Securing Console’s management port

Webhooks call the Prisma Cloud API on Console’s management ports over either HTTP or HTTPS.

Although it is convenient to test webhooks with HTTP, we strongly recommend that you set up webhooks to call Console over HTTPS. To call webhooks over HTTPS, you must install a certificate trusted by the registry. For more information about securing Console’s management port with a custom cert, see Custom certs for Console access.

*By default, Prisma Cloud uses self-signed certificates to secure HTTP traffic. Self-signed certificates are not supported (trusted) by Docker Hub, and Docker Registry would require you to configure Prisma Cloud as a trusted CA (not supported, and not recommended).*
Instead install a certificate signed by a trusted certificate authority (CA), such as Comodo or Symantec.

**Setting up webhooks**

To set up webhook-initiated scans, configure your registry’s webhook with the URL provided in Console. The following procedure shows you how to set up webhooks in Docker Hub.

**Prerequisites:** Docker Hub, with Automated Builds enabled.

**STEP 1** | Open Console.

**STEP 2** | Go to **Defend > Vulnerabilities > Registry**.

**STEP 3** | Scroll down to the section **Registry webhooks**, then enter the following information:

1. Set **Allow image scans initiated by registry webhooks** to **On**.

   In order for this setting to be persisted, you must modify table under **Registry Settings** by adding, modifying, or deleting an entry, then clicking **Save**.

2. In the drop down list, select the DNS name or IP address that Docker Hub can use to reach Console.

   Your selection generates a URL that you will use to configure Docker Registry.

3. Copy the URL.

   By default, the generated URL employs HTTP. For HTTPS, replace http:// with https://.

**STEP 4** | Configure your repository.

The following sections show how to configure Docker Hub and Nexus Repository. For other repositories, consult the vendor’s documentation.

- Docker Hub
- Nexus Repository

**STEP 5** | Test the integration by triggering a build.

**STEP 6** | Go to **Monitor > Vulnerabilities > Registry** to view the scan report. Prisma Cloud scans the image as soon as it is built.
**Configuring Docker Hub**

Configure your Docker Hub repository.

**STEP 1** | Log into Docker Hub.

**STEP 2** | Select a repository, and then click Webhooks.

**STEP 3** | Create a new webhook. Specify a name, and paste the URL you copied from Console.

**STEP 4** | Click Save.

**Configuring Nexus Repository**

Configure the Nexus Repository. When setting up webhooks in Nexus Repository, select the "component" event type for triggering the webhooks.
Configure VM image scanning

Prisma Cloud can scan Linux Amazon Machine Images (AMIs).

The following AMIs aren’t supported:

- Images that don’t use cloud-init for bootstrapping, such as Red Hat Enterprise Linux CoreOS (CoreOS for OpenShift). RHCOS uses Ignition.
- Images that use paravirtualization.
- Images that only support old TLS protocols (less than TLS 1.1) for utilities such as curl. For example, Ubuntu 12.10.
- Encrypted images.

Prerequisites

The service account Prisma Cloud uses to scan AMIs must have at least the following policy:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeSecurityGroups",
                "ec2:CreateSecurityGroup",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:DeleteSecurityGroup",
                "ec2:DescribeImages",
                "ec2:DescribeInstances",
                "ec2:RunInstances",
                "ec2:CreateTags",
                "ec2:TerminateInstances"
            ],
            "Resource": "*"
        }
    ]
}
```

Deployment

VM image scanning is handled by the Console. Prisma Cloud’s Console scans a VM image by creating a VM instance which is running the VM image to be scanned. When you configure Prisma Cloud to scan VM images, you can define the number of scanners to use. Defining more than one scanner means that the Console will create a number of VM instances to scan multiple VM images simultaneously. For scanning large numbers of VM images, increase the number of scanners to improve throughput and reduce scan time.

If you remove a VM image, or it becomes unavailable, Prisma Cloud maintains the scan results for 30 days. After 30 days, the scan results are purged.

VM images scan settings

STEP 1 | Open Console.
STEP 2 | Go to Defend > Vulnerabilities/Compliance > Hosts > VM Images.

STEP 3 | Click Add Scope.

Each scope has the following parameters.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Specify the type of VM images to scan. The current supported VM images version is Amazon Machine Image (AMI).</td>
</tr>
<tr>
<td>Console Address</td>
<td>Specify the Console URL for the scanner VM instance to use.</td>
</tr>
<tr>
<td>Region</td>
<td>Specify the AWS region to scan.</td>
</tr>
<tr>
<td>VM images</td>
<td>Specify the names of the VM images to scan. This field supports pattern matching. To scan all VM images, simply enter wildcard (*).</td>
</tr>
<tr>
<td>Tags</td>
<td>Specify the AWS tags to scan. Use the key-value pattern 'key:value'. This field supports pattern matching. To scan VM images with all AWS tags, simply enter wildcard (*).</td>
</tr>
<tr>
<td>Excluded VM images</td>
<td>Specify VM images to exclude from the scan. This field supports pattern matching.</td>
</tr>
<tr>
<td>Credentials</td>
<td>Specify the credentials required to access the VM images. If the credentials have already been created in the Prisma Cloud credential store, select it. If not, click Add New.</td>
</tr>
<tr>
<td>Number of scanners</td>
<td>Number of AMIs to concurrently scan. Increase the number of scanners to increase throughput and reduce scan time.</td>
</tr>
<tr>
<td>Cap</td>
<td>Specify the maximum number of VM images to scan, sorted according to the last modified date. The most recently modified VM image is scanned first, followed by the image next most recently modified, and so on. To scan all VM images, set CAP to 0.</td>
</tr>
</tbody>
</table>

VM images rules

To define which VM images to scan, create a new VM images scan rule.

STEP 1 | Open Console.

STEP 2 | Go to Defend > Vulnerabilities/Compliance > Hosts > VM Images.

STEP 3 | Click Add Rule.

STEP 4 | Fill out your policy.
STEP 5 | Click Save.

Additional scan settings

Additional scan settings can be found under Manage > System > Scan, where you can set the VM images scan interval.
Configure code repository scanning

Prisma Cloud can scan GitHub repositories and identify vulnerabilities in your software's dependencies. Modern apps are increasingly composed of external, open source dependencies, so it's important to give developers tools to assess those components early in the development lifecycle. Repository scanning gives you early insight into the software as it's being developed, and long before apps are packaged (e.g. as a container) and deployed by CI/CD pipelines.

Currently, Prisma Cloud supports Python, Java, and JavaScript (Node.js).

Prerequisites

Prisma Cloud authenticates with the GitHub API using user-generated API tokens. The following scopes are required for scanning private repos. Prisma Cloud doesn't modify or write to your repos.

- repo — Full control of private repositories
- repo:status — Access commit status
- repo_deployment — Access deployment status
- public_repo — Access public repositories
- repo:invite — Access repository invitations
- security_events — Read and write security events

If you're scanning public repos only, select just the public_repo scope. The benefit of creating an access token for scanning public repos is that GitHub grants you a higher rate limit to their API, which Prisma Cloud utilizes for scanning.

Deployment

Prisma Cloud selects the repositories to scan according to a user-defined scope. For example, you might want to scan all repositories in your organization or just a subset of them. For each repo in scope, Prisma Cloud searches for well-known package manifest files, and enumerates the dependencies listed in them. Those dependencies are assessed against the latest threat data in the Intelligence Stream.

Code repository scans is handled by Console.

The following table lists the manifest files known to the scanner.

<table>
<thead>
<tr>
<th>Package manager</th>
<th>File name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java (Gradle)</td>
<td>build.gradle, build.gradle.kts, gradle.properties</td>
</tr>
<tr>
<td>Java (Maven)</td>
<td>pom.xml</td>
</tr>
<tr>
<td>Python (pip)</td>
<td>req*.txt</td>
</tr>
<tr>
<td>JavaScript (NPM)</td>
<td>package.json, package-lock.json, npm-shrinkwrap.json, bower.json</td>
</tr>
</tbody>
</table>

Finally, Prisma Cloud can continuously monitor your code repositories for vulnerabilities by rescanning on every push event. Prisma Cloud integrates with GitHub using webooks, which notify the scanner when there are changes in the repository.
Prisma Cloud uses the GitHub API. The GitHub API is rate-limited. For unauthenticated requests, which can be used to scan public repositories, the cap is very low (60 requests/hour). Here the rate limit is gauged by IP address. For authenticated requests, which can scan either public or private repositories, the cap is 5000 requests/hour. Here the rate limit is gauged per account.

Set up your credentials

Generate a personal access token in GitHub, and then save it in the Prisma Cloud Credentials Store so that the scanner can access your repositories for scanning.

STEP 1 | Generate a GitHub access token.

1. Log into your GitHub account.
2. Go to Settings > Developer Settings > Personal access tokens.
3. Click Generate new token.
4. Set the scope to repo.
If you're scanning public repos only, select just the public_repo scope.

5. Click **Generate token**. If your account requires SSO, enable it.
6. Copy the generated token.

[56b6559bc460917d28605dafaa2f58a980561625](#)  

**STEP 2** | Save the token in Prisma Cloud’s credentials store.
1. Log into Prisma Cloud Console.
2. Go to **Manage > Authentication > Credentials Store**.
3. Click **Add Credential**.
4. Enter a **Name** for the credential.
5. In **Type**, select GitHub access token.
6. In **Access Token**, paste the access token you generated in GitHub.
7. Click **Save**.

**Configure the repos to scan**

Specify the repositories to scan. If your repository specifies dependencies in non-standard package manifest files, specify them here so the scanner can parse them. If there are manifests the scanner should ignore, specify them here as well.

**STEP 1** | Open Console.

**STEP 2** | Go to **Defend > Vulnerabilities > Code Repositories**.

**STEP 3** | Click **Add Scope**. If this is your first repository, click **Add the first item**.

Each scope spec has the following parameters:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provider</td>
<td>GitHub is currently the only supported provider.</td>
</tr>
<tr>
<td>Type</td>
<td>To scan all repos in an organization, including both public and private repos, set the type to <strong>Private</strong>. You'll need to set up an access token, so that Prisma Cloud can access your repos. To scan public repositories not related to your account or organization, set the type to <strong>Public</strong>. When type is <strong>Public</strong>, credentials are not required, although API access to GitHub is capped to a very low value. Even if you're only scanning public repos, we recommend that you set up an access token for authenticated access.</td>
</tr>
<tr>
<td>Credential</td>
<td>Specify credentials for the repository owner. If the credentials have already been created in the Prisma Cloud credentials store, select it. If not, click <strong>Add New</strong>.</td>
</tr>
<tr>
<td>Repositories</td>
<td>Specify the repositories to scan in the format: owner/name When you've selected a credential, the drop-down lists all repositories in the owner's account. Wildcards are supported when the repo type is <strong>Private</strong>. They aren't supported when the type is <strong>Public</strong>.</td>
</tr>
<tr>
<td>Excluded manifest paths</td>
<td>Specify paths to be excluded for analysis. Wildcards are supported.</td>
</tr>
<tr>
<td>Advanced settings &gt; Explicit manifest names</td>
<td>Supported for Python only. Specify any additional file names that should be included for analysis. If you have a custom naming scheme for your manifest files, specify them here so that the scanner can find and parse them.</td>
</tr>
<tr>
<td>Advanced settings &gt; Python version</td>
<td>For a more accurate analysis of your app's dependencies, specify the version of Python you deploy in production. Otherwise, the scanner assumes the latest available version of Python.</td>
</tr>
</tbody>
</table>

**STEP 4** | Click **Add**.

**STEP 5** | Click **Save**.
Scan repos on push events

Configure GitHub webhooks to rescan your repositories on push events.

**STEP 1 |** Open Console.

**STEP 2 |** Go to **Defend > Vulnerabilities > Code Repositories.**

**STEP 3 |** In **Webhook settings,** select the publicly accessible name or IP address GitHub will use to notify Prisma Cloud that a push event occurred.

**STEP 4 |** Copy the URL.

**STEP 5 |** Configure GitHub.

1. Log into GitHub, select a repo, and go to **Settings > Webhooks.**
2. Click **Add webhook.**
3. In **Payload URL,** paste the URL you copied from Prisma Cloud Console.
4. In **Content type,** select **application/json.**
5. Select **Disable SSL verification.**
   
   For Compute Edition, you can enable SSL verification if your Console runs under a domain with a valid certificate signed by a known authority.

   For Prisma Cloud Enterprise Edition, select **Enable SSL verification.**
6. Leave all other settings in their default state.
7. Click **Add webhook.**
8. Verify that the ping webhook was delivered successfully.

**Policy**

Prisma Cloud ships with a default rule that alerts on vulnerabilities. In **Defend > Vulnerabilities > Code Repositories,** create vulnerability rules to tailor what's reported.

Additional scan settings can be found under **Manage > System > Scan,** where you can set the **scan interval.** By default, it’s 24 hours.
Malware scanning

Besides detecting software vulnerabilities (CVEs) and compliance issues (such as images configured to run as root), Prisma Cloud also detects malware in your container images. No special configuration is required to enable this feature.

Malware data is sourced from commercial providers, Prisma Cloud Labs, and open source lists. The image scanner looks for malware in binaries in the image layers, including the base layer.

*Malware scanning and detection is supported for Linux container images only. Windows containers are not supported.*

Detecting malware

When Prisma Cloud detects malware in an image, it logs the vulnerability in the image scan report.

To review the results of an image scan:

**STEP 1** | Open Console, then go to Monitor > Vulnerabilities > Images.

**STEP 2** | Click on an image to get a detailed report from the last image scan.

**STEP 3** | In the detailed report, click on the Compliance tab.

Issues with vulnerability ID 422 means that your image contains a file with an md5 signature of known malware.

What’s next?

Custom malware data can be uploaded to Prisma Cloud. After uploading your data, it is used in all subsequent images scans.

For more information about uploading custom malware data to Console, see Import custom malware data.
Vulnerability risk tree

Because Prisma Cloud knows the state of all the images in your environment, it can show you all the places you might be at risk to a given set of vulnerabilities. To generate a risk tree, provide a CVE, and Prisma Cloud returns:

- A list of images that contain packages affected by the specified CVE.
- A list of running containers (created from the images listed above) that are affected by the specified CVE.
- A list of namespaces where the containers affected by the specified CVE reside.
- A list of hosts where the images affected by the specified CVE reside.
- A list of serverless functions that are affected by the specified CVE.

The risk tree lets you create a detailed map of your exposure to a vulnerability, and can help you identify the best way to resolve it in your upstream images.

Generating a risk tree

Prisma Cloud's Vulnerability Explorer shows you risk trees for the top ten vulnerabilities in your container ecosystem. To see the risk tree for any arbitrary CVE, use the search tool at the top of the "Top Ten lists" table or Prisma Cloud API.

To generate a risk tree, submit a CVE to the API. The API returns an ordered tree of the images that contain those vulnerabilities, containers that are derived from those images, namespaces where these containers reside, and hosts where those images live. This allows you to automate, with a single API call, the creation of a detailed map of your exposure to the vulnerabilities.

To generate a risk tree, use the following endpoint:

GET /api/v1/stats/vulnerabilities/impacted-resources?cve=<CVE-ID>

For example, to generate a risk tree for CVE-2016-2109:

GET /api/v1/stats/vulnerabilities/impacted-resources?cve=CVE-2016-2109

The following listing shows an example response. For complete details about the response object, see the API reference.

```json
{
   "_id": "CVE-2017-6983",
   "riskTree": {
      
      "sha256:154de23b60b2a0651401012afff4c2da485f8076043e7241288bbeb88c9965fa": [
         
         "image": "docker.io/library/ubuntu-exploit:latest",
         "container": "",
         "host": "",
         "namespace": "",
         "factors": {
            "network": false,
            "internet": false,
            "rootPrivilege": false,
            "noSecurityProfile": false,
            "privilegedContainer": false
         }
      ]
   }
}
```
"sha256:45919d98e870eeb1b1d4ccb9458f992f372d9fad5c5f3efd034bc569a575f0ff9": [
  {
    "image": "docker.io\morello\docker-whale:latest",
    "container": "confident_archimedes",
    "host": "ian-23.c.cto-sandbox.internal",
    "namespace": "",
    "factors": {
      "network": false,
      "internet": false,
      "rootPrivilege": true,
      "noSecurityProfile": false,
      "privilegedContainer": false
    }
  }
],
"sha256:67759a80360cbaf77ec1e8eaa0590f07ba04c26ef496efbc9039f217fd9d6": [
  {
    "image": "docker.io\library\ubuntu:14.04",
    "container": "",
    "host": "",
    "namespace": "",
    "factors": {
      "network": false,
      "internet": false,
      "rootPrivilege": false,
      "noSecurityProfile": false,
      "privilegedContainer": false
    }
  }
],
"registryImages": {
},
"hosts": [
  "ian-23.c.cto-sandbox.internal"
],
"functions": {
}
Detect vulnerabilities in unpackaged software

Typically, software in images is added through a package manager, such as apt, yum, npm. Prisma Cloud has a diverse set of upstream vulnerability data sources covering many different package managers across operating systems, including coverage for Node, Python, Java, and Ruby components. In these cases, Prisma Cloud typically uses the package manager’s metadata to discover the installed components and versions and compares this data to the realtime CVE data feed provided via the intelligence stream. However, sometimes you may install software into images without using a package manager, by just having a line in a Dockerfile to ADD the binary to the image or building it via a configure, make, install approach. In these cases, there is no package manager data associated with the application.

Prisma Cloud uses a variety of advanced analysis techniques to detect metadata about software not installed via package managers. This analysis then feeds our existing vulnerability detection and blocking mechanisms, continuing to give you a single view of all the vulnerabilities within a given image, regardless of whether they’re from the distribution layer, an app package manager, or added independently.

Supported apps

The following apps are currently supported. But with future requests from customers on additional support this list may be extended.

- Redis
- Nginx
- Mongo
- Mysql
- Httpd
- Java
- Apache
- Postgres
- Node
- Ruby
- Python
- PHP

Nothing is required to enable the functionality described in this article. It is enabled by default.

The following screenshot shows what a vulnerability scan report looks like when a vulnerability is discovered in a binary that was not installed into the image with a package manager:

<table>
<thead>
<tr>
<th>OS</th>
<th>CVE Score</th>
<th>Version</th>
<th>Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>High</td>
<td>file version 5.11-26.el7 has 1 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>High</td>
<td>expat version 2.1.0-8.el7 has 2 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>High</td>
<td>curl version 7.29.0-25.el7.centos has 12 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>High</td>
<td>cracklib version 2.6.0-31.el7 has 1 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>High</td>
<td>bind-license version 9.9.4-29.el7_2.3 has 2 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>Node.js</td>
<td>Medium</td>
<td>node version 0.10.41 has 5 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>Node.js</td>
<td>Medium</td>
<td>serve version 2.3.0 has 1 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Medium</td>
<td>request version 2.42.0 has 1 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Medium</td>
<td>util-linux version 2.23.2-26.el7_2.3 has 1 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Medium</td>
<td>unzip version 6.0-15.el7 has 2 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Moderate</td>
<td>tar version 1.26-29.el7 has 1 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Medium</td>
<td>systemd-tls version 219-19.el7_212 has 1 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Medium</td>
<td>systemd-tls version 219-19.el7_212 has 1 vulnerabilities. VIEW DETAILS</td>
<td></td>
</tr>
</tbody>
</table>
Customers can open support tickets to request support for additional binaries apart from those aforementioned.
CVSS scoring

Because severity terminology can vary between projects, Prisma Cloud normalizes severity ratings into a common schema. Prisma Cloud leverages the CVSS 3.0 scoring system.

Mappings

We only normalize vulnerability ratings for the purpose of creating rules. Console’s Monitoring section shows vendor terminology, not Prisma Cloud’s normalized scores (low, medium, high, critical).

The following table maps popular vendor terminology to Prisma Cloud normalized scores:

<table>
<thead>
<tr>
<th>Vendor terminology</th>
<th>Prisma Cloud score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unimportant</td>
<td>Low</td>
</tr>
<tr>
<td>Unassigned</td>
<td>Low</td>
</tr>
<tr>
<td>Negligible</td>
<td>Low</td>
</tr>
<tr>
<td>Not yet assigned</td>
<td>Low</td>
</tr>
<tr>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Moderate</td>
<td>Medium</td>
</tr>
<tr>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Important</td>
<td>High</td>
</tr>
<tr>
<td>Critical</td>
<td>Critical</td>
</tr>
</tbody>
</table>

In the absence of project-specific terminology, Prisma Cloud normalizes using the CVSS base scores defined by NIST. In addition to the numeric CVSS scores, NVD provides severity rankings of Low, Medium, High, and Critical. These qualitative rankings are simply mapped from the numeric CVSS scores:

<table>
<thead>
<tr>
<th>CVSS base score</th>
<th>Prisma Cloud severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 - 3.9</td>
<td>Low</td>
</tr>
<tr>
<td>4.0 - 6.9</td>
<td>Medium</td>
</tr>
<tr>
<td>7.0 - 8.9</td>
<td>High</td>
</tr>
<tr>
<td>9.0 -10.0</td>
<td>Critical</td>
</tr>
</tbody>
</table>
Google Cloud Container Builder

As a prerequisite for performing vulnerability scanning with Google Cloud Container Builder as explained here, download `scan_helper.c` from here.
Windows container image scanning

There are a number of things to consider when scanning Windows container images. First, Prisma Cloud Console only runs on Linux hosts. Prisma Cloud Defender, which does the actual scanning work, comes in a number of flavors. On Windows, Prisma Cloud supports Container Defender and Host Defender.

To scan Windows images:

- The Windows Intelligence Stream must be enabled. You can find the setting under Manage > System > Intelligence. By default, the Windows Intelligence Stream is disabled.
- The container OS version must match the host OS version where Defender runs. For example, Defender on Windows Server 1803 can scan nanoserver:1803, but it can’t scan nanoserver:1809. Conversely, Defender on Windows Server 1809 can scan nanoserver:1809, but it can’t scan nanoserver:1803.
- Prisma Cloud requires a privileged user inside the container to scan it. In more recent versions of Windows (Windows Server, version 1803 or higher, build 17134 or higher), Prisma Cloud uses the ContainerAdministrator account. This account has complete access to the whole file system and all of the resources in the container. In older versions of Windows, specifically Windows Server 2016 (version 1607, build 14393), ContainerAdministrator does not exist, so Prisma Cloud uses the default user.
Serverless function scanning

Prisma Cloud can scan serverless functions for vulnerabilities. Prisma Cloud supports AWS Lambda, Google Cloud Functions, and Azure Functions.

Serverless computing is an execution model in which a cloud provider dynamically manages the allocation of machine resources and schedules the execution of functions provided by users. Serverless architectures delegate the operational responsibilities, along with many security concerns, to the cloud provider. This new model raises new security concerns, while some familiar old concerns still apply. In particular, your app itself is still prone to attack. The vulnerabilities in your code and associated dependencies are the footholds attackers use to compromise an app. Prisma Cloud can show you a function’s dependencies, and surface the vulnerabilities in those dependent components.

Capabilities

For serverless, Prisma Cloud can scan Node.js, Python, and Java packages. There is currently no support for AWS or Azure C# functions.

Prisma Cloud scans are triggered by the following events:

- When the settings change, including when new functions are added for scanning.
- When you explicitly click the Scan button in the Monitor > Vulnerabilities > Functions page.
- Periodically. By default, Prisma Cloud rescans serverless functions every 24 hours, but you can configure a custom interval in Manage > System > Scan.

Scanning a serverless function

Configure Prisma Cloud to periodically scan your serverless functions. Unlike image scanning, all function scanning is handled by Console.

STEP 1 | Open Console.

STEP 2 | Go to Defend > Vulnerabilities > Functions.

STEP 3 | Click Add serverless account.

STEP 4 | In the dialog, enter the following settings:

1. In Provider, select your cloud platform.
2. Specify a Region.
3. Specify a function name.

   Wildcards are supported. See the table at the top of the page for the supported syntax and examples.
4. Select or create credentials so that Prisma Cloud can access your account.
   - AWS — Specify either an IAM user credential (access key ID and secret access key) or IAM role.
   - Google — Specify a service key.
   - Azure — Specify a user access token.
5. Specify a cap for the number of functions to scan.

   Prisma Cloud scans the X most recent functions, where X is the cap value.
6. Click Add.

STEP 5 | Click the yellow save button.
STEP 6 | View the scan report. Go to Monitor > Vulnerabilities > Functions.

Authenticating with AWS

The serverless scanner is implemented as part of Console. The scanner requires the AWSLambdaReadOnlyAccess permissions policy.

If authenticating with an IAM user, use the Security Token Service (STS) to temporarily issue security credentials to Prisma Cloud to scan your Lambda functions. AWS STS is considered a best practice per the AWS Well-Architected Framework. For more on how to use AWS STS, see here.

When authenticating with an IAM user, Console can access and scan functions across multiple regions. The following dialog shows Console set up to scan Lambda functions using an IAM user with STS. The user credentials (access key ID and secret access key) were previously entered in the Prisma Cloud credentials store, and the role with the AWSLambdaReadOnlyAccess permissions policy was already created in AWS.

![Add serverless account](image)

The Prisma Cloud serverless scanner can also authenticate with AWS using an IAM role. If Console authenticates with AWS using an IAM role, it can only scan the functions in the region where the EC2 instance is deployed. The following dialog shows Console set up to scan Lambda functions using an IAM role:

![Add serverless account](image)
Scanning Azure Functions

Azure Functions are architected differently than AWS Lambda and Google Cloud Functions. Azure function apps can hold multiple functions. The functions are not segregated from each other. They share the same file system. Rather than separately scanning each function in a function app, download the root directory of the function app, which contains all its functions, and scan them as a bundle.

Prisma Cloud only scans Linux functions that use External package URL as the deployment technology. For more information, see Deployment technologies in Azure Functions.

To do this, you must know the Region, Name (of the function), and Service Key. To get the Service Key, download and install the Azure CLI, then:

STEP 1 | Log into your account with a user that has the User Account Administrator role.

```bash
$ az login
```

STEP 2 | Get the service key.

```bash
$ az ad sp create-for-rbac --sdk-auth --name twistlock-azure-serverless-scanning --role contributor
```

Sample output from the previous command:

```plaintext
{
  "clientId": "f8e9de20-45bd-af94-ae11-b9r8c5tffy3b6",
  "clientSecret": "4dfds4b2-6d27-512-b5ff-56123043c4dc",
  "subscriptionId": "e19332m-z2bd-501c-dd11-234m547a944e",
  "tenantId": "c189c61a-6c27-41c3-9949-ca5c8cc4a624",
  "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",
  "resourceManagerEndpointUrl": "https://management.azure.com/",
  "activeDirectoryGraphResourceId": "https://graph.windows.net/",
  "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",
  "galleryEndpointUrl": "https://gallery.azure.com/",
  "managementEndpointUrl": "https://management.core.windows.net/"
}
```
STEP 3 | Copy the JSON output, which is your secret key, and paste it into the Service Key field for your Azure credentials in Prisma Cloud Console.

Scanning functions with twistcli

You can also use the twistcli command line utility to scan your serverless functions. First download your serverless function as a ZIP file, then run:

```
$ twistcli serverless scan <SERVERLESS_FUNCTION.ZIP>
```

To view scan reports in Console, go to Monitor > Vulnerabilities > Functions > CI or Monitor > Compliance > Functions > CI.

Twistcli Options

- **--address URI** --
  Required. Complete URI for Console, including the protocol and port. Only the HTTPS protocol is supported. By default, Console listens to HTTPS on port 8083, although your administrator can configure Console to listen on a different port.
  Example: --address https://console.example.com:8083
- **-u, --user USERNAME** --
  Username to access Console. If not provided, the TWISTLOCK_USER environment variable will be used if defined, or "admin" is used as the default.
- **-p, --password PASSWORD** --
  Password for the user specified with -u, --user. If not specified on the command-line, the TWISTLOCK_PASSWORD environment variable will be used if defined, or otherwise will prompt for the user’s password before the scan runs.
- **--project PROJECT NAME** --
  Interface with a specific supervisor Console to retrieve policy and publish results.
  Example: --project "Tenant Console"
- **--details** --
  Show all vulnerability details.
- **--tlscacert PATH** --
  Path to Prisma Cloud CA certificate file. If no CA certificate is specified, the connection to Console is insecure.
- **--include-js-dependencies** --
  Include javascript package dependencies.
- **--token TOKEN** --
  Token to use for Prisma Cloud Console authentication. Tokens can be retrieved from the API endpoint api/v1/authenticate or from the Manage > Authenticate > User Certificates page in Console.
- **--cloudformation-template PATH** --
  Path to the CloudFormation template file in JSON or YAML format. Prisma Cloud scans the function source code for AWS service APIs being used, compares the APIs being used to the function permissions, and reports when functions have permissions for APIs they don't need.
Function name to be used in policy detection and Console results. When creating policy rules in Console, you can target specific rules to specific functions by function name. If this field is left unspecified, the function zip file name is used.

- **--output-used-apis** --
  Report APIs used by the function

- **--publish** --
  Publish the scan result to the Console. True by default.
VMWare Tanzu blobstore scanning

Prisma Cloud for TAS can scan the droplets in your blobstores for vulnerabilities. Prisma Cloud can be configured to scan your blobstores periodically. Defenders are the entities that perform the scanning.

When you install PCF Defender in your environment, it automatically scans the running apps and hosts in your environment without any special configuration required.

Tanzu stores large binary files in blobstores. Blobstores are roughly equivalent to registries. One type of file stored in the blobstore is the droplet.

Droplets are archives that contain ready to run applications. They are roughly equivalent to container images. Droplets contain the OS stack, a buildpack (which contains the languages, libraries, and services used by the app), and custom app code. Before running an app on your infrastructure, the Cloud Controller stages it for delivery by combining the OS stack, buildpack, and source code into a droplet, then storing the droplet in a blobstore.

The twistcli command line tool also lets you scan droplet files directly. You can integrate twistcli into your CLI to pass or fail builds based on vulnerability thresholds.

Configure Prisma Cloud to scan a blobstore

Prisma Cloud can scan internal and external blobstores, and blobstores configured to use the Fog Ruby gem or WebDAV protocol.

External blobstores that require a custom authentication flow, such as those offered by cloud providers, are not supported.

Prequisite: You've already installed PCF Defender in your environment.

STEP 1 | Log into Prisma Cloud Console.
STEP 2 | Go to Defend > Vulnerabilities > PCF Blobstore.
STEP 3 | Click Add Blobstore.
STEP 4 | Specify the cloud controller.
STEP 5 | Specify a Defender to execute the scanning.
    Prisma Cloud lists all the agentIDs where Defender is installed. To correlate the agentID to the Diego cell's IP address, and determine which host runs a Defender, log into any Diego cell, and inspect /var/vcap/instance/dns/records.json. This file shows the correlation between agentID and host IP address.
STEP 6 | Specify the droplets to scan. To scan all droplets, enter a wildcard (*).
STEP 7 | Specify the maximum number of droplets to scan. To scan all droplets, enter 0.
STEP 8 | Click Add.
STEP 9 | Click Save.
Review scan reports

Scan reports show all vulnerabilities found in the droplets in your blobstores. By default, droplets are rescanned every 24 hours.

A droplet, which is an artifact of the app staging process, contains the minimum required data to specify an app (binaries/libraries). Droplets are stored in blobstores. Review scan reports for droplets in Monitor > Vulnerabilities > PCF Blobstore.

When an application is run in a Diego cell, it's run on top of a stack, currently cflinuxfs3, which is derived from Ubuntu Bionic 18.04. Defender automatically scans all running applications (buildpack and docker). Review the scan reports for running apps in Monitor > Vulnerabilities > Images.

If you compare the findings for a buildpack app in Monitor > Vulnerabilities > PCF Blobstore and Monitor > Vulnerabilities > Images, you'll notice a difference in the number of findings. Remember that Monitor > Vulnerabilities > Images reports any additional findings in the app's underlying stack that would not be found in the droplet alone.

When TAS stages Docker-based apps, it doesn't stage an associated droplet in the blobstore. Therefore, blobstore scanning alone won't cover Docker-based apps. If you're running Docker containers in TAS, and you want to scan the images before they run, then configure Prisma Cloud to scan the container registry.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to Monitor > Vulnerabilities > PCF Blobstore to see a list of summary reports for each droplet.

STEP 3 | To drill into a specific scan report, click on a row in the table.
Prisma Cloud helps enterprises monitor and enforce compliance for hosts, containers and serverless environments. Use the compliance management system to enforce standard configurations and security best practices.

- Compliance Explorer
- Manage compliance
- CIS Benchmarks
- Prisma Cloud Labs compliance checks
- Serverless functions compliance checks
- Windows compliance checks
- Custom compliance checks
- Extensible compliance checks
- Trusted images
- Host scanning
- VM image scanning
- Detect secrets
- Cloud discovery
Compliance Explorer

Compliance Explorer gives you a picture of the overall compliance of the entities in your container environment.

To view Compliance Explorer, go to Monitor > Compliance > Compliance Explorer.

Reviewing the data

Compliance Explorer consists of two parts:

- **Roll-up charts** — Show the overall compliance for each entity type (images, containers, and hosts) as a percentage of total checks passed over total checks enabled. A grade of 100% means there are no compliance issues. The trend charts shows how your compliance has changed over the past 30 days.

- **Table of compliance issues** — Lists all compliance checks that failed. The checks that are evaluated are determined by the rules you’ve defined in Defend > Compliance > Policy. Issues are listed by compliance ID. Clicking on a row opens a dialog that lists all entities that do not comply with the given ID.

Statistical data is calculated every 24 hours. You can force Console to recalculate the statistics for the current day with the current data by clicking the Refresh button in the top left of Compliance Explorer. The Refresh button has a red marker when new data is available to be crunched.
Manage compliance

Prisma Cloud can monitor and enforce compliance settings across your environment. Out of the box, Prisma Cloud supports hundreds of discrete checks that cover images, containers, hosts, clusters, and clouds.

Applications are typically built with numerous components. Many components have established best practices for securing them against attack. Not everyone has the bandwidth to painstakingly work through the details of every best practice to determine which ones are the most important. Prisma Cloud lets your security team centrally review all best practices, enable the ones that align with your organization’s security mandate, then evenly enforce them across your environment.

Prisma Cloud’s predefined checks are based on industry standards, such as the CIS benchmarks, as well as research and recommendations from Prisma Cloud Labs. Additionally, you can implement your own compliance checks with scripts or XCCDF.

Enforcement

Compliance rules are defined and applied in the same way as vulnerability rules. For checks that can be performed on static images, those checks are performed as images are scanned (either in the registry or on local hosts). Results are then returned to Console and displayed in the compliance reports under Monitor > Compliance.

When compliance rules are configured with block actions, they are enforced when a container is created. If the instantiated container violates your policy, Prisma Cloud prevents the container from being created.

Note that compliance enforcement is only one part of a defense in depth approach. Because compliance enforcement is applied at creation time, it is possible that a user with appropriate access could later change the configuration of a container, making it non-compliant after deployment. In these cases, the runtime layers of the defense in depth model provide protection by detecting anomalous activity, such as unauthorized processes.

Assume that you want to block any container that runs as root. The flow for blocking such a container is:

1. Prisma Cloud admin creates a new compliance rule that blocks containers from running as root.
2. The admin optionally targets the rule to a specific resources, such as a set of hosts, images, or containers.
3. Someone with rights to create containers attempts to deploy a container to the environment.
4. Prisma Cloud compares the image being deployed to the compliance state that it detected when it scanned the image. For deploy-time parameters, the specific Docker client commands sent are also analyzed.

   1. If the comparison determines that the image is compliant with the policy, the ‘docker run’ command is allowed to proceed as normal, and the return message from Docker Engine is sent back to the user.
   2. If the comparison determines that the image is not compliant, the container_create command is blocked and Prisma Cloud returns an error message back to the user describing the violation.

5. In both success and failure cases, all activities are centrally logged in Console and (optionally) syslog.

Creating compliance rules

This procedure shows you how to set up a container compliance rule. As an example, you configure Prisma Cloud to block any containers running as root.

STEP 1 | Open Console, then go to Defend > Compliance > Containers and Images.

STEP 2 | Click Add rule.
1. Enter a rule name, such as **my-rule**.
2. In the search field under **Compliance actions**, enter **Container is running as root**.
   
   As you type, the available checks are filtered to match your search query.
3. For check 599 (Container is running as root), set the action to **Block**.
4. Under **Add resources**, accept the default settings. The default filter applies your rule to all containers, images, hosts, and labels in your environment.
5. Click **Save**.
   
   Your rule is now activated.

**STEP 3 | Verify that your rule is being enforced.**

1. Connect to a host running Defender, then run the following command, which starts an Ubuntu container with a root user (uid 0).

   ```
   $ docker run -u 0 -ti library/ubuntu /bin/sh
   ```

   Defender should block the command with the following message:

   ```
   docker: Error response from daemon: oci runtime error: [Prisma Cloud]
   Container operation blocked by policy: my-rule, has 1 compliance issues.
   ```

**Reporting full results**

By default, Prisma Cloud reports only the compliance checks that fail. Sometimes you need both negative and affirmative results to prove compliance. You can configure Prisma Cloud to report checks that both pass and fail.

The contents of a full compliance report (both passed and failed checks) is the sum of all applied rules. If your compliance policy raises an alert for only two checks, your compliance report will show the results of two checks. To report on **all** compliance checks, set all compliance checks to either alert or block.

**STEP 1 | Open Console, then go to Defend > Compliance > [Containers and Images | Hosts].**

**STEP 2 | Click Add rule.**

1. Enter a rule name.
2. Under **Reported results**, click **Passed and Failed Checks**.
3. Click **Save**.
   
   Your rule is now activated.

**STEP 3 | Verify that the compliance reports show both passed and failed checks.**

1. Go to Defend > Compliance, select any tab, then click on a resource in the table to open its scan report. You will see a list of checks that have both passed and failed.
<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Severity</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>CIS</td>
<td>high</td>
<td>Fail</td>
<td>(CIS_Docker_CE_v1.0 - 4.1) Image should be created with a user</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>406</td>
<td>CIS</td>
<td>medium</td>
<td>Fail</td>
<td>(CIS_Docker_CE_v1.0 - 4.6) Add HEALTHCHECK instruction to the container image</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>422</td>
<td>twistlock</td>
<td>critical</td>
<td>Pass</td>
<td>Image contains malware</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>426</td>
<td>twistlock</td>
<td>high</td>
<td>Pass</td>
<td>Image contains binaries used for crypto mining</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>425</td>
<td>twistlock</td>
<td>high</td>
<td>Pass</td>
<td>Private keys stored in image</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>424</td>
<td>twistlock</td>
<td>high</td>
<td>Pass</td>
<td>Sensitive information provided in environment variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>423</td>
<td>twistlock</td>
<td>high</td>
<td>Pass</td>
<td>Image is not trusted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>420</td>
<td>twistlock</td>
<td>medium</td>
<td>Pass</td>
<td>Image is not updated to latest</td>
</tr>
</tbody>
</table>
CIS Benchmarks

The CIS Benchmarks provide consensus-oriented best practices for securely configuring systems. Prisma Cloud provides checks that validate the recommendations in the following CIS Benchmarks:

- Docker Benchmark
- Kubernetes Benchmark
- Distribution Independent Linux
- Amazon Web Services Foundations

We have graded each check using a system of four possible scores: critical, high, medium, and low. This scoring system lets you create compliance rules that take action depending on the severity of the violation. If you want to be reasonably certain that your environment is secure, you should address all critical and high checks. By default, all critical and high checks are set to alert, and all medium and low checks are set to ignore. We expect customers to review, but probably never fix, medium and low checks.

There are just a handful of checks graded as critical. Critical is reserved for things where your container environment is exposed to the Internet, and can result in a direct attack by somebody on the outside. They should be addressed immediately.

Prisma Cloud has not implemented CIS checks marked as *Not Scored*. These checks are hard to define in a strict way. Other checks are not implemented because the logic is resource-heavy, results depend on user input, or files cannot be parsed reliably.

Additional details about Prisma Cloud’s implementation of the CIS benchmarks

The compliance rule dialog provides some useful information. Compliance rules for containers can be created under *Defend > Compliance > Containers and Images*, while compliance rules for hosts can be created under *Defend > Compliance > Hosts*.

*Benchmark versions* – To see which version of the CIS benchmark is supported in the product, click on the *All types* drop-down list.
Grades — To see Prisma Cloud’s grade for a check, see the corresponding Severity column.

Built-in policy library — To enable the checks for the PCI DSS, HIPAA, NIST SP 800-190, and GDPR standards, select the appropriate template.

Prisma Cloud didn’t implement the following recommendations from the CIS Distribution Independent Linux benchmark:

- 1.7.2 - Ensure GDM login banner is configured — By default, most server distributions ship without a windows manager. A manual assessment is required.
• 2.2.1.2 - Ensure ntp (Network Time Protocol) is configured — CIS did not score this recommendation. A manual assessment is required.
• 2.2.1.3 - Ensure chrony is configured — CIS did not score this recommendation. A manual assessment is required.
• 5.3.1 - Ensure password creation requirements are configured — This recommendation cannot be implemented generically because password requirements vary from organization to organization. A manual assessment is required.
Prisma Cloud Labs compliance checks

Prisma Cloud Labs compliance checks are designed by our research team and fill gaps not offered by other benchmarks. Like all compliance checks, Prisma Cloud’s supplementary checks monitor and enforce a baseline configuration across your environment.

Prisma Cloud Labs compliance checks can be enabled or disabled in custom rules. New rules can be created under **Defend > Compliance > Policy.**

**Checks**

- **597 — Secrets in clear text environment variables (container and serverless function check)**
  Checks if a running container (instantiated from an image) or serverless function contains sensitive information in its environment variables. These env vars can be easily exposed with docker inspect, and thus compromise privacy.

- **598 — Container app is running with weak settings**
  Weak settings incidents indicate that a well-known service is running with a non-optimal configuration. This covers settings for common applications, specifically: Mongo, Postgres, Wordpress, Redis, Kibana, Elastic Search, RabbitMQ, Tomcat, Haproxy, KubeProxy, Httpd, Nginx, MySql, and registries. These check for things such as the use of default passwords, requiring SSL, etc. The output for a failed compliance check will contain a "Cause" field that gives specifics on the exact settings detected that caused a failure.

- **599 — Container is running as root (container check)**
  Checks if the user value in the container configuration is root. If the user value is 0, root, or "" (empty string), the container is running as a root user, and the policy’s configured effect (ignore, alert, or block) is actuated.

- **420 — Image is not updated to latest (image check)**
  For running containers, Prisma Cloud checks that the creation time of each layer in image:tag is the same as its corresponding image:tag in the registry.

  For any image pulled from a password protected registry/repo, the registry must be configured in Prisma Cloud. To add a registry, go to **Defend > Vulnerabilities > Registry.**

  If an image does not belong to any user configured registry, and the image origin is Docker Hub, the image is compared against image:tag in Docker Hub.

  Each layer in the image is assessed separately. If a layer cannot be found in the registry, it is skipped, and the next layer is assessed.

- **422 — Image contains malware (image check)**
  Checks if any binary in the image matches the md5 checksum for known malicious software.

- **423 — Image is not trusted (image check)**
  Checks if unauthorized (untrusted) images are pulled or loaded into your environment.

  Prisma Cloud provides a mechanism to specify specific registries, repositories, and images that are considered trusted. Enable this check to prevent unauthorized containers from running in your critical environment. For more information, see **Trusted images.**

- **424 — Sensitive information provided in environment variables (image and serverless function check)**
Checks if images or serverless functions contain sensitive information in their environment variables. Container images define environment variables with the Dockerfile ENV instruction. These environment variables can be easily exposed with `docker inspect`.

- **425 — Private keys stored in image (image and serverless function check)**

  Searches for private keys stored in an image or serverless function. If found, the policy effect (ignore, alert, block) is applied on deployment.

- **426 — Image contains binaries used for crypto mining (image check)**

  Detects when there are crypto miners in an image. Attackers have been quietly poisoning registries and injecting crypto mining tools into otherwise legitimate images. When you run these images, they perform their intended function, but also mine Bitcoin for the attacker. This check is based on research from Prisma Cloud Labs. For more information, see Real World Security: Software Supply Chain.

- **428 — Package binaries should not be altered**

  Checks the integrity of package binaries in an image. During an image scan, every binary's checksum is compared with its package info. If there's a mismatch, a compliance issue is raised.

  Besides scan-time, this compliance issue an also be raised at run-time if a modified binary is spawned.

### Prisma Cloud Labs Istio compliance checks

The Istio family of compliance checks lets you enforce a secure Istio configuration and address risks such as misconfigured TLS settings and universally scoped service roles. The goals of the compliance rules are to:

- Ensure mutual TLS is configured correctly (enabled and over HTTPs).
- Ensure RBAC policy is configured with service level access control (service x can only talk with service y).
- Ensure RBAC policy is not too permissive.

**Checks**

- **427 — Configure TLS per service using Destination Rule traffic policy**

  Ensures mutual TLS is globally enabled.

- **429 — Enable Istio authorization by creating RbacConfig named default**

  Removes any additional RbacConfig and ensure there is only one RbacConfig named default

- **430 — Set RbacConfig mode to ON, ON_WITH_INCLUSION or ON_WITH_EXCLUSION**

  Extracts RbacConfig resource and make sure it is not set to OFF.

- **432 — Avoid using service role rules that grant access to all services**

  Ensures RBAC rules not too permissive. Verify there are no wildcards in the service roles.

- **433 — Avoid binding service roles to all service accounts**

  Ensures service roles are at the service level and not the namespace level, which grants access to all services in the namespace.
Serverless functions compliance checks

Prisma Cloud Labs has developed compliance checks for serverless functions. Currently, only AWS Lambda is supported.

In AWS Lambda, every function has an execution role. Execution roles are identities with permission policies that control what functions can and cannot do in AWS. When you create a function, you specify an execution role. When the function is invoked, it assumes this role.

When Prisma Cloud scans the functions in your environment, it inspects the execution role for overly permissive access to AWS services and resources. Two fields are inspected: resource and action.

Resource

Specifies the objects to which the permission policy applies. Resources are specified with ARNs. ARNs let you unambiguously specify a resource across all of AWS. ARNs have the following format:

```
ar:partition:service:region:account-id:resource
```

Where:

- **service**—Identifies the AWS product, such as Amazon S3, IAM, or CloudWatch Logs.
- **resource**—Identifies the objects in the service. It often includes the resource type, followed by the resource name itself. For example, the following ARN uniquely identifies the user Francis in the IAM service:

```
ar:aws:iam::58697563310:user/Francis
```

Action

Describes the tasks that can be performed on the service. For example, ec2:StartInstances, iam:ChangePassword, and s3:GetObject. Wildcards can be used to grant access to all the actions of a given AWS service. For example, s3:* applies to all S3 actions.

Types of issues

The following permission policy is tightly scoped. It grants read-write only access to the Books table. Prisma Cloud would not flag an execution role with this type of permissions policy.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "dynamodb:GetItem",
      "dynamodb:BatchGetItem"
    ],
  }
}
```

The following permissions policy has been implemented carelessly. It allows all DynamoDB operations on all tables owned by the AWS account in the current region, including dynamodb:DeleteTable, which has serious implications for the integrity and availability of your data. This type of configuration would raise
compliance check 437 because the execution role permits all DynamoDB operations, and it's unlikely a function actually needs this range of capabilities.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "AllAPIActionsOnBooks",
    "Effect": "Allow",
    "Action": "dynamodb:*",
    "Resource": "*"
  }
}
```

Compliance check details

The following checks are supported:

- **434: Sensitive information provided in environment variables**
  Detects when functions contain environment variables (such as MYSQL_PASSWORD) that expose sensitive information.

- **435: Private keys stored in function**
  Detects private keys in functions.

- **436: Unbounded service access**
  Detects functions with permission to run all actions on all services and their resources.

- **437: Overly permissive service access**
  Detects functions with permission to run all actions on one or more services.

- **438: Broad resource access**
  Detects functions that granted access to all resources in one or more services.

- **439: Suspicious function actions**
  Detects functions with permission to run actions that are used in exploits and attacks. Includes things like cloudtrail:StopLogging, cloudtrail:UpdateTrail that allow disabling and changing the output of CloudTrail logging.

- **440: Unused service API with information disclosure risk**
  Detects functions with permissions to unused APIs that could allow information disclosure.

- **441: Unused service API with data leakage risk**
  Detects functions with permissions to unused APIs that could leak data.

- **442: Unused service API with data tampering risk**
  Detects functions with permissions to unused APIs that could allow data tampering.

- **443: Unused service API with lateral movement risk**
  Detects functions with permissions to unused APIs that could allow an attacker to move laterally.

- **444: Unused service API with denial of service risk**
  Detects functions with permissions to unused APIs that could facilitate a denial of service attack.

- **445: Unused service API with information exfiltration risk**
  Detects functions with permissions to unused APIs that could allow data exfiltration.

- **446: Unused service API with persistent access risk**
  Detects functions with permissions to unused APIs that allow persistent access.

- **447: Unused service API with privilege elevation risk**
Detects functions with permissions to unused APIs that allow privilege elevation.

Scanning serverless functions

Configure Prisma Cloud to periodically scan your serverless functions. Function scanning is handled by Defender. Console dynamically selects an available Defender to execute the scan job.

**STEP 1** | Open Console.

**STEP 2** | Go to **Defend > Vulnerabilities > Functions**.

**STEP 3** | Click **Add serverless account**.

**STEP 4** | In the dialog, enter the following settings:

1. In **Provider**, select **AWS Lambda**.
2. Specify a Region.
3. Specify a function name or pattern. Wildcards are supported. See the table at the top of the page for the supported syntax and examples.
4. Select or create credentials so that Prisma Cloud can access your account. Specify either an AWS IAM user credential (access key ID and secret access key) or IAM role.
5. Specify a cap for the number of functions to scan. Prisma Cloud scans the X most recent functions, where X is the cap value. To scan all functions, set the cap to 0.
6. Click **Add**.

**STEP 5** | Click the save button.

**STEP 6** | To view the scan report, go to **Monitor > Compliance > Functions**.
Windows compliance checks

Windows compliance checks were developed by Prisma Cloud Labs. They can be enabled in your host compliance policy.

Create Windows host compliance rules in Defend > Compliance > Hosts. In the new rule dialog, select Windows host from the Types drop-down list.

The following checks are supported:

- **200001: Verify Windows Defender antivirus is running**
  - Microsoft's built in Windows Defender antivirus service is running.

- **200002: Verify Windows Defender antivirus is enabled**
  - Microsoft's built in Windows Defender service antivirus, anti-malware, and anti-spyware features are enabled

- **200003: Verify Windows Defender always-on protection is enabled**
  - Always-on protection consists of real-time protection, behavior monitoring, and heuristics to identify malware based on known suspicious and malicious activities.

- **200004: Verify antivirus signatures match defined frequency**
  - Windows antivirus signatures are overdue based on your frequency policy.
• **200005: Verify antivirus signatures are up-to-date** --
   Windows antivirus signatures list must be updated within the last 14 days. If 14 days elapse without an update, signatures are stale. This interval is required to be effective against current threats.

• **200006: Verify anti-spyware signatures match defined frequency** --
   Windows anti-spyware signatures are overdue based on your frequency policy.

• **200007: Verify anti-spyware signatures are up-to-date** --
   Windows anti-spyware signatures list must be updated within the last 14 days. If 14 days elapse without an update, signatures are stale. This interval is required to be effective against current threats.

• **200201: Verify Windows Defender Control Flow Guard (CFG) is enabled** --
   Control Flow Guard (CFG) is a platform security feature that combats memory corruption vulnerabilities. By placing tight restrictions on where an application can execute code, CFG makes it harder for exploits to execute arbitrary code through vulnerabilities, such as buffer overflows. This check is applicable to Windows Server 2019 only.

• **200202: Verify Windows Defender Data Execution Prevention (DEP) is enabled** --
   Data Execution Prevention (DEP) monitors memory to stop malicious code from running. It monitors all processes and services and stops a program if it isn’t running correctly in memory. This check is applicable to Windows Server 2019 only.

• **200203: Verify Windows Defender Address Space Layout Randomization (ASLR) is enabled** --
   Address space layout randomization (ASLR) prevents exploitation of memory corruption vulnerabilities. It prevents an attacker from reliably jumping to an exploited function in memory by randomly arranging the position (address) of the stack, heap, and loaded libraries. This check is applicable to Windows Server 2019 only.

• **200300: Verify Windows Firewall public profile is enabled** --
   This setting is applied when a connection to a domain is made through a public network, such as at an airport, hotel, or coffee shop. Since the security of these networks is unknown and not really controlled by the user running the computer, it is suggested that the Public network profile of settings be more restrictive than either the Domain network or Private network.

• **200400: Verify Windows Update is enabled** --
   Windows Update is a service which automates downloading and installing Microsoft Windows software updates.

• **200401: Verify Windows Update is set to automatically install** --
   Verify that Windows is configured to automatically download and install updates at a regular interval.

---

- If **Windows Defender antivirus is not installed or running**, all Windows Defender related checks (200001, 200002, 200003, 200201, 200202, 200203) fail with the following cause: “Windows Defender antivirus service is not installed/running”.

- Although checks 200004/5 and 200006/7 look similar, they clarify the root cause of the issue when assessed separately. Checks 200004/6 verify the update frequency policy, while 200005/7 verify that signatures are actually up-to-date. Checks 200004/6 show whether the defined frequency is suboptimal (greater than 14 days), while checks 200005/7 show if there was a failure to update the signatures according to the defined policy (whether it’s 14 days or some other interval).

- If no definition files (signatures) are available, checks 200004 and 200006 fail with the following cause: “Windows Defender definition files are not available”. Definitions can be removed with the following command:

```
"%ProgramFiles%\Windows Defender\MpCmdRun.exe" --removedefinitions
```
Custom compliance checks

Custom image checks give you a way to write and run your own compliance checks to assess, measure, and enforce security baselines in your environment. Although Prisma Cloud supports OpenSCAP and XCCDF, these frameworks are complicated, and they can be overkill when all you want to do is run a simple check. Prisma Cloud lets you implement your own custom image checks with simple scripts.

A custom image check consists of a single script. The script’s exit code determines the result of the check, where 0 is pass and 1 is fail. Scripts are executed in the container’s default shell. For many Linux container images, the default shell is bash, but that’s not always the case. For Windows container images, the default shell is cmd.exe.

If you want to use a specific shell, or if your default shell is in a non-standard location, use the shebang interpreter directive at the top of your compliance check to specify the path to the executable. For example, the following directive specifies that the Linux Bourne-again (bash) shell should parse and interpret the compliance check.

```
#!/bin/bash
```

Defender runs the compliance checks inside a container instantiated from the image being scanned. Due to risks associated with running arbitrary code, all compliance checks are executed in a restricted sandboxed environment.

Every compliance check in the system has a unique ID. Custom image checks are automatically assigned an ID, starting with the number 9000. As new custom checks are added, they are automatically assigned the next available ID (9001, 9002, and so on).

If a new rule with custom compliance checks is added, or an existing rule is updated with a new custom compliance check, Prisma Cloud drops the cached scan results for registries, and rescans registry images. In a scaled-out environment with large registries, repeated changes to custom compliance checks could have a negative impact on Prisma Cloud’s performance.

Creating a new custom image check

The flow for writing and operationalizing a custom image check is:

- Write a custom image check.
- Create a new compliance rule that includes your custom image check, and specifies what action to take when the check fails (ignore, alert, block).

**STEP 1 | Open Console**

**STEP 2 | Write a new custom check.**
1. Go to **Defend > Compliance > Custom**.
2. Click **Add check**.
3. Enter a name and description.
4. Specify the severity of the compliance issue.
5. Enter a **script**.
6. Click **Save**.

**STEP 3 | Update the compliance policy to run your check.**
1. Go to **Defend > Compliance > Containers and Images**.
2. Click Add rule.
3. Enter a rule name.
4. Under Compliance actions, narrow the compliance checks displayed. On the All types drop-down list, select Custom > Image.
   You should see a list of custom checks you've implemented, starting with ID 9000.
5. Select an action for your custom check (Ignore, Alert, or Block).
6. Click Save.

STEP 4 | Validate your setup by reviewing the compliance reports under Monitor > Compliance.

Example scripts

The following example scripts show how to run some basic checks, such as checking file permissions. Use them as starting point for your own scripts. Any special utilities or programs required by your script must be installed in the image being evaluated.

File permissions (Linux)
The following script checks the permissions for the /bin/busybox file. Assuming busybox is installed in your image, this check should pass.

```bash
if [ $(stat -c %a /bin/busybox) -eq 755 ]; then
  echo 'test permission failure' && exit 1;
fi
```

File exists (Linux)
The following script checks if /tmp/foo.txt exists in the container file system. If it doesn't exist, the check fails.

```bash
if [ ! -f /tmp/foo.txt ]; then
  echo "File not found!"
  exit 1
fi
```

User exists (Linux)
The following script checks if the user John exists. If the user exists, the check passes. Otherwise, it fails.

```bash
if grep -Fxq "John" /etc/passwd
then
  echo yes
else
  echo "user not found!"
  exit 1
fi
```

File exists (Windows)
The following script checks if C:\Users exists. If it does, the check passes.

```bash
IF EXIST C:\Users Echo test permission failure && exit 1
```
**File does not exist (Windows)**

This check is the inverse of the previous check. The script checks if `C:\Users` doesn't exist. If it doesn't exist, the check passes.

```bash
IF NOT EXIST C:\Users Echo test permission failure && exit 1
```
Extensible compliance checks

Prisma Cloud lets you import custom security checklists and evaluate them against your container images. Custom checklists complement the predefined compliance checks already provided in the default Prisma Cloud installation.

Prisma Cloud can consume Extensible Checklist Configuration and Description Format (XCCDF) benchmarks and checklists. XCCDF is an open standard defined by the National Institute of Standards and Technology (NIST) that automates the assessment of an application’s configuration and the testing of its compliance to security rules. Checklists are expressed in XML. For more information about XCCDF, see the specification.

For example, your organization might require that each container running a Payment Card Industry (PCI) workload embed a manifest file that describes the team responsible for the chain of custody of the credit card data. With Prisma Cloud and XCCDF, you can configure a rule to check for compliance to this requirement and take action (alert or block) when a container is missing this file. Your audit team can then use Console as the central point for monitoring and enforcing compliance to this setting.

To use XCCDF with Prisma Cloud, your environment must meet the following requirements:

- Your XCCDF benchmark must be encapsulated in a SCAP datastream. Datastreams are defined in the SCAP v1.2 specification. For more information, see the technical specification. If your benchmark is not in datastream format, you can convert it. For more information, see Procedure: Converting XCCDF checklists to datastream format.
- Your container images must be derived from an RPM-based distribution, such as CentOS, Fedora, or Red Hat Enterprise Linux (RHEL).
- Your base image must have glibc version 2.17 or later. Distributions that ship with glibc 2.17 or later include CentOS 7, RHEL 7, and Fedora 20. You can run the following command in your container to get the version of glibc:

  ```
  $ ldd --version
  ldd (GNU libc) 2.17
  ```

Getting started

To get started with XCCDF, first locate an `.xml` file.

Here is a sample to get you started:

**STEP 1** | Install a Fedora container.

**STEP 2** | Run the following:

  ```
  $ dnf install scap-security-guide
  ```

**STEP 3** | Grab any xml file under `/usr/share/xml/scap/ssg/content/`, specifically the `-ds.xml` ones. Use these as a template going forward.

Importing an XCCDF Datastream

Set up XCCDF compliance checks.

**STEP 1** | Open `twistlock.cfg` for editing.
STEP 2 | Enable SCAP by setting `SCAP_ENABLED` to `true`.

```
SCAP_ENABLED=true
```

STEP 3 | Load the new configuration setting.

If you have not installed Prisma Cloud yet, follow the regular installation procedure. Otherwise, follow the upgrade procedure for Console, which loads the new configuration without impacting the rest of Console's data or state.

STEP 4 | If Prisma Cloud has already been installed, redeploy your Defenders.

If you have deployed Defenders in a Kubernetes or OpenShift cluster, perform the following steps:

1. SSH to the node where Defender runs registry scanning.
2. Retrieve the `openscap.tar.gz` distribution from the Prisma Cloud API and extract into Defender's working directory.

```
$ curl -k -u "<TWISTLOCK_CONSOLE_ADMIN>" \\n    https://<twistlock_console>:8083/api/v1/util/openscap.tar.gz \\
-o openscap.tar.gz

$ sudo tar -xvf openscap.tar.gz -C /var/lib/twistlock/utils/openscap
```

3. Repeat on all nodes where Defender performs SCAP scanning.

STEP 5 | If you have deployed Defenders within a Kubernetes / OpenShift cluster perform the following steps:

1. SSH onto the node that the Defender performing registry scanning is running
2. Pull the `openscap.tar.gz` distribution via your Console’s API and extract into the Defenders working directory.

```
$ curl -k -u "<twistlock_console_admin>" https://
    <twistlock_console>:8083/api/v1/util/openscap.tar.gz \
$ sudo tar -xvf openscap.tar.gz -C /var/lib/twistlock/utils/openscap
```

3. Repeat on all nodes in which the Defender performs SCAP scanning.

STEP 6 | Open Console, and go to Manage > System > SCAP.

STEP 7 | Click the +DATASTREAM button, and select a datastream to upload.

A benchmark, with its available profiles, is added to the table. Prisma Cloud assigns a vulnerability ID for each profile, which can be used to set up your policies. Vulnerability IDs for benchmark profiles start at 4000.

Setting your policy

In order to process a checklist in a profile, set up a new policy.

STEP 1 | Open Console, then go to Defend > Compliance.

STEP 2 | Click the +COMPLIANCE RULE button to create a new rule.

1. In Rule Name, enter an identifier for your rule.
2. Select the profile you want to process, and then set an action (NONE, ALERT, or BLOCK) when a rule in your checklist fails.
By default, XCCDF profiles are assigned a vulnerability ID that start at 4000.

3. Click SAVE to activate your rule.

**STEP 3|** Check the results of an image scan.

1. Go to Monitor > Compliance > Images.
2. Select an image from the table.

Any rule that does not pass is listed in this tab. The severity of an issue is determined by the rule in the benchmark file.

### Converting XCCDF checklists to datastream format

If your XCCDF checklist complies to version 1.2 of the SCAP specification, but it is not in datastream format, you must first convert it to datastream format before importing it into Console.

To convert an xccdf, cpe, cpe-dictionary, or oval checklist to datastream format:

**STEP 1|** Install the oscap utility. For more information, see the OpenSCAP documentation.

```
$ yum install openscap-scanner
```

**STEP 2|** Verify that oscap was installed successfully.

```
$ oscap version
```

**STEP 3|** Convert your checklist.

Assuming your checklist is named myChecklist-{cps-dictionary | cps-oval | oval | xccdf}.xml, run the following command:

```
$ oscap ds sds-compose myChecklist-xccdf.xml myChecklist-ds.xml
```

*Your checklist name should not contain any spaces or parenthesis.*

The resulting output is a datastream that can be directly imported into Prisma Cloud.
Trusted images

Trusted images is a security control that lets you declare, by policy, which registries, repositories, and images you trust, and how to respond when untrusted images are started in your environment.

Image provenance is a core security concern. In NIST SP 800-190 (Application Container Security Guide), the section on countermeasures for major risks (Section 4) says:

"Organizations should maintain a set of trusted images and registries and ensure that only images from this set are allowed to run in their environment, thus mitigating the risk of untrusted or malicious components being deployed."

Container runtimes, such as Docker Engine, will run, by default, any container you ask it to run. Trusted images lets you explicitly define which images are permitted to run in your environment. If an untrusted image runs, Prisma Cloud emits an audit, raises an alert, and optionally blocks the container from running.

Modern development has made it easy to reuse open source software. Pulling images from public registries, such as Docker Hub, is simple and fast, and it removes a lot of friction in operations. Retrieving and executing software with such ease, however, runs contrary to many organizations’ security policies, which mandate that software originates from approved providers and distribution points. The Trusted Images rule engine lets you specify registries, repositories, and images that are considered trustworthy.

Trusted images isn’t supported for nested virtualization, also known as Docker-in-Docker, on platforms such as DC/OS Kubernetes.

Feature overview

Trusted images is disabled by default. To enable it, go to Defend > Compliance > Trusted Images > Policy.

After enabling the feature, you must specify the images you trust. Declare trust using objects called trust groups. Trust groups collect related registries, repositories, and images in a single entity. Then use those entities for writing policy rules.

The default policy consists of a single rule that alerts on all images started in your environment. Build out your policy by writing new rules. Rules let you define:

- Explicitly allowed trust groups.
- Explicitly denied trust groups
- An action to take when an image isn’t trusted.

When a container starts in your environment, Defender assesses the event against your trust policy, and then acts accordingly. Rules in a policy are evaluated top-down. The criteria for matching an event to a rule is the hostname. When a matching rule is found, the rule is processed. No subsequent rules are processed. The first rule that matches the hostname holds the verdict for all images that can run on that host. If the image being started matches an explicitly denied trust group, the rule effect is applied. If an image doesn’t match either the list of explicitly allowed trust groups or explicitly denied trust groups, the rule effect is also applied.

Audits are created when the effect of a rule is alert or block. You can review audits in Monitor > Events. When reviewing audits, you can optionally add the image to a trust group to quickly adjust your policy and clean up false positives.

The Console UI provides a number of features to surface trust in your environment.

- Image scan reports have indicators in the report header to show whether an image is trusted or not. See:
  - Monitor > Compliance > Containers and Images
  - Monitor > Vulnerabilities > Images
A dedicated page in Monitor > Compliance > Trusted Images, shows a snapshot of all running images in your environment and their trust status. The table is updated at scan-time, which is once per 24 hours by default. However, the page lets you force a re-scan and refresh the results.

Also note that updated policies aren’t automatically reflected in the view. If you change a rule in your Trusted Images policy, re-scan the images in your environment to update the view.

Trust indicators in the Console UI

Badges are shown throughout Console to clearly delineate between trusted and untrusted images. The following badges are used:

- **Trusted** — Explicitly trusted by a user-defined rule (Defend > Compliance > Trusted Images > Policy).

- **Untrusted**. An image is considered untrusted if it’s untrusted on at least one host.

Badges are shown in the following pages:

- Scan reports (click on a row in the table to open an image scan report):
  - Monitor > Compliance > Containers and Images
  - Monitor > Vulnerabilities > Images
  - Snapshot of all running containers and their trust status. The table is updated at scan-time.
  - Monitor > Compliance > Trusted Images

Events Viewer

Prisma Cloud generates an audit for every image that is started in your environment, but fails to comply with your trust policy. Audits can be reviewed under Monitor > Events > Trust Audits. When reviewing audits, you can optionally add the image to a trust group.

Establishing trust with rules

Prisma Cloud monitors the origin of all containers on the hosts it protects.

Policies are built on rules, and rules reference trust groups. Trust groups are collections of related registries and repositories.

Trust is established by one of the following factors:

- Point of origin (registry and/or repository),
- Base layer(s).

*Trusting images by image tag isn’t supported.*

Establishing trust by registry and repository

Prisma Cloud lets you specify trust groups by registry and repository. If you specify just a registry, all images in the registry are trusted.
Establishing trust by base layer

Images can have layers in common. If your organization builds and approves specific base images for use in containerized apps, then you can use this mechanism to enforce compliance.

For example, consider the ubuntu:16.04 image. If you run `docker inspect`, the layers are:

```
"Layers": [
 "sha256:a94e0d5a7c404d0e6fa15d8cd4010e69663bd8813b5117fba7d1365a73656df9",
 "sha256:88888b9b1b5b7bce5db41267e669e6da63ee95736cb904485f96f29be648fda",
 "sha256:52f389ea437ebf419d1c9754d0184b57ed545c951666ee86951d9f6af8d6035e",
 "sha256:52a7ea2bb533dc2a91614795760a67fb807561e8a588204c4858a300074c082b",
 "sha256:db584c622b50c3b8f9b8b94c270cc5f235e5f23ec4aacea8ce67a8c16e0fbd"
]
```

Now consider a new image, where ubuntu:16.04 is the base OS. The following Dockerfile shows how such an image is constructed:

```
FROM ubuntu:16.04
RUN apt-get update
ADD hello.txt /home/hello.txt
WORKDIR /home
```

After building the image, and inspecting the layers, you can see that both images share the same first five layers.

```
"Layers": [
 "sha256:a94e0d5a7c404d0e6fa15d8cd4010e69663bd8813b5117fba7d1365a73656df9",
 "sha256:88888b9b1b5b7bce5db41267e669e6da63ee95736cb904485f96f29be648fda",
 "sha256:52f389ea437ebf419d1c9754d0184b57ed545c951666ee86951d9f6af8d6035e",
 "sha256:52a7ea2bb533dc2a91614795760a67fb807561e8a588204c4858a300074c082b",
 "sha256:db584c622b50c3b8f9b8b94c270cc5f235e5f23ec4aacea8ce67a8c16e0fbd",
 "sha256:29d16833b7ef90f9c63466967c58330bd5134dfe1fafe21bb8c729e69084058f",
 "sha256:1d622b0ae83a00049754079a2bbbf7841321a24cf2937ea2d57e6e3b562ab9"
]
```

Creating trust groups manually

Trust groups are collections of related registries and repositories. Policies are built on rules, and rules reference trust groups.

When setting up a trust group, you can explicitly specify registries and repositories to trust.
Prisma Cloud supports leading and trailing wildcard matches as described in the following table:

<table>
<thead>
<tr>
<th>Match type</th>
<th>Registry only</th>
<th>Repository only</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact match</td>
<td>reg</td>
<td>repo</td>
<td>reg/repo</td>
</tr>
<tr>
<td>Suffix match</td>
<td>reg*</td>
<td>repo* repo/*</td>
<td>reg/repo* reg/repo/*</td>
</tr>
<tr>
<td>Prefix match</td>
<td>*reg</td>
<td>*repo</td>
<td>*reg/repo</td>
</tr>
<tr>
<td>Both suffix &amp; prefix</td>
<td><em>reg/</em></td>
<td><em>repo/</em></td>
<td><em>reg/repo/</em></td>
</tr>
</tbody>
</table>

Examples:
- All repos under a parent repo:
Creating trust groups based on what’s running in your environment

STEP 1 | Open Console.

STEP 2 | Go to Defend > Compliance > Trusted Images > Trust Groups.

STEP 3 | Click Add New Group.

STEP 4 | In Name, enter a group name.

STEP 5 | In Type, select how you want to specify an image.

By Image:

There are two ways to specify images:

Method 1 - Choose from a list of containers already running in your environment. In the table, select the images you trust, and click Add To Group.

Method 2 - Specify a registry address and/or repository, and click Add To Group. If you specify just a registry, then all images in the registry are trusted. If you specify just a repository, the registry is assumed to be Docker Hub.

As you add entries to the trust group, the entries are enumerated in the Group Images table at the bottom of the dialog.

By Base Layer:

Prisma Cloud lets you import the base layers from any image in your environment. If Prisma Cloud has seen and scanned an image, it is available in the Image drop-down list.

Select an image, import it, and then review the SHA256 hashes for the base layers. For example, if the secteam/ubuntu:16.04 is your trusted base OS, select it from the Image drop-down list, and click Import.

STEP 6 | Click Save.
All. Then you could select all images on these hosts and add them to the trust group. Later, you could create a rule for these prod hosts by specifying the host resource as "prod**" (wildacrd is necessary here), and add the new trust group to the allowed groups.

Writing policy

After declaring the images you trust with trust groups, write the rules that make up your policy.

Prisma Cloud evaluates the rules in your trusted images policy from top to bottom until a match is found based on host name. If the image being started in your environment matches a host name in a rule, Prisma Cloud applies the actions in the rule and stops processing any further rules. If no match is found, no action is taken.

You should never delete the default rule, Default - alert all, and it should always be the last rule in your policy. The default rule matches all hosts (*). It serves as a catchall, alerting you to images that aren’t captured by any other rule in your policy.
If you delete all rules in your policy, including the default rule, all images in your environment will be considered trusted.

Assuming the default rule is in place, policy is evaluated as follows:

- **A rule is matched** — The rule is evaluated.
- **A rule is matched, but no trust group is matched** — The image is considered untrusted. Prisma Cloud takes the same action as if it were explicitly denied.
- **No rule match is found** — The default rule is evaluated, and an alert is raised for the image that was started. The default rule is always matched because the hostname is set to a wildcard.

**STEP 1** | Open Console.

**STEP 2** | Go to **Defend > Compliance > Trusted Images > Policy**.

**STEP 3** | Click **Add Rule**.

**STEP 4** | Enter a rule name.

**STEP 5** | In **Effect**, specify how Prisma Cloud responds when it detects an explicitly denied image starting in your environment. This action is also used when a rule is matched (by hostname), but no trust group in the rule is matched.  
  - **Ignore** — Do nothing if an untrusted image is detected.
  - **Alert** — Generate an audit and raise an alert.
  - **Block** — Prevent the container from running on the affected host. Blocking isn’t supported for Windows containers.

**STEP 6** | Specify the rule’s scope.  
By default, the rule applies to all hosts in your environment. Pattern matching is supported.

**STEP 7** | Explicitly allow or deny images by trust group.

**STEP 8** | (Optional) Append a custom message to the block action message.  
Custom messages help the operator better understand how to handle a blocked action. You can enhance Prisma Cloud’s default response by appending a custom message to the default message. For example, you could tell operators where to go to open a ticket.

**STEP 9** | Click **Save**.

Your rule is added to the top of the rule list. Rules are evaluated from top to bottom. The rule at the top of the table has the highest priority. The rule at the bottom of the table should be your catch-all rule.
Host scanning

Prisma Cloud scans all hosts where Defender is installed. Defender scans hosts for the following types of vulnerabilities:

- **Host configuration**: Vulnerabilities in the host setup.
- **Docker daemon configuration**: Vulnerabilities that stem from misconfiguring your Docker daemons. Docker daemon derives its configuration from various files, including /etc/sysconfig/docker or /etc/default/docker. Misconfigured daemons affect all container instances on a host.
- **Docker daemon configuration files**: Vulnerabilities that arise from improperly securing critical configuration files with the correct permissions.
- **Docker security operations**: Recommendations and reminders for extending your current security best practices to include containers.

Reviewing host scan reports

Prisma Cloud lets you filter the displayed hosts by searching for specific hosts or by collection. Collections support AWS tags. When creating new collections, specify the tags you want to use for filtering in the **Labels** field.

You can filter the displayed hosts by searching for specific hosts or by choosing a collection. Collections support AWS tags. When creating a new collection, add the tags you want to use for filtering to the **Labels** field.

**STEP 1** | Open Console, then go to **Monitor > Compliance > Hosts > Running Hosts**.

**STEP 2** | Click on a host in the list.

A report for the compliance issues on the host is shown.
### Host Details

D: ion-t.ccto-sancbox.internal
OS distribution: Ubuntu 16.04.2 LTS
Modified: Aug 11, 2017 12:40:20 PM

<table>
<thead>
<tr>
<th>Id</th>
<th>Category</th>
<th>Type</th>
<th>Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>218</td>
<td>Docker</td>
<td>daemon config</td>
<td>medium</td>
<td>Disable Userland Proxy (CIS 2.18)</td>
</tr>
<tr>
<td>214</td>
<td>Docker</td>
<td>daemon config</td>
<td>medium</td>
<td>Enable live restore (CIS 2.14)</td>
</tr>
<tr>
<td>213</td>
<td>Docker</td>
<td>daemon config</td>
<td>medium</td>
<td>Disable operations on legacy registry (v1) (CIS 2.13)</td>
</tr>
<tr>
<td>212</td>
<td>Docker</td>
<td>daemon config</td>
<td>medium</td>
<td>Configure centralized and remote logging (CIS 2.12)</td>
</tr>
<tr>
<td>211</td>
<td>Docker</td>
<td>host config</td>
<td>medium</td>
<td>Use authorization plugin (CIS 2.11)</td>
</tr>
<tr>
<td>115</td>
<td>Docker</td>
<td>host config</td>
<td>medium</td>
<td>Audit Docker files and directories - /usr/bin/docker-runc (CIS 1.13)</td>
</tr>
<tr>
<td>114</td>
<td>Docker</td>
<td>host config</td>
<td>medium</td>
<td>Audit Docker files and directories - /usr/bin/docker-containerd (CIS 1.12)</td>
</tr>
<tr>
<td>112</td>
<td>Docker</td>
<td>host config</td>
<td>medium</td>
<td>Audit Docker files and directories - /etc/default/docker (CIS 1.10)</td>
</tr>
<tr>
<td>111</td>
<td>Docker</td>
<td>host config</td>
<td>medium</td>
<td>Audit Docker files and directories - docker.sock (CIS 1.9)</td>
</tr>
<tr>
<td>110</td>
<td>Docker</td>
<td>host config</td>
<td>medium</td>
<td>Audit Docker files and directories - docker.service (CIS 1.8)</td>
</tr>
<tr>
<td>28</td>
<td>Docker</td>
<td>daemon config</td>
<td>medium</td>
<td>Enable user namespace support (CIS 2.8)</td>
</tr>
<tr>
<td>27</td>
<td>Docker</td>
<td>daemon config</td>
<td>medium</td>
<td>Set default ulimit as appropriate (CIS 2.7)</td>
</tr>
</tbody>
</table>

All vulnerabilities identified in the latest host scan can be exported to a CSV file by clicking on the **CSV** button in the top right of the table.
VM image scanning

Prisma Cloud can scan the virtual machine (VM) images in your AWS environment for the following types of vulnerabilities:

- **Host configuration**: Vulnerabilities in the VM image setup.
- **Docker daemon configuration**: Vulnerabilities that stem from misconfiguring your Docker daemon. The Docker daemon derives its configuration from various files, including `/etc/sysconfig/docker` or `/etc/default/docker`.
- **Docker daemon configuration files**: Vulnerabilities that arise from setting incorrect permissions on critical configuration files.
- **Docker security operations**: Recommendations and reminders for extending your current security best practices to include containers.
- **Linux configuration**: Compliance of Linux hosts. For example, ensure mounting of the `hfs` filesystem is disabled.

Reviewing VM image scan reports

To view the health of the VM images in your environment:

**STEP 1** | Open Console, then go to Monitor > Compliance > Hosts > VM images.

**STEP 2** | Click on a VM image on the list.

A report for the compliance issues on the VM image is shown.
All compliance issues identified in the latest VM image scan can be exported to a CSV file by clicking on the CSV button in the top right of the table.
Detect secrets

Prisma Cloud can detect sensitive information that is improperly secured inside images and containers. Scans can detect embedded passwords, login tokens, and other types of secrets. To detect improperly secured secrets, add the following checks to your compliance policy.

**Compliance check ID 424**

This check detects sensitive information provided in environment variables of image. The data so provided can be easily exposed by running `docker inspect` on the image and thus compromising privacy.

**Example**

```
```

**Response**

```
Sending build context to Docker daemon 2.048 kB
Step 1/2 : FROM alpine:latest
  ---> 88e169ea8f46
Step 2/2 : ENV PASSWORD = secret
  ---> Using cache
  ---> 8f3627bc339b
Error: [Prisma Cloud] Image operation blocked by policy: (No secrets attached), violates: The environment variable PASSWORD contains sensitive data
```

**Compliance check ID 425**

This check detects private keys stored in an image.

**Example**

Navigate to Defend > Compliance. Add a new compliance rule to block running an image with private key in it.
$ docker --tlsverify -H aqsa.c.cto-sandbox.internal:9998 build -t aqsa:secretv1

Sending build context to Docker daemon 5.632 kB
Step 1/2 : FROM alpine:latest
  ---> 88e169ea8f46
Step 2/2 : ADD private_key /root/.ssh/id_rsa
  ---> Using cache
  ---> c6e8e2496663
Error: [Prisma Cloud] Image operation blocked by policy: (No secrets attached), violates: Private keys stored in image /root/.ssh/id_rsa

Set the action to ALERT instead of BLOCK, then go to Monitor > Compliance after running the image. Click on the image under Images tab.

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Severity</th>
<th>Description</th>
</tr>
</thead>
</table>
Compliance check ID 597

This check detects sensitive information provided in environment variables of container.
Cloud discovery

It's difficult to ensure that all your apps running on all the different types of cloud services are being properly secured. If you're using multiple cloud platforms, you might have many separate accounts per platform. You could easily have hundreds of combinations of providers, accounts, and regions where cloud native services are being deployed.

Cloud Platforms discovery helps you find all cloud-native services being used in AWS, Azure, and Google Cloud, across all regions, and across all accounts. Cloud Provider discovery continuously monitors these accounts, detects when new services are added, and reports which services are unprotected. It can help mitigate your exposure to rogue deployments, abandoned environments, and sprawl.

Cloud Platforms discovery offers coverage for the following services.

**Registries:**
- AWS
- Azure
- Google Cloud

**Serverless functions:**
- AWS
- Azure
- Google Cloud

**Managed platforms:**
- AWS ECS
- AWS EKS
- Azure Kubernetes Service (AKS)
- Azure Container Instances (ACI)
- Google Kubernetes Engine (GKE)

**Virtual machines:**
- AWS EC2

1 One-click protection is currently not yet available for these services. One-click protection lets you deploy Prisma Cloud to protect a resource directly from the scan results page.

Configuring cloud platforms discovery

Set up Prisma Cloud to scan your cloud platform accounts for cloud-native resources and services. Then configure Prisma Cloud to protect them with a single click.

**Prerequisites:**
- For AWS, you've got a service account with following minimum permissions policy:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": ["lambda:ListFunctions","ecr:DescribeRepositories"]
      }
   ]
}
```
STEP 1 | Open Console.

STEP 2 | Go to Defend > Compliance > Cloud Platforms.

STEP 3 | Select the accounts to scan with the Discovery checkbox. If there are no accounts in the table, add one in the credentials store.

STEP 4 | Click Save.

STEP 5 | Review the scan report.

1. Go to Monitor > Compliance > Cloud Discovery to see the scan report in tabular format.
2. Go to Radar and select Cloud to see the scan report in a visual format.
3. Click Protect for the entities you want Prisma Cloud to scan for vulnerabilities.
   When you click Protect, a new scan rule is proposed. Select the appropriate credential, tweak the scan rule as desired, then click Add.
4. Scan reports can viewed under Monitor > Vulnerabilities > {Registry|Functions}.

Configuring cloud compliance scans

Prisma Cloud can assess your AWS account against the CIS Amazon Web Services Foundations v1.2.0 benchmark. This benchmark provides prescriptive guidance for configuring security options for a subset of Amazon Web Services. It has four sections:

- Identity and Access Management
- Logging
- Monitoring
- Networking

As with all scanning in Prisma Cloud, there are two flows:

- Periodic scanning, which is configurable in Manage > System > Scan, and set to a default of once every 24 hours.
- Manual scanning, which lets you force a scan immediately by pressing the Scan button in Monitor > Compliance > Cloud Compliance.

Prerequisites: 

- For Azure, you've created a service principal with the reader role.
- You have a service account with the following minimum permissions policy.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "iam:ListPolicies",
                "iam:GenerateCredentialReport",
                "iam:GetPolicyVersion",
                "iam:GetAccountPasswordPolicy",
                "s3:GetBucketLogging",
                "iam:ListEntitiesForPolicy",
                "logs:DescribeMetricFilters",
                "sns:ListSubscriptions",
                "cloudtrail:GetEventSelectors",
                "s3:GetBucketAcl",
                "config:DescribeConfigurationRecorderStatus",
                "s3:GetBucketPolicy",
                "iam:ListVirtualMFADevices",
                "cloudtrail:DescribeTrails",
                "kms:ListKeys",
                "config:DescribeConfigurationRecorders",
                "s3:ListAllMyBuckets",
                "kms:ListAliases",
                "cloudwatch:DescribeAlarms",
                "iam:ListUsers",
                "iam:GetCredentialReport",
                "s3:GetBucketLocation",
                "iam:GetAccountSummary"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "sns:ListSubscriptionsByTopic",
                "kms:GetKeyRotationStatus",
                "cloudtrail:GetTrailStatus",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies"
            ],
            "Resource": [
                "arn:aws:iam::*:user/*/",
                "arn:aws:cloudtrail::*:trail/*/",
                "arn:aws:kms::*:key/*/",
                "arn:aws:sns::*:*
            ]
        }
    ]
}
```

**STEP 1** | Open Console.

**STEP 2** | Go to Defend > Compliance > Cloud Platforms.
STEP 3 | Select the accounts to scan with the Compliance checkbox. If there are no accounts in the table, add one in the credentials store. Compliance checks are only available for AWS.

STEP 4 | Choose the compliance checks to enable. By default, all critical and high checks are set to alert.

STEP 5 | Click Save.

STEP 6 | Go to Monitor > Compliance > Cloud Compliance to review the scan reports in tabular format. Alternatively, go to Radar, select Cloud, and click through the markers to explore the corresponding account’s compliance results.
Runtime defense

Runtime defense is the set of features that provide both predictive and threat based active protection for running containers. For example, predictive protection includes capabilities like determining when a container runs a process not included in the origin image or creates an unexpected network socket. Threat based protection includes capabilities like detecting when malware is added to a container or when a container connects to a botnet.

- Runtime defense
- Runtime defense for processes
- Runtime defense for networking
- Runtime defense for file systems
- Runtime defense for hosts
- Custom runtime rules
- Blocked containers
- Import and export individual rules
- Discrete blocking
- Runtime defense for AWS Fargate
- Incident Explorer
Runtime defense

Runtime defense is the set of features that provide both predictive and threat based active protection for running containers. For example, predictive protection includes capabilities like determining when a container runs a process not included in the origin image or creates an unexpected network socket. Threat based protection includes capabilities like detecting when malware is added to a container or when a container connects to a botnet.

Prisma Cloud introduced runtime defense all the way back in our 1.1 release. In releases since then, we’ve continuously added to the feature set. This article describes the current architecture.

Prisma Cloud has distinct sensors for file system, network, and process activity. Each sensor is implemented individually, with its own set of rules and alerting. The runtime defense architecture is unified to both simplify the admin experience and to show more detail about what Prisma Cloud automatically learns from each image. Runtime defense has two principle object types: models and rules.

Container Models

Models are the results of the autonomous learning that Prisma Cloud performs every time we see a new image in an environment. A model is the ‘allow list’ for what a given image should be doing, across all runtime sensors. Models are automatically created and maintained by Prisma Cloud and provide an easy way for administrators to view and understand what Prisma Cloud has learned about their images. For example, a model for an Apache image would detail the specific processes that should run within containers derived from the image and what network sockets should be exposed.

Navigate to Monitor > Runtime > Container Models. Click on the image to view it’s model.

There is a 1:1 relationship between models and images; every image has a model and every model applies to a single unique image. For each image, a unique model is created and mapped to the image digest. So, even if there are multiple images with the same tags, Prisma Cloud will create unique models for each.

Models are built from both static analysis (such as building a hashed process map based on parsing an init script in a Dockerfile ENTRYPOINT) and dynamic behavioral analysis (such as observing actual process activity during early runtime of the container). Models can be in one of 3 modes: Active, Archived, or Learning.
For containers in Kubernetes clusters, Prisma Cloud considers the image, namespace, cluster, and deployment (YAML) file when creating models.

- When the same image runs in multiple different clusters, Prisma Cloud creates separate models for each image in each cluster.
• When the same image runs in multiple different namespaces, Prisma Cloud creates separate models for each image in each namespace.
• When there are multiple running instances of an image in the same namespace, Prisma Cloud creates a single model.
• When there are multiple running instances of an image in the same namespace, but started from different deployment (YAML) files, Prisma Cloud creates multiple container models, one for each deployment.

Prisma Cloud shows you how models map to specific images. Go to Monitor > Runtime > Container Models, click a model in the table, and click the General tab.

Capabilities

Some containers are difficult to model. For example, Jenkins containers dynamically build and run numerous processes, and the profile of those processes changes depending on what's being built. Constructing accurate models to monitor processeses in containers that build, run, test, and deploy software is impractical, although other aspects of the model can still have utility. Prisma Cloud automatically detects known containers, and overrides one more aspects of the model with capabilities.

Capabilities are discrete enhancements to the model that tune runtime behaviors for specific apps and configurations. Rather than changing what's learned in the model, they modify how Prisma Cloud acts on observed behaviors.

For example, the following model for the Jenkins container is enhanced with the capability for writing and executing binaries.
Learning mode

Learning mode is the phase in which Prisma Cloud performs either static or dynamic analysis. Because the model depends on behavioral inputs, images stay in learning mode for 1 hour to complete the model. After this 1 hour, Prisma Cloud enters a 'dry run' period for 24 hours to ensure there are no behavioral changes and the model is complete. If during this 24 hours period, behavioral changes are observed, the model goes back to Learning mode for additional 24 hours. The behavioral model uses a combination of machine learning techniques and typically requires less than 1 hour of cumulative observation time for a given image (it might comprise of a single container running the entire learning period or multiple containers running for some time slice where the sum of the slices is 1 hour). During this period, only threat based runtime events (malicious files or connections to high risk IPs) are logged. Prisma Cloud automatically detects when new images are added anywhere in the environment and automatically puts them in learning mode.
- Relearn: You can relearn an existing model by clicking the **Relearn** button in the **Actions** menu. This is an additive process, so any existing static and behavioral modeling remains in place.
- Manual Learning: Users can manually alter the duration of learning at any time by starting and stopping the **Manual Learning** option in the **Actions** menu. This should be done with discretion because the model may or may not complete within the time period due to manual interruption. There is no time limit for manual learning and depends on user’s choice.
Active mode

Active mode is the phase in which Prisma Cloud is actively enforcing the model and looking for anomalies that violate it. Active mode begins after the initial 1 hour that the Learning mode takes to create a model. Because models are explicit allow lists, in enforcing mode, Prisma Cloud is simply looking for variances against the model. For example, if a model predicted that a given image should only run the foo process and Prisma Cloud observes the bar process has spawned, it would be an anomaly. Prisma Cloud automatically transitions models from learning mode into enforcing mode after the model is complete. During this period, runtime events are logged.

During the initial dry run period (the first 24 hours), model may switch automatically from Active mode to Learning mode depending on the behavioral changes observed, as mentioned above. This automatic switching only happens during the first 24 hours of model initiation. If violations are observed later on, they are logged as runtime alerts under Monitor > Runtime.

Archived mode

Archived mode is a phase that models are transitioned into after no containers are actively running them. Models persist in archived mode for 24 hours after being archived, after which point they're automatically removed by an internally managed garbage collection process. Archived mode essentially acts as a 'recycle bin' for models, ensuring that a given image does not need to go through learning mode again if it frequently starts and stops while also ensuring that the list of models does not continuously grow over time.

Models display all the learned data across each of the runtime sensors to make it easy to understand exactly what Prisma Cloud has learned about an image and how it will protect it. However, what if you need to customize the protection for a given image, set of images, or containers? That's the job of rules.

Rules

Rules provide administrative control over how Prisma Cloud uses the autonomous models to protect an environment. For example, if Prisma Cloud's model for the Apache image includes the process httpd, but you know that process bar will eventually need to run within it and you want to ensure that foo never runs, you can use a rule that applies to images named httpd to add bar to the allowed process list and add foo to the blocked process list.
# Create A New Runtime Rule

<table>
<thead>
<tr>
<th>Tab</th>
<th>Processes</th>
<th>Networking</th>
<th>File System</th>
<th>System Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Rule name**: apache_images

**Twistlock Advanced Threat Protection**: On

**Detect Kubernetes attacks**: Off

### Containers

- **Add a container**: *(x)*

### Images

- **lab-twistlock-com:5000/infra/portal_http:latest**: *(x)*
  - **Add an image**: *(x)*

### Hosts

- **Add a host**: *(x)*

### Labels

- **Add a label**: *(x)*
Rules allow administrators to add explicitly allow and explicitly blocked object to each sensor. Rules and models are evaluated together to create a resultant policy. The process for doing so is as follows:

\[ \text{model (which contains only allowed objects) + allowed objects from rule - blocked objects from rule} = \text{resultant policy} \]

From the previous example:

\[ \text{model (httpd) + allowed object from rule (bar) - blocked object from rule (foo) = httpd and bar are allowed and foo always is an anomaly regardless of the model} \]

To learn about rules ordering and pattern matching, refer to Rule ordering and pattern matching.

By default, Prisma Cloud has a single rule that simply says 'use the models'. As with every other rule within Prisma Cloud, you can easily customize these behaviors by creating new rules, applying them to the desired objects with our regex like filtering, and then ordering them properly. Rules are executed sequentially, and once a match is found for the 'applies to' conditions, the actions in that rule are enforced. Only a single rule is ever enforced for a given scenario; while rules are combined with models as described above, rules themselves are never combined.
VMWare Tanzu TAS

Runtime rules for VMWare Tanzu TAS apps can be scoped by app name and space ID. Specify values for app name and space ID in the **Labels** field of your runtime rules. This field is auto-populated with values from your environment.

```
pcf-application-name:<value>
pcf-space-id:<value>
```

**Create new runtime rule**

<table>
<thead>
<tr>
<th>Rule name</th>
<th>Enter the rule name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notes</td>
<td>Enter notes</td>
</tr>
</tbody>
</table>

**Scope**

- **Containers**: Specify a container
- **Images**: Specify an image
- **Namespaces**: Specify a namespace
- **Hosts**: Specify a host
- **Labels**: Specify a label

**Cloud Accounts IDs**: Specify an account ID

**General**

- **Processes**
- **Networking**
- **File System**
- **Custom Rules (0)**

**Basic monitoring**

- **Prisma Cloud Advanced Threat Protection**: On
- **Kubernetes attacks**: Off
- **Suspicious queries to cloud provider APIs**: Off

**Best practices**

One of the key goals is to minimize the amount of work customers have to do to manage runtime defense. For most customers, this means that only the default rule needs to be present and that Prisma Cloud will automatically create and manage models for it. Because the behavioral learning aspects of model creation have been significantly improved most customers will not need to create rules to change model behavior. Some exceptions may be in containers that are long running but change behavior throughout their lifecycle. This is atypical in most environments, as containers that need to be upgraded are typically destroyed and
reprovisioned with a new image. However, if you do need to customize rules, here are some best practices for doing so:

**Minimize the number of rules** — Creating static rules requires time and effort to build and maintain; only create rules where necessary and allow the autonomous models to provide most of the protection.

**Precisely target rules** — Be cautious of creating rules that apply to broad sets of images or containers. Providing wide ranging runtime exceptions can lower your overall security by making rules too permissive. Instead, target only the specific containers and images necessary.

**Name rules consistently** — Because rule names are used in audit events, choose consistent, descriptive names for any rules you create. This simplifies incident response and investigation. Also, consider using Prisma Cloud’s alert profile feature to alert specific teams to specific types of events that are detected.
Runtime defense for processes

Prisma Cloud provides runtime protection to ensure that processes that run in your environment meet the parameters of the learned models and the rules defined in Console. This article provides more information on the options available for tailoring the monitoring and response within the runtime rules.

Prisma Cloud ships with a rule named Default - alert on suspicious runtime behavior that enables runtime protection for processes by default. You can further refine your policy by creating additional custom rules that target specific resources, enable or disable protection features, and define exceptions to the automatically generated allow-list models.

Process rules

New runtime rules can be created in Console in Defend > Runtime > Container Policy. Click Add rule, specify a rule name and scope in the General tab, then select the Processes tab to define how Prisma Cloud handles new processes in running containers.

For more information on how rules are applied, and how models and rules are combined to create a resultant policy, see the article on runtime defense.

Effect

When behavior is detected that deviates from your runtime policy (resultant from the combination of your container model and your rules), Prisma Cloud Defender takes action. For processes, the Defender can be set into one of four modes.

- **Disable** — Defender doesn’t provide any protection for processes.
- **Alert** — Defender raises alerts when it detects process activity that deviates from your defined runtime policy. These alerts are visible in Monitor > Events > Container Audits.
- **Prevent** — Defender stops the process (and just the process) that violates your policy from executing. This is known as discrete blocking.
- **Block** — Defender stops the entire container if a process that violates your policy attempts to run.

Note that besides taking action on processes outside of the allow-list model, Defender also takes action when existing binaries that have been modified are executed. For example, an attacker might replace httpd (Apache) with an older version that can be exploited. Prisma Cloud raises alerts for each of the following cases:

- A modified binary is executed,
- A modified binary listens on a port,
- A modified binary makes an outbound connection.

Detection

Prisma Cloud can detect anomalous process activity. These features can be independently enabled or disabled.

**Detect crypto miners**

Prisma Cloud can detect crypto miners. If detected, a crypto miner incident type is created in Incident Explorer. When this option is enabled, Defender takes action on this type of incident according to the configured effect.
Detect processes used for lateral movement

Prisma Cloud can detect processes, such as netcat, known to facilitate lateral movement between resources on a network. If detected, a lateral movement incident type is created in Incident Explorer. When this option is enabled, Defender takes action on this type of incident according to the configured effect.

Detect parent child process relationships

As part of the model, Prisma Cloud learns what processes are invoked, and the parent processes that triggered the invocation. If this option is enabled, Defender can act on processes that are invoked by a parent other than that which is specified by the model. This action may show up as an audit in a number of different incident types in Incident Explorer.

Explicitly allowed and denied processes

The fields for Explicitly allowed processes and Explicitly denied processes let you tailor your runtime models. Processes can be listed by name or MD5 hash.
Runtime defense for networking

Prisma Cloud provides runtime protection to ensure that network connections made to and from your containers meet the policy defined by rules and modeling. Response to deviations from the resultant policy can be customized through runtime rules.

Prisma Cloud ships with a rule named **Default - alert on suspicious runtime behavior** that enables runtime protection for networking by default. You can further refine your policy by creating additional custom rules that target specific resources, enable or disable protection features, and define exceptions to the automatically generated allow-list models.

Networking rules

Prisma Cloud can monitor container networking activity for patterns that indicate an attack might be underway. These features can be independently enabled or disabled with runtime rules. The final policy that’s enforced is the sum of the container model and your runtime rules.

New runtime rules can be created in Console in **Defend > Runtime > Container Policy**. Click **Add rule**, specify a rule name and scope in the **General** tab, then select the **Networking** tab to define how Prisma Cloud handles container networking protection.

For more information on how rules are applied, and how models and rules are combined to create a resultant policy, see the article on runtime defense.

IP connectivity

When Prisma Cloud detects an outgoing connection that deviates from your runtime policy, Prisma Cloud Defender can take action. Networking rules let you put Defender into one of three modes:

- **Disable** — Defender does not provide any networking protection.
- **Alert** — Defender raises alerts when targeted resources establish connections that violate your runtime policy. The corresponding audits can be reviewed under **Monitor > Events > Container Audits**.
- **Block** — Defender stops the container if it establishes a connection that violates your runtime policy. The corresponding audit can be reviewed under **Monitor > Events > Container Audits**.

The fields for **Explicitly allowed** and **Explicitly denied** let you tailor the runtime models for known good and known bad network connections. These rules define the policy for listening ports, outbound internet ports for Internet destinations, and outbound IP addresses. Defining network policy through runtime rules lets you specify permitted and forbidden behavior for given resources, and instructs Defender on how to handle traffic that deviates from the resultant policy.

Detect port scanning

Port scans are used by attackers to find which ports on a network are open and listening. If enabled, Defenders detect network behavior indicative of port scanning. If detected, a **port scanning incident** is created in Incident Explorer.

Raw sockets

Prisma Cloud can monitor your environment for raw sockets, which can indicate suspicious activity. Raw sockets let programs manipulate packet headers and implement custom protocols to do things such as port scanning.

Raw socket detection is enabled by default.
DNS

Modern attacks, particularly coordinated, long running attacks, use short lived DNS names to route traffic from the victim's environment to command and control systems. This is common in large scale botnets. When DNS monitoring is enabled (Alert, Prevent, or Block) in your runtime rules, Prisma Cloud analyzes DNS lookups from your running containers. By default, DNS monitoring is disabled.

Dangerous domains are detected as follows:

- **Prisma Cloud Intelligence Stream** — Prisma Cloud’s threat feed contains a list of known bad domains.
- **Behavioral container models** — When learning a model for a container, Prisma Cloud records any DNS resolutions that a container makes. When the model is activated, Defender monitors network traffic for DNS resolutions that deviate from the learned DNS resolutions. Audits from DNS monitoring may be used to trigger a data exfiltration incident.

You can see the domains in the model by going to **Monitor > Runtime > Container Models**, clicking on a model, then opening the **Networking** tab. Known good domains are listed under **Behaviorally learned domains**.

- **Explicit allow and deny lists**: Runtime rules let you augment the Prisma Cloud’s Intelligence Stream data and models with your own explicit lists of known good and bad domains. Define these lists in your runtime rules.

In your runtime rules, set **Effect** in the DNS section to configure how Defender handles DNS lookups from containers:

- **Disable**: DNS monitoring is disabled. DNS lookups are not modeled in learning mode. DNS lookups aren't analyzed when models are active.
- **Alert**: DNS monitoring is enabled. DNS lookups are modeled in learning mode. DNS lookups are analyzed when models are active. Anomalous activity generates audits.
- **Prevent**: DNS monitoring is enabled. DNS lookups are modeled in learning mode. DNS lookups are analyzed when models are active. Anomalous activity generates audits. Anomalous DNS lookups are dropped.
- **Block**: DNS monitoring is enabled. DNS lookups are modeled in learning mode. DNS lookups are analyzed when models are active. Anomalous activity generates audits. When anomalous DNS lookups are detected, the entire container is stopped.
Runtime defense for file systems

Prisma Cloud's runtime defense for container file systems continuously monitors and protects containers from suspicious file system activities and malware.

**File system integrity**

Prisma Cloud monitors and protects against the following types of suspicious file system activity:

- Changes to any file in folders *not* in the runtime model.
- Changes to binaries or certificates anywhere in the container.
- Changes to SSH administrative account configuration files anywhere in the container.
- Presence of malware anywhere in the container.

**Malware protection**

Defender monitors container file systems for malicious certs and binaries using data from the Prisma Cloud Intelligence Stream. Console receives the Prisma Cloud feed, and then distributes it to all deployed Defenders. You can optionally supplement the Prisma Cloud feed with your own custom data.

When a file is written to the container file system, Defender compares the MD5 hash of the file to the MD5 hash of known malware. If there is a match, Defender takes the action specified in your rules. Defender also looks for attributes that make files suspicious, including signs they've been rigged for anti-analysis.

By default, Defender monitors both the container root file system and any data volumes. Container root file systems reside on the host file system. In this diagram, the running container also has a data volume. It mounts the db/ directory from the host file system into its own root file system. Both locations are monitored by Defender.

The following diagram shows how Prisma Cloud protects containers from malicious files:
File system integrity monitoring

Prisma Cloud ships with a default rule that monitors container file system integrity. You can see this rule under Defend > Runtime > Container Policy > Default - alert on suspicious runtime behavior. The default rule configures Prisma Cloud to continuously monitor and alert on suspicious file system activities in all running containers in your entire environment or cluster. When a rule is triggered, and the effect is alert, an audit is generated. You can view audits under Monitor > Events > Container Audits.

File system integrity defense

Create new runtime rules to augment the default rule. Runtime rules enable not only the detection, but also the prevention, of file system integrity violations so that your running containers can be actively defended.

Rule order and pattern matching are critical when designing policy.

The following procedure shows you how to create a custom rule to ensure file system integrity.

STEP 1 | Go to to Defend > Runtime > Container Policy.

STEP 2 | Click Add Rule.

1. Enter a name for your rule. Spaces are permitted.
2. Specify a scope.
   For this example, use an image name for one of your running containers and leave wildcards in all the other fields. For example, alpine:latest.
3. Leave Prisma Cloud Advanced Threat Protection enabled. For more information, see TATP.
4. Optionally select Kubernetes Attacks. For more information, see Kubernetes attacks.
5. Click the **File System** tab.

6. To ensure broad file system protection, set **File system monitoring** to **Enabled**.

7. Under **Denied & Fallback**, set the effect when this rule is triggered.
   
   - **Alert** – Generates an audit and raises an alert when an file system integrity violation is detected. Container continues to run.
   
   - **Prevent** – Prevents any attempt to violate file system integrity. Container continues to run. For more information, see discrete blocking.
   
   - **Block** – Stops the container when an attempt to violate the file system integrity is detected.

   *Audits are generated and alerts are sent (email, Slack, etc) when the effect is Alert, Prevent, or Block.*

8. Under **Denied & Fallback**, leave both **Changes to binaries and certificates** and **Changes to SSH and admin account configuration files** set to **Enabled**.

9. Rules can augment learned models by explicitly denying folders in the model or explicitly allowing folders that aren't in the model. The fields for denying or allowing folders are optional. Enter one or more paths. Do not use asterisks. By default, folders in the learned model are allowed, and all other folders are denied.
Runtime defense for hosts

Without secure hosts, you cannot have secure containers. Host machines are a critical component in the container environment, and they must be secured with the same care as containers. Prisma Cloud Defender collects data about your hosts for monitoring and analysis.

Runtime host protection is designed to continuously report an up-to-date context for your hosts. You can set alerts for filesystem, process, network, log file events, and more. Some events, such as process executions, can be blocked.

It’s not designed to enforce policy. The one exception is Defender’s process monitoring for hosts, which can alert or block undesired processes.

Enabling host runtime protection

Runtime protection for hosts is enabled by default. When Defender is installed, it automatically starts collecting data about the underlying host. Prisma Cloud ships with a default rule named Default - alert on suspicious runtime behavior, which enables some basic monitoring. To see the rule, open Console, then go to Defend > Runtime > Host Policy.

As part of the default rule, Prisma Cloud Advanced Threat Protection (TATP) is enabled. TATP supplements runtime protection by alerting you when:

- Malware is found anywhere on the host file system.
- Connections are made to banned IP addresses.
- Attempts to hijack execution flow is detected on the host.

Process monitoring is also enabled in the default rule, with both crypto miner detection and SSH session history enabled. To view the data collected, go to Monitor > Runtime > Host Observations, and select a host from the table.

Host runtime policy

Create new rules to enhance host protection. Go to Defend > Runtime > Host Policy, and click Add Rule.
Rules can be scoped by:

- Hostname.
- Labels.
- Cloud account.

For labels, Prisma Cloud supports AWS tags, as well as distro attributes. Distro attributes are designed for central security teams that manage the policies in Console, but have little influence over the operational practices of the groups that run apps in the environments being secured. If the central security team can’t rely on naming conventions or labels to apply policies that are OS-specific (e.g. different compliance checks for different OSs), they can leverage the distro attributes. Supported distro attributes are:

- Distro name — "osDistro:<value>" (e.g. "osDistro:Ubuntu")
- Distro version — "osVersion:<value>" (e.g. "osVersion:20.04")

**Process monitoring**

Process monitoring lets you alert or block specific processes by explicit policy. The Processes tab in the host runtime rule dialog has suggestions for processes known to aid exploits.

SSH history tracking can be disabled in this tab, where the hosts in scope are set in the General tab.

**Log inspection**

Prisma Cloud lets you collect and analyze operating systems and application logs for security events. For each inspection rule, specify the log file to parse and any number of inspection expressions. Inspection expressions support the RE2 regular expression syntax.

A number of predefined rules are provided for apps such as sshd, mongod, and nginx.
Networking

Prisma Cloud lets you secure host networking. You can filter DNS traffic and alert on inbound and outbound connections.

DNS

When DNS monitoring is enabled, Prisma Cloud filters DNS lookups. By default, DNS monitoring is disabled. Dangerous domains are detected as follows:

- Prisma Cloud Intelligence Stream -- Prisma Cloud's threat feed contains a list of known bad domains.
- Explicit allow and deny lists -- Host runtime rules let you augment the Prisma Cloud's Intelligence Stream data with your own lists of known good and bad domains.

When DNS monitoring is enabled, configure how Defender handles DNS lookups:

- **Alert** — Anomalous activity generates audits.
- **Prevent** — Anomalous activity generates audits. Anomalous DNS lookups are dropped.

IP connectivity

You can raise alerts when inbound or outbound connections are established. Specify inbound ports, and outbound IPs and ports.

Outbound connections are event-driven, which means that as soon as a process attempts to establish a connection, you'll be notified. Prisma Cloud polls inbound connections, which means you'll be notified periodically, and not necessarily the moment an inbound connection is established.

Activities

Set up rules to audit host events.

File integrity management (FIM)

Changes to critical files can reduce your overall security posture, and they can be the first indicator of an attack in progress. Prisma Cloud FIM continually watches the files and directories in your monitoring profile for changes. You can configure to FIM to detect:

- Reads or writes to sensitive files, such as certificates, secrets, and configuration files.
- Binaries written to the file system.
- Abnormally installed software. For example, files written to a file system by programs other than apt-get.

A monitoring profile consists of rules, where each rule specifies the path to monitor, the file operation, and exceptions.
The file operations supported are:

- **Writes** to files or directories. When you specify a directory, recursive monitoring is supported.
- **Reads.** When you specify a directory, recursive monitoring isn't supported.
- **Attribute changes.** The attributes watched are permissions, ownership, timestamps, and links. When you specify a directory, recursive monitoring isn't supported.

**Monitoring**

To view the data collected about each host, go to **Monitor > Runtime > Host Observations**, and select a host from the table.

**Apps**

The **Apps** tab lists the running programs on the host. New apps are added to the list only on a network event.

*Prisma Cloud automatically adds some important apps to the monitoring table even if they don't have any network activity, including cron and systemd.*
For each app, Prisma Cloud records the following details:

- Running processes (limited to 10).
- Outgoing ports (limited to 5).
- Listening ports (limited to 5).

Prisma Cloud keeps a sample of spawned processes and network activity for each monitored app, specifically:

- Spawned process — Processes spawned by the app, including observation timestamps, user name, process (and parent process) paths, and the executed command line (limited to 10 processes).
- Outgoing ports — Ports used by the app for outgoing network activity, including observation timestamps, the process that triggered the network activity, IP address, port, and country resolution for public IPs (limited to 5 ports).
- Listening ports — Ports used by the app for incoming network activity, including the listening process and observation timestamps (limited to 5 ports).

Proc events will add the proc only to existing apps in the profile. Defender will cache the runtime data, saving timestamps for each of the 10 processes last spawn time.

Limitations:

- Maximum of 100 apps.
- Last 10 spawned processes for each app.
SSH session history

The SSH events tab shows ssh commands run in interactive sessions, limited to 100 events per hour.

<table>
<thead>
<tr>
<th>User</th>
<th>IP</th>
<th>Process Path</th>
<th>Command</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubuntu</td>
<td>34.100.87.242</td>
<td>/bin/nc.openbsd</td>
<td>/bin/nc.openbsd</td>
<td>Jul 21, 2020 9:45:22 AM</td>
</tr>
<tr>
<td>ubuntu</td>
<td>34.100.87.242</td>
<td>/usr/bin/sudo</td>
<td>sudo nc -lp 555</td>
<td>Jul 21, 2020 9:45:22 AM</td>
</tr>
<tr>
<td>ubuntu</td>
<td>34.100.87.242</td>
<td>/bin/nc.openbsd</td>
<td>/bin/nc.openbsd</td>
<td>Jul 21, 2020 9:45:17 AM</td>
</tr>
<tr>
<td>ubuntu</td>
<td>34.100.87.242</td>
<td>/usr/bin/curl</td>
<td>curl <a href="http://www.google.com">www.google.com</a></td>
<td>Jul 21, 2020 9:44:10 AM</td>
</tr>
<tr>
<td>ubuntu</td>
<td>34.100.87.242</td>
<td>/usr/bin/dircolors</td>
<td>/usr/bin/dircolors</td>
<td>Jul 21, 2020 9:43:54 AM</td>
</tr>
<tr>
<td>ubuntu</td>
<td>34.100.87.242</td>
<td>/usr/bin/dirmime</td>
<td>dirmime /usr/bin/lesspipe</td>
<td>Jul 21, 2020 9:43:54 AM</td>
</tr>
<tr>
<td>ubuntu</td>
<td>34.100.87.242</td>
<td>/usr/bin/basename</td>
<td>basename /usr/bin/lesspipe</td>
<td>Jul 21, 2020 9:43:54 AM</td>
</tr>
<tr>
<td>ubuntu</td>
<td>34.100.87.242</td>
<td>/bin/bash</td>
<td>/bin/sh /usr/bin/lesspipe</td>
<td>Jul 21, 2020 9:43:54 AM</td>
</tr>
<tr>
<td>ubuntu</td>
<td>34.100.87.242</td>
<td>/bin/bash</td>
<td>/bin/bash</td>
<td>Jul 21, 2020 9:43:54 AM</td>
</tr>
</tbody>
</table>

Security updates

Prisma Cloud periodically checks for security updates. It’s implemented as a compliance check. This feature is supported only for Ubuntu/Debian distributions with the "apt-get" package installer.

Prisma Cloud probes for security updates every time the scanner runs (every 24 hours, by default). The check is enabled by default in Defend > Compliance > Hosts in the Default - alert on critical and high rule.

Compliance actions

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Severity</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>449</td>
<td>Linux host</td>
<td>high</td>
<td>Ignore</td>
<td>Ensure no pending OS security updates</td>
</tr>
</tbody>
</table>
The security updates tab shows pending security updates (based on a new compliance check that was added for this purpose). Supported for Ubuntu and Debian

On each host scan, Prisma Cloud checks for available package updates marked as security updates. If such updates are found, they’re listed under the security updates tab.

Audits

Audits can be viewed under Monitor > Events.
Custom runtime rules

Prisma Cloud's approach to scaling runtime defense in big, fluid environments is to model runtime behavior with machine learning. Machine learning reduces the effort required to manually create and maintain loads of rules to secure running software. When machine learning doesn't fully capture the range of acceptable runtime behaviors, rules provide a way to declaratively augment models with exceptions and additions.

Custom rules offer another, additional mechanism to protect running software. Custom rules are expressions that give you a precise way to describe and detect discrete runtime behaviors. Runtime sensors in your environment already detect process, file system, and network activity, then pass those events to Prisma Cloud for processing. Expressions let you examine various facets of an event in a programmatic way, then take action when they evaluate to true. Custom rules can be applied to both hosts and containers.

For example, the expression grammar supports the following logic:

"If user Jake runs binary netcat with parameter -l, log an alert"

Rule library

Custom rules are stored in a central library, where they can be reused. Besides your own rules, Prisma Cloud Labs also distributes rules via the Intelligence Stream. These rules are shipped in a disabled state by default. You can review, and optionally apply them at any time.

Custom rules are written and managed in Console under Defend > Runtime > Custom Rules. Click Add rule to bring up the online editor. The compiler checks for syntax errors when you save the rule.

There are four types of rules, but only three are relevant to runtime:

• processes
• filesystem
• networking-outgoing

Expression grammar

Expressions let you examine the contents of process, file system, and network events.

For example, any time a process is forked on a host protected by Container Defender or Host Defender, a process event fires. The following very simple expression looks for processes named netcat:

proc.name = "netcat"

Expressions have the following grammar:

expression: term (op term / in )*

• term --
  integer | string | keyword | event | '(' expression ')', unaryOp term
• in --
  '(' integer | string (',' integer | string)*?
• op --
  and | or | > | < | >= | # | = | !=
• unaryOp --
  not
• keyword --
startswith | contains

- **string** --
  Strings must be enclosed in double quotes
- **integer** --
  int
- **event** --
  process, file system, or network

**Process events**

Process events fire when new processes are forked. Expressions can examine the following attributes of a new process.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc.name</td>
<td>string</td>
<td>Process name.</td>
</tr>
<tr>
<td>proc.pname</td>
<td>string</td>
<td>Parent process name.</td>
</tr>
<tr>
<td>proc.path</td>
<td>string</td>
<td>Full path to the program.</td>
</tr>
<tr>
<td>proc.user</td>
<td>string</td>
<td>User to whom the process belongs.</td>
</tr>
<tr>
<td>proc.interactive</td>
<td>bool</td>
<td>Interactive process.</td>
</tr>
<tr>
<td>proc.cmdline</td>
<td>string</td>
<td>Command line.</td>
</tr>
<tr>
<td>proc.service</td>
<td>string</td>
<td>Only for host rules.</td>
</tr>
</tbody>
</table>

**Filesystem events**

Filesystem events fire when there are writes to disk. All properties of the process doing the writes are accessible from this context. Expressions can examine the following attributes of file system write activity.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.path</td>
<td>string</td>
<td>Path of the file being written.</td>
</tr>
<tr>
<td>file.dir</td>
<td>string</td>
<td>Directory of the file being written.</td>
</tr>
<tr>
<td>file.type</td>
<td>enum</td>
<td>File type. Supported types are: elf, secret, regular, and folder.</td>
</tr>
</tbody>
</table>

**Networking events**

Network events fire when a process tries to establish an outbound connection. Expressions can examine the following attributes when network events fire:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc.name</td>
<td>string</td>
<td>Name of process initiating the outbound network connection.</td>
</tr>
<tr>
<td>net.outgoing_port</td>
<td>string</td>
<td>Outbound port.</td>
</tr>
<tr>
<td>net.outgoing_ip</td>
<td>string</td>
<td>Outgoing IP address. The following expression looks for outbound connections to a range of IP addresses: net.outgoing_ip # &quot;1.1.1.1&quot; and net.outgoing_ip # &quot;1.1.1.9&quot;</td>
</tr>
<tr>
<td>net.private_subnet</td>
<td>bool</td>
<td>Private subnet.</td>
</tr>
</tbody>
</table>

**Example expressions**

The Prisma Cloud Labs rules in the rule library are the best place to find examples of non-trivial expressions.

**STEP 1** | In Console, go to **Defend > Runtime > Custom Rules**.

**STEP 2** | In the **Type** column, filter add a filter for processes, filesystem, or network outgoing.

**STEP 3** | Click on any rule that starts with **Prisma Cloud Labs** to see the implementation.

### Activating custom rules

Your runtime policy is defined in **Defend > Runtime > {Container Policy | Host Policy}**, and it’s made up of models and rules. Your expressions (aka custom rules) can be added to runtime rules, where you further specify what action to take when expressions evaluate to true. Depending on the event type, the following range of actions are supported: allow, alert, prevent, or block.

Custom rules are processed like all other rules in Prisma Cloud: the policy is evaluated from top to bottom until a matching rule is found. After the action specified in the matching rule is performed, rule processing for the event terminates.

> **Within a runtime rule, custom rules are processed first, and take precedence over all other settings. Be sure that there is no conflict between your custom rules and other settings in your runtime rule, such as allow and deny lists.**

**STEP 1** | Open Console, and go to **Defend > Runtime > {Container Policy | Host Policy}**.

**STEP 2** | Click **Add rule**.

**STEP 3** | Enter a name for the rule.

**STEP 4** | Click the **Custom Rules** tab.

**STEP 5** | Click **Select rules**, choose the rules to add, and click **Apply**.

**STEP 6** | Specify an effect for each expression.
Limitations

There are number of things that custom rules cannot do:

- The proc.cmdline and file.type fields are not supported in prevent mode. You'll get an error if you try to attach a custom rule to a runtime rule with these fields and the action set to prevent.

- Prisma Cloud cannot inspect command line arguments before a process starts to run. If you explicitly deny a process and set the effect to Prevent in the Process tab of a runtime rule, the process will never run, and Prisma Cloud cannot inspect its command line arguments. The same logic applies to custom rules that try to allow processes that are prevented by other policies. For example, consider process 'foo' that is an explicitly denied by a runtime rule, with the effect set to Prevent. You cannot allow 'foo -bar' in a custom runtime rule by analyzing proc.cmdline for for '-bar'.

- Prisma Cloud doesn't support prevent on write operations to existing files. For example, consider the following expression:

```
file.path = "/tmp/file"
```

If this expression is added to a runtime rule, and the effect is set to prevent, then Prisma Cloud will prevent the creation of such a file. If the file already exists, however, Prisma Cloud won't prevent any write operation to it, but will raise an alert.
Blocked containers

Prisma Cloud’s runtime defense system compares the state of a running container to the predictive model created for it during its learning period. When abnormal activity is detected, such as executing an unknown process, Prisma Cloud can:

- Raise an alert by generating an audit. Audits are shown under Monitor > Events > Container Audits. If you have an alert channel configured, such as email or Slack, audits are forwarded there too. Alert is the default action.
- Block the container by stopping it altogether. To enable blocking, create a new runtime rule.
- Prevent just the discrete process or file system write (not the entire container). To enable prevent, create a new runtime rule.

Blocking action

Blocking stops potentially compromised containers from running in your environment.

Prisma Cloud blocks containers under the following conditions:

- A container violates its runtime model, and you’ve installed a runtime rule with the action set to block. For example, if an attacker infiltrates a container and tries to run a port scan using nc, then the container would be blocked if nc weren’t a known, allowed process.
- A newly started container violates a vulnerability or compliance rule, and those rules have the action set to block. Prisma Cloud scans all images before they run, to enforce policies about what’s allowed to execute in your environment. For example, you policy might call for blocking any container with critical severity vulnerabilities.

Runtime rules can be created under Defend > Runtime > Container Policy. Vulnerability rules can be created under Defend > Vulnerabilities > Policy, and compliance rules can be created under Defend > Compliance > Policy.

Viewing blocked containers

Blocking immediately stops a container, taking it out of service. Blocked containers are never restarted. To see a list of blocked containers, go to the container audits page under Monitor > Events > Container Audits.
When a container is stopped, Prisma Cloud takes no further action to keep it stopped. Orchestrators, such as Kubernetes and Docker Swarm, start a fresh container in the blocked container's place. Orchestrators have their own mechanism for maintaining a set point, so they ignore the restart policy defined in the image's Dockerfile.

There is an exception when you run containers in a Docker-only environment (no orchestrator) and Prisma Cloud blocks a container. In this case, Prisma Cloud must take additional action to keep the container blocked. To prevent the container from automatically restarting, Prisma Cloud modifies the container's restart policy to always unless stopped. If you want to unblock a container, connect to the node with the blocked container, and manually modify the container's Docker configuration.

### Blocked container artifacts

Forensic investigators can inspect a blocked container's artifacts to determine why it was stopped. You can capture all of the container's contents, including its file system data, with the docker export command. Go to the node with the blocked container and run:

```
$docker export [container_id] > /path/filename.tar
```
Import and export individual rules

Prisma Cloud lets you import and export rules from one Console to another. Every rule created in Prisma Cloud under the **Defend** section has copy and export buttons in the **Actions** menu. An import button is located at the bottom of every rule table.

**Copying rules**

To copy a rule:

**STEP 1** | Go to **Defend > Runtime > [Vulnerabilities | Compliance | Access]**.

**STEP 2** | Click **Actions > Copy** for the rule you want to copy.

A dialog box named **Edit copy of...** opens.

**STEP 3** | Make any desired changes to the copied rule.

**STEP 4** | Click **Save**.

**Exporting rules**

Click **Actions > Export** next to any rule to export it in json format.

**Example**

```json
{
  "name": "Default - ignore Prisma Cloud components",
  "owner": "system",
  "effect": "alert",
  "resources": {
    "hosts": [
      "*
    ],
    "images": [
      "docker.io/twistlock/private:console*"
    ],
    "labels": [
      "*
    ],
    "containers": [
      "twistlock_console"
    ],
    "services": []
  }
}
```

**Importing rules**

A rule can be imported into Console in JSON format. To capture a rule in JSON format, use the export function described above.
Discrete blocking

Prisma Cloud lets you create runtime rules that block discrete processes inside a container. It is an alternative to stopping an entire container when the violation of a runtime rule is detected.

**Capabilities**

Prisma Cloud runtime rules let you deny specific processes. When you specify the **Prevent** action in a runtime rule, Prisma Cloud blocks containers from running processes that are not defined in the model or the explicitly allowed processes list. The rest of the container continues to execute without disruption. The alternative to discrete blocking is container blocking, which stops the entire container when a denied process is detected.

*The Prevent action is not supported on Debian 8.*

Prisma Cloud also lets you deny file system writes to specific directories. Like the process rule, file system rules can be configured with the **Prevent** action, which blocks the creation and modification of any files in the specified directories. This mechanism is designed to prevent bad actors from writing certificates or binary attack tools to disk, all without killing the process that initiated the write or stopping the entire container.

*The Prevent action for host file system rules requires a Linux kernel version 4.20 or later.*

*The Prevent action in file system rules is not supported when the Docker storage driver is set to aufs. It is supported for other storage drivers, such as devicemapper and overlay2. If you specify a Prevent action, but the storage driver does not support it, Prisma Cloud will respond with an alert and log the following message in Defender’s log: “Docker storage driver on host doesn’t support discrete file blocking”*

Creating discrete blocking rules

Discrete blocking rules are created under **Defend > Runtime**.

Runtime rules have both a **Process** and a **File System** tab. These tabs let you configure the following effects:

- **Disable** --
  - Deactivates the runtime sensor.
- **Alert** --
  - Logs an event for a violation. For process rules, the process continues to run. For file system rules, the file system write is permitted.
- **Prevent** --
  - For process rules, the process is killed. For file system rules, the file write is prevented.
- **Block** --
  - Stops the entire container when a violation is detected.

**STEP 1 | Open Console**
STEP 2 | Go to Defend > Runtime > Container Policy.

STEP 3 | Click Add rule.

STEP 4 | In the General tab, specify a rule name.

STEP 5 | Click on the Processes tab.

STEP 6 | Under Effect, select Prevent.

STEP 7 | In Explicitly denied processes, specify a list of process names or MD5 hashes for processes that should be blocked from running.

STEP 8 | Click Save.
Runtime defense for AWS Fargate

App Embedded Defenders monitor your Fargate tasks to ensure they execute as designed, protecting tasks from running suspicious processes or making suspicious network connections.

Policies let you define:

- Allow process activity. Enables verification of launched processes against policy.
- Allow networking activity. Enables verification of domain name resolution, and inbound and outbound network connections.

Besides runtime policy, you can also configure the WAAS application firewall to protect front-end Fargate containers.

Securing Fargate tasks

To secure a Fargate task, embed the Prisma Cloud Fargate Defender into it. The steps are:

1. Define your policy in Prisma Cloud Console.
2. Embed the Fargate Defender into your task definition.
3. Start the service.

Sample task

You can use the following sample task definition to test Prisma Cloud’s Fargate Defender. The associated container includes an `entry.sh` script that runs `mkdir` and then makes various outbound network requests to `yahoo.com` and `google.com` using `wget`. It then sleeps for 5 minutes and exits.

```json
{
  "requiresCompatibilities": ["FARGATE"],
  "containerDefinitions": [
    {
      "entryPoint": ["entry.sh"],
      "portMappings": [],
      "command": null,
      "image": "matthewabq/twistlock-fargate-auto",
      "name": "twistlock-fargate-task"
    }
  ],
  "family": "twistlock-fargate-task",
  "volumes": [],
  "networkMode": "awsvpc",
  "memory": "512",
  "cpu": "256"
}
```

Your task definition must including matching `entrypoint` and `cmd` parameters from the `Dockerfile(s)` of the `image(s)` in your task. Because Prisma Cloud does not see the actual images as part of the embedding flow, it depends on having these parameter present to reliably insert Defender into the task startup flow. If your `Dockerfile` does not include an `entrypoint` parameter, a default one, such as `/bin/sh`, must be used in the task definition. However, because the `cmd` parameter is optional, if your `Dockerfile` does not include a `cmd` parameter, one is not required in the task definition.
Defining your policy

Add runtime protection for your Fargate task by defining a runtime rule for it in Prisma Cloud Console.

By default, there are no rules in the Fargate runtime policy. Fargate Defenders dynamically retrieve policies from Console as they are updated. You can embed Fargate Defender into a task with empty or very simple initial policies, and refine them as needed later.

This procedure demonstrates how to block the sample task from executing a new process and establishing outbound network connections. You will create a new rule that prevents \texttt{mkdir} from running in the container named \texttt{twistlock-fargate-task}, and blocks outbound network requests to \texttt{yahoo.com}. If you’ve got your own task, configure the rule to meet your own specific objectives. By default, new rules apply to all images and containers \((\ast)\), but you can target them to specific images or containers using pattern matching.

**STEP 1 |** Log into Prisma Cloud Console.

**STEP 2 |** Go to Defend > Runtime > App Embedded Policy.

**STEP 3 |** Click Add rule.

1. Enter a rule name.
2. By default, the rule applies to all images and all containers.
   Target the rule to specific images or containers. A task definition declares the container name in the containerDefinitions→name field.
3. Click the Processes tab.
4. Set Effect to Prevent.
5. Click the Networking tab.
6. Set Effect to Prevent.
7. Click Save.

Embedding the Fargate Defender

Prisma Cloud cleanly separates the code developers produce from the Fargate containers we protect. Developers don’t need to change their code to accommodate Prisma Cloud. They don’t need to load any special libraries, add any files, or change any manifests. When a container is ready to be deployed to test or production, run your task definition through a transform tool to automatically embed the Fargate Defender, then load the new task definition into AWS.

The method for embedding the Fargate Defender was designed to seamlessly integrate into the CI/CD pipeline. You can call the Prisma Cloud API to embed the Fargate Defender into your task definition. Documentation for these API endpoints will be available shortly. See Jenkins Fargate example on how to integrate this process into a Jenkins Pipeline build.

**STEP 1 |** Log into Prisma Cloud Console.

**STEP 2 |** Go to Manage > Defenders > Deploy.

**STEP 3 |** In the first drop-down menu (1a), select the name or IP address Fargate Defender can use to connect to Console.

A list of IP addresses and hostnames are pre-populated in the drop-down list. If none of the items are valid, select the Names tab and add a new subject alternative name (SAN) using Add SAN button. After adding a SAN, your IP address or hostname will be available in the drop-down list in the "Deploy" tab.
Selecting an IP address in a evaluation setup is acceptable, but using a DNS name is more resilient. If you select Console’s IP address, and Console’s IP address changes, your Defenders will no longer be able to communicate with Console.

STEP 4 | In the second drop-down list (1b), select the Defender type of App Embedded.

STEP 5 | In Deployment type, select Fargate Task.

STEP 6 | Embed the Fargate Defender into your task definition.
   1. Copy and paste your task definition into the left-hand box.
   2. Click Generate Protected Task. A new task definition with the Fargate Defender is generated.
   3. Copy the new task definition from right-hand box.

Creating a task definition in AWS

Create a new task definition in AWS with the output from the previous section. If you already have an existing task definition, create a new revision.

This section is geared to creating a new task definition based on the sample task.

STEP 1 | Log into the AWS Management Console.

STEP 2 | Go to Services > ECS.

STEP 3 | Click Task Definitions, then click Create new Task Definition.
   1. Select Fargate, then click Next step.
   2. Scroll to the bottom of the page, and click Configure via JSON.
   3. Delete the prepopulated JSON, then paste the JSON generated for task from the previous section.
   4. Click Save.
   5. Click Create.
   6. Click View task definition.

Testing the task

Since the container associated with your task automatically executes mkdir and wget in the entrypoint script, simply launch your Fargate task, wait a few minutes, then review the audits in Prisma Cloud Console.

Prerequisite: You have already created an ECS cluster.

STEP 1 | Log into the AWS Management Console.

STEP 2 | Go to Services > ECS.

STEP 3 | Click Clusters, then select one of your clusters.

STEP 4 | Click the Services tab, then click Create.
   1. For Launch type, select Fargate.
   2. For Task Definition, select your task. If you’re using the sample task definition, select twistlock-fargate-task, and select the revision that contains the embedded Defender.
   3. For Service name, enter twistlock-fargate-task.
   4. For Number of tasks, enter 1.
   5. Click Next step.
   6. Select a Cluster VPC and Subnets, then click Next step.
7. For **Service Auto Scaling**, select **Do not adjust the service's desired count**, then click **Next step**.
8. Review your settings, then click **Create Service**.

**STEP 5 | Validate the results.**

1. Click **View Service**.

When **Last status** is **Running**, your Fargate task is running. The sample task automatically executes `mkdir` and `wget` (to google.com and yahoo.com), then exits after 5 minutes.
After a short time has passed, audits appear in Prisma Cloud Console. To review them, go to Monitor > Events > App Embedded Audits. You should see audits with the following messages:

<table>
<thead>
<tr>
<th>DNS resolution of suspicious name yahoo.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>/bin/mkdir launched from /bin/dash and is explicitly blocked in the runtime rule. Full command: mkdir test</td>
</tr>
</tbody>
</table>

**WAAS for Fargate**

All the capabilities of standard WAAS are available for Fargate tasks. The only difference is that Fargate Defenders run as a reverse proxies to all other containers in the task. As such, when you set up WAAS for Fargate, you must specify the exposed external port where Fargate Defender can listen, and the port (not exposed to the Internet) where your web application listens. WAAS for Fargate forwards the filtered traffic to your application port - unless an attack is detected and you chose Prevent in your WAAS for Fargate rule.

For more information on the type of attacks that Prisma Cloud detects and prevents, see Prisma Cloud WAAS.

To add an application firewall to a Fargate based web container:

**STEP 1 | Embed the Fargate Defender** into your web container’s Fargate task.

You can utilize the same sample Fargate task with one change: replace image matthewabq/twistlock-fargate-auto with httpd:2.4. The httpd:2.4 image is an Apache web container listening on default port 80.

**STEP 2 | Add a rule to protect your Fargate web container.**

1. Go to Defend > Firewalls > WAAS for Fargate and click Add rule.
2. Enter a rule name and select the desired protections, such as SQLiAttack protection.
3. Select Alert or Prevent.
4. Enter a port number for the External Port, then enter one for the web container Application Port. Typically this is 80 for HTTP and 443 for HTTPS). For this example, enter 8080 for the External Port and 80 for the Application Port.
5. Enter your Fargate task name Wildcards are allowed, but do NOT include the task version.
6. Click Save.
Create A New CNAF Rule

<table>
<thead>
<tr>
<th>General</th>
<th>HTTP Headers</th>
<th>File Uploads</th>
<th>Intelligence Gathering</th>
<th>Advanced</th>
</tr>
</thead>
</table>

Rule name: Firewall for Fargate based web container

Action: Disable, Alert, Prevent

- [X] Twistlock Advanced Threat Protection

- [ ] SQLi attack protection
- [X] CSRF protection
- [X] Attack tool protection
- [X] Malformed request protection
- [X] XSS attack protection
- [X] Clickjacking protection
- [X] Shellshock protection

Port Mapping

<table>
<thead>
<tr>
<th>External Port</th>
<th>Application Port</th>
<th>TLS</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>8080</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TLS: False

Add

Tasks

- [ ] matt-fargate

Add a task

All traffic to your Fargate web container will now be examined and protected by the embedded Fargate Defender.

STEP 3 | Test your Fargate WAAS protected task.
1. Run your protected web application Fargate task.

   Before launching your Prisma Cloud protected Fargate task, modify the security group’s inbound rules to permit TCP connections on the exposed port (8080) that you entered in the Fargate WAAS rule. This is the external port that allows you to access your web container. The security group’s inbound rules can be modified while the task is running. To disable WAAS protection, disable the WAAS rule, and re-expose the application’s real port by modifying the security group’s inbound rule.

2. Access your Fargate web container by browsing to the public IP address of your container. Specify the external port as defined in your WAAS rule and security group.

3. Test SQLi attack protection by running the following curl command:

   ```
curl -o reply.html -H 'Content-Type: application/json' -X POST \
   -d "\"-1+union+all+select+1,group_concat(user,0x3a,file_priv),3,4+from 
mysql.user--\"\" \nhttp://<public ip of fargate container>:8080
   ```

   The command should return a `reply.html` file that states the request was blocked by Prisma Cloud. There will also be an audit in Prisma Cloud Console at Monitor > Events > WAAS for App Embedded.

---

**Jenkins Fargate example**

Passing the Fargate task definition to your Prisma Cloud Console’s API returns the Prisma Cloud protected Fargate task definition. Use this task definition to start Prisma Cloud protected Fargate containers. This example demonstrates using the Jenkins Pipeline build process to:

- Call the Prisma Cloud Console’s API endpoint for Fargate task creation.
- Pass the Fargate task definition to the API.
- Capture the returned Prisma Cloud protected Fargate task definition.
- Save the Prisma Cloud protected Fargate task definition within the Pipeline’s archive https://<jenkins>/job/<pipeline_name>/<job#>/artifact/tw_fargate.json

In this example we have placed our simple task `fargate.json` and `Jenkinsfile` in a GitHub repository.
STEP 1 | Create an account in Prisma Cloud with the Defender Manager role.

STEP 2 | Create a Jenkins username/password credential for this account called twistlockDefenderManager.

STEP 3 | The $TL_CONSOLE Jenkins global variable was defined when the Prisma Cloud Jenkins plugin was installed.

STEP 4 | Create a Jenkins Pipeline
   1. Definition: Pipeline script from SCM
   2. SCM: Git
   3. Repository URL: <path to repository that contains both the Jenkinsfile and fargate.json>
4. Credentials: <credentials for repository>
5. Script path: Jenkinsfile
6. Save

**STEP 5 | Run Build Now**

**Stage View**

Average stage times:
(Average full run time: ~1s)

**STEP 6 |** The tw_fagate.json file will be within the archive of this build https://<jenkins>/job/<pipeline_name>/<job#>/artifact/tw_fargate.json
```json
{
    "containerDefinitions": [
        {
            "command": null,
            "entryPoint": [
                "/bin/tw/fargate/fargate_defender.sh",
                "fargate",
                "entrypoint",
                "entry.sh"
            ],
            "environment": [
                {
                    "name": "TW_IMAGE_NAME",
                    "value": "matthewabq/twistlock-fargate-auto"
                },
                {
                    "name": "TW_CONTAINER_NAME",
                    "value": "twistlock-fargate-task"
                }
            ],
            "image": "matthewabq/twistlock-fargate-auto",
            "mountPoints": [
                {
                    "containerPath": "/bin/tw/fargate/policy",
                    "readOnly": true,
                    "sourceVolume": "tw_policy"
                }
            ],
            "name": "twistlock-fargate-task",
            "portMappings": [],
            "volumesFrom": [
                {
                    "readOnly": false,
                    "sourceContainer": "TwistlockDefender"
                }
            ],
            "entryPoint": [
                "/usr/local/bin/defender",
                "fargate",
                "sidecar"
            ],
            "environment": [
                {
                    "name": "INSTALL_BUNDLE",
                    "value": "eyJvYmVyc2lvbl9fY2F0YWxvZ HickdQcRFl3OTJzc1Y"
                },
                {
                    ...
                }
            ]
        }
    ]
}
```
Incident Explorer

Incident Explorer elevates raw audit data to actionable security intelligence, enabling a more rapid and effective response to incidents. Rather than having to manually sift through reams of audit data, Incident Explorer automatically correlates individual events generated by the firewall and runtime sensors to identify unfolding attacks.

Audit events generated as a byproduct of an attack rarely occur in isolation. Attackers might modify a configuration file to open a backdoor, establish a new listener to shovel data out of the environment, run a port scan to map the environment, or download a rootkit to hijack a node. Each of these attacks is made up of a sequence of process, file system, and network events. Prisma Cloud’s runtime sensors generate an audit each time an anomalous event outside the allow-list security model is detected. Incident Explorer sews these discrete events together to show the progression of a potential attack.

To learn more about the challenges of incident response in cloud native environments, and how Prisma Cloud can help, see this webinar recording.

Viewing incidents

To view incidents, go to Monitor > Runtime > Incident Explorer. Click on an incident to examine the events in the kill chain. Clicking on individual events shows more information about what triggered the audit. After you have examined the incident, and have taken any necessary action, you can declutter your workspace by archiving the incident.
All the raw audit events that comprise the incident can be found in the audit data tab. To see the individual events and export the data to a CSV file, go to Monitor > Events > Container Audits.

Incident Explorer is organized to let you quickly access the data you need to investigate an incident. The following diagram shows the contextual data presented with each incident:
• (1) **Story** — Sequence of audits that triggered the incident.

• (2) **Image, container, and host reports** — Scan reports for each resource type. Scan reports list vulnerabilities, compliance issues, and so on.

• (3) **Connections** — Incident-specific radar that shows all connections to/from the container involved in the incident. Its purpose is to help you assess risk by showing you a connection graph for the compromised asset.

• (4) **Documentation** — Detailed steps for investigating and mitigating every incident type.

• (5) **Forensics** — Supplemental data collected and stored by Defender to paint a better picture of the events that led to an incident.

**Forensics**

Prisma Cloud Forensics is a lightweight distributed data recorder that runs alongside all the containers in your environment. Prisma Cloud continuously collects detailed runtime information to help incident response teams understand precisely what happened before, during, and after a breach.

Forensic data consists of additional supplemental runtime events that complement the data (audits) already captured by Prisma Cloud’s runtime sensors. It provides additional context when trying to root cause an incident. Each Defender collects and stores forensic data in a fixed-sized first-in-first-out log file on the host where it runs. Forensic data is only downloaded to Console when it’s needed for an investigation. This architecture enables Defender to store large amounts of data without any impact on network bandwidth or server processing (on the host where Console runs).

For containers, the following event types can be found in a forensic data set:

• Process spawned — Process was run in the container. Fields: timestamp, container ID, PID, PPID, path, command, arguments.
• Container started — Container was started. Fields: timestamp, container ID.
• Binary created — Executable file or binary blob was created (file system event). Fields: timestamp, container ID, user, PID, path.
• Listening port — Container is listening on a network port. Fields: timestamp, container ID, PID, path to executable that’s listening, listening start time, port.
• Connection established — Connection was established (incoming or outgoing) between the container and another entity. Fields: timestamp, container ID, source, destination, destination port.
• Runtime profile — Runtime action was allowed for the container image while it was in learning mode. Fields: timestamp, container ID, user, PID, PPID, path, command.
• Runtime audit — Event occurred in a container that violates your runtime policy (model + runtime rules). Fields: timestamp, container ID, user, audit message, attack type, effect (alert or block).

For hosts, Defender collects all raw process events:
• Process spawned — Process was run on the host. Fields: timestamp, hostname, path, PID, parent PID, parent path, user, command, interactive (true or false), program name.
• Binary created — Executable file or binary blob was created (file system event). Fields: timestamp, app, user, PID, path.
• Runtime profile — Runtime action was allowed for an app while it was in learning mode. Fields: timestamp, app, user, capabilities, command.
• Runtime audit — Event occurred in a container that violates your runtime policy (model + runtime rules). Fields: timestamp, app, user, audit message, attack type, effect (alert or block).

Forensics data is retrieved:
• After Prisma Cloud detects an incident. A minute after an incident occurs, Prisma Cloud collects forensic data from the relevant Defenders, and archives the data in Console. By default, Console stores up to 100 incident snapshots, which are managed on a FIFO basis.
• On-demand. Forensics data can be retrieved for review at any time from the Console UI.

Configuring data collection

To configure Forensics, go to Manage > System > Forensics. By default, forensic data collection is enabled. With forensic data collection enabled, Defender requires an additional 1 MB of memory and 110 MB of storage space. If enabled, you can specify the amount of storage space allocated to each Defender. At minimum, Defender requires 100 MB for container forensics and 10 MB for host forensics. You can specify a maximum of 1000 MB for each category.

Several settings dictate what type of data is collected and for how long:

• Max number of incident snapshots Console can store — After an incident occurs, Prisma Cloud collects and saves the relevant forensic data set in Console. To control the amount of data Console stores, Prisma Cloud caps the number of data sets and manages them on a FIFO basis.
• Collect network snapshots — When this option is enabled, the forensic package that you can download from Console includes a netstat-style snapshot of the current connections.
• Collect network firewall snapshots — When this option is enabled, the forensic data includes the Connection established event type, which shows incoming and outgoing connection details, including source IP, destination IP, and destination port.

Viewing forensic data

Forensic data is associated with incidents.

Note that if you configure to Prisma Cloud to send out alerts on channels, such as email or Slack, when incidents occur, the alert messages will contain a direct link for downloading the forensics data.

STEP 1 | Open Console, and go to Monitor > Runtime > Incident Explorer.
STEP 2 | In the **Active** tab, select an incident.

STEP 3 | Click on **View forensic data**.

Viewing container forensic data

While Incident Explorer presents forensic data relevant to specific incidents, you can also view all available forensic data at anytime outside the scope of an incident.

For containers, forensic data is collected on a per-model basis. To retrieve and review the forensic data for a container:

**STEP 1 |** Open Console, and go to **Monitor > Runtime > Container Models**.

**STEP 2 |** In the table, click the microscope icon for the container of interest.

<table>
<thead>
<tr>
<th>Label</th>
<th>OS</th>
<th>Entrypoint</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>twister:2020-02.256</td>
<td>Alpine Linux v3.9</td>
<td>/usr/local/bin/defender</td>
<td>Active</td>
</tr>
<tr>
<td>twister:2019-02.256</td>
<td>Alpine Linux v3.9</td>
<td>/app/server</td>
<td>Active</td>
</tr>
<tr>
<td>cloud-discovery:latest</td>
<td>Alpine Linux v3.8</td>
<td>/app</td>
<td>Active</td>
</tr>
<tr>
<td>Prometheus:latest</td>
<td>BusyBox 1.29.3</td>
<td>/bin/prometheus --config.file=/etc/prometheus/prometheus.yml</td>
<td>Active</td>
</tr>
<tr>
<td>test</td>
<td>Ubuntu 18.04.1 LTS</td>
<td>sh</td>
<td>Active</td>
</tr>
</tbody>
</table>

Events are displayed in a coordinated timeline-table interface.
Viewing host forensic data

To retrieve and view the forensic data for a host:

**STEP 1** | Open Console, and go to Monitor > Runtime > Host Models.

**STEP 2** | Click the Host toggle button.

**STEP 3** | In the table, click the microscope button for the host of interest.
Incident types

This section describes the incident types surfaced in Incident Explorer.

Backdoor admin accounts

Backdoors are a method for bypassing normal authentication systems, and are used to secure remote access to a system.

Backdoor admin account incidents surface event patterns that indicate an actor might have created or modified a configuration to enable the continued use of a privileged account.

Investigation

In the following incident, you can see that a shell was used to modify /etc/passwd, potentially enabling an attacker to add or change a user account. In addition, there was other suspicious process activity that made a copy of /etc/passwd.

The first step in an investigation is to validate that the changes represent a bona fide security incident. In this case, the events that led to the incident seem to indicate a valid security incident, but you should examine the changes to /etc/passwd to see if they represent the potential for an attacker to maintain persistence.

Having determined that this is a bona fide incident, then the next steps focus on determining how an attacker was able to modify the system configuration. This would, generally, be a post-compromise approach to maintain access to the compromised systems. Check Incident Explorer for additional incidents, such as hijacked processes. Review additional runtime audits for the source to see if there are other clues. Review access to the container and ensure that the affected account(s) weren’t subsequently used for further access to systems and data.

Mitigation

A full mitigation strategy for this incident begins with resolving the issues that allowed the attacker to modify the system configuration.

Ensure that compliance benchmarks are appropriately applied to the affected resources. For example, if the critical file systems in the container are mounted read-only, it will be more difficult for an attacker to change a configuration to their advantage.
Backdoor SSH access

Backdoors give attackers a way to bypass normal authentication systems, and are used to secure remote access to a system.

Backdoor SSH access incidents indicate that an attacker might have changed the configuration of a resource to enable remote access to the resource.

Investigation

In the following incident, you can see two audits. The first audit is a file system event that shows a new certificate was created in `/etc/ssl/certs`. An attacker could use this certificate for follow-on access to the container.

The first step in an investigation is to validate that the changes represent a bona fide security incident. In this example, it’s unlikely that a new cert named `bad_guy.pem.pub` is a valid change, but it might not always be so clear.

After validating that this is a security incident, the next step is determining how an attacker was able to modify the system configuration. This would, generally, be a post-compromise approach to maintain access to the compromised systems. Check Incident Explorer for other potentially related incidents, such as hijacked processes. Review additional runtime audits for the source to see if there are other clues.

Review access to the container and ensure that accesses weren't subsequently used for further access to systems and data.

Mitigation

A full mitigation strategy for this incident begins with resolving the issues that allowed the attacker to modify the system configuration.

Ensure that compliance benchmarks are appropriately applied to the affected resources. For example, if the critical file systems in the container are mounted read-only, it will be more difficult for an attacker to change a configuration to their advantage.

Brute force

A Brute Force incident surfaces a combination of audit events that indicate a protected resource is potentially being affected by an attempted DoS.

Investigation

In the following incident, you can see that a container received a flood of attempted actions to the extent that the Web Application and API Security (WAAS) blocked the source.
Review the WAAS audit logs to determine any further impact:

Additionally, review the logs of potentially affected applications to determine if there was any further impact.

**Mitigation**

Ensure that WAAS rules provide protection for exposed services.

**Crypto miners**

Crypto miners are software used to generate new coins in cryptocurrencies such as Bitcoin and Monero. These can be used legitimately by individuals; however, in containerized environments, they are often executed by attackers as a means of monetizing compromised hosts.

Unless you are intentionally running a crypto miner, this alert most likely indicates a security incident in which an attacker was able to introduce a crypto miner into your infrastructure and execute it.

Our research indicates that the potential attack vectors include:

- A Kubernetes or Docker endpoint exposed to the Internet that allows unauthenticated access, or that is protected with weak credentials.
- A registry exposed to the Internet that allows unauthenticated users, or users with weak or common passwords, to make changes to stored images.
- Vulnerable code in a containerized service that has been exploited, followed by lateral movement and remote code execution.
Investigation

The first step in determining how the crypto miner was introduced is to determine if this is an existing image which has had unwanted processes introduced into it or if this is an entirely unwanted image. You can inspect the image itself in the Prisma Cloud Console.

We can see that this image comes from Docker Hub and that it is not an image that was developed internally. In this case, we would want to dig deeper into how the image was pulled and the container executed. You may have many sources of this information including the Prisma Cloud Docker access logs (Monitor/Access/Docker), which have been exported to CSV and filtered here:

This shows that a user account, ‘alice’, was used to run ‘docker exec’ and start the container, and that the command was run locally. From here, we would want to review authentication logs on the system to determine how ‘alice’ was able to logon and to review other data to determine what else ‘alice’ was able to accomplish.

If the image was an existing one that the enterprise legitimately uses, the next steps in the investigation would be to determine how the image was modified to include the crypto miner. Start by reviewing the image in any registry where it is stored and looking at a history of changes made to the image. It may be necessary to walk through the entire CI/CD pipeline to determine if changes were made prior to being pushed to the registry.

Mitigation

As soon as the investigation is complete, remove all instances of the running container (docker stop quirky_payne | docker rm quirky_payne in this case). If the container(s) were started with an orchestrator like Kubernetes, it may be necessary to remove any configuration that would cause them to restart.

If the image was pushed to a registry, take steps to remove affected versions from the registry.

Secure all access, starting with any point of entry that was found. Ensure that only needed endpoints are exposed to the Internet and that authentication is required at each endpoint that could, directly or indirectly, result in remote code execution. Ensure accounts have strong passwords and, where possible, two-factor authentication.

Investigate any successful attack vectors that were found in the investigation. This may not be the only successful attack to have used this approach; instead, it may just be the most visible one.

Data exfiltration

Data exfiltration is the unauthorized transfer of data from one system to another.

Data exfiltration incidents are triggered when a pattern of audits indicate attempts to move data to an external location.
Investigation

The following screenshot shows a data exfiltration incident. Two audits, taken together, indicate that:

- This container used DNS to resolve ix.io, a website similar to pastebin.com, that lets users upload text.
- An outbound connection was made to an IP address.

In order for Prisma Cloud to learn and model the names a container resolves during the course of normal operation and to detect the resolution of known bad domains, you must enable DNS monitoring in your runtime network rules. DNS monitoring is not currently supported on Docker Swarm.

As a first step, it's important to validate whether these factors (IP and DNS name, in this case) can be explained by expected behavior. Using a search on virustotal.com, or similar service, you can determine that the IP address is associated with ix.io. Since it is highly unusual for a production service to use a service such as this, it is safe to assume that this is a bona fide security incident.

The next step is to review other audits in this timeframe:

A review of the audit log shows that a system file, `/etc/group`, was copied and posted to ix.io using curl. It is likely that this is part of a larger incident.

The next step is to determine how the actor was able to access and exfiltrate data. There may be additional incidents generated that shed light on this question. For example, hijacked process and lateral movement incidents from this source or related sources might have transpired just before this incident.

If there are no clues in other logged incidents, some other potential avenues of investigation include:

- Review Docker access logs in Prisma Cloud (if enabled) to determine if execution was initiated from the host.
- Review Docker logs for the container.
- Review available application logs.
Mitigation

The first step in mitigating the exfiltration of data is to determine the content and sensitivity of data potentially affected.

- If the data represents critical system information, such as stored secrets, take the appropriate action to protect potentially exposed systems.
- If the data represents sensitive data such as PII, take the appropriate steps as specified by policy and recommended by legal counsel.

In the longer-term, implementing runtime rules to prevent or block anomalous behavior can help to contain future attempts at data exfiltration.

Hijacked processes

A hijacked process incident indicates that an existing process has been used in ways that are inconsistent with its expected behavior. This type of incident could be a sign that a process has been used to compromise a container.

Investigation

The following incident shows that `java`, which is an expected process in this Struts2 container, has launched a bash shell. This is decidedly unexpected behavior. You can also see that it wrote out a suspicious new file named `.java`.

The first step in an investigation is to determine if this is indeed malicious behavior. Reviewing the audit logs under Monitor > Events > Container Audits shows a pattern of behavior that is troubling. A number of commands are being executed by Java, including a copy of the sensitive `/etc/passwd` file.

The next step in the investigation is to determine how an attacker was able to hijack the process. A likely culprit is a vulnerability in the code deployed to the container. Reviewing the vulnerability scan report for
the underlying image shows that it contains a package with a remote code execution vulnerability. This vulnerability is remotely exploitable and exploit code is readily available.

Reviewing the application logs for this container, with `docker logs <CONTAINER-NAME>`, shows errors consistent with the exploitation of CVE-2017-5638.

Mitigation

The first step in mitigating the issue is fixing the root cause and redeploying the image. Prisma Cloud’s layer view of vulnerabilities shows developers where the vulnerability was introduced. For this image, you can see that CVE-2017-5638 is present in a .war file downloaded at the end of the Dockerfile.
Additionally, enabling prevention of runtime process events would provide future defense-in-depth.

**Kubernetes attacks**

Exploiting weaknesses in the container orchestrator to manipulate cluster settings is known as a Kubernetes attack. This incident indicates attempts to directly access Kubernetes infrastructure from within a running container. This may be an attempt to compromise the orchestrator.

Actions that can trigger this incident include attempts to download and use Kubernetes administrative tools within a container, in addition to any attempts to access Kubernetes metadata.

To detect Kubernetes attacks, you must have a runtime rule with the **Detect Kubernetes attacks** option enabled.

**Investigation**

The first step in an investigation is to validate that the changes represent a bona fide security incident. Having determined that this is a bona fide incident, then the next steps focus on determining how an attacker would have gained access to the resources with access to the Kubernetes cluster. Also, it is important to restrict access to your cluster by following best practices regarding access control.

Review your Kubernetes cluster to ensure that no actions were taken to compromise your cluster. In addition, closely review the audit actions and the forensic data available through incident explorer to understand the scope of the incident.

**Mitigation**

A full mitigation strategy for this incident begins with resolving the issues that allowed the attacker to attempt to access the Kubernetes infrastructure.

For additional protection, customize your runtime rules to **prevent** or **block** actions that access the metadata services or the open local kubelet port. Compliance rules should include checks set to **alert** or **block** to ensure your containers and hosts are following the best practices for Kubernetes.

**Lateral movement**

Lateral movement incidents indicate that an attacker is using tools and techniques that enable movement between resources on a network.
Investigation

The following incident shows that netcat was used to establish a listener on port 9000.

This behavior is a probable precursor to creating a reverse shell, allowing network-based remote control of another resource.

Your investigation should focus on:

- Determining how the process in the alert, such as `nc.openbsd`, was executed. Review additional entries in Incident Explorer and other audits from the source, looking for unusual process execution, hijacked processes, and explicit execution of commands.
- Reviewing container runtime audits to determine if the target successfully connected.
- If the target did successfully connect, determine what the attacker was able to do and if they were able to move further through the network.

Mitigation

After determining the cause of the process execution, resolve the problem, whether it be an exposed vulnerability, a configuration issue, or something else.

For additional protection, enable the `prevent` or `block` actions in the applicable runtime rules to take action when anomalous processes, such as `netcat`, are executed.

Port scanning

Port scans are a method for finding which ports on a network are open and listening. It is a reconnaissance technique that gives attackers a map of where they can further probe for weaknesses.

Port scanning incidents indicate that a container is attempting to make an unusual number of outbound network connections to hosts and ports to which it does not normally connect. Port scanning could be a post-compromise attempt to use the container to find other resources on the network as a precursor to lateral movement.

Investigation

The following screenshot shows a port scanning incident.
The first step in an investigation is to determine whether the source of the outbound network activity was an otherwise-valid process that was misused or a newly introduced process. Prisma Cloud audit logs are a great place to start. The runtime container audits, under **Monitor > Events > Container Audits**, show that a shell was launched immediately before the port scan, and that the shell was used to launch nmap. Nmap is a popular network scanning tool.

The next step in the investigation is to determine how nmap was introduced and executed. Some plausible scenarios include:

- A user account was used to execute nmap via a Docker command from the host. If enabled, Prisma Cloud access logs would show which user ran the command and when it was run.
- A remote code execution vulnerability was used to run nmap remotely. If the Prisma Cloud Web Application and API Security (WAAS) was configured to protect this container’s inbound traffic, the WAAS logs may help with your investigation. Additionally, logs from the services in the container, such as Apache access logs, may shed additional light on the incident.

Once the cause has been identified, the next step in the investigation is to review the services that the actor may have discovered via port scanning and to inspect those containers to ensure that there hasn’t been additional lateral movement. Container runtime audits may show specific connection attempts.

**Mitigation**

Mitigation and remediation for a port scanning incident should focus on resolving the issue that allowed execution of the responsible process.

**Service violation**

Service violation incidents indicate that a service running on a protected host has attempted to use privileges beyond what is expected.

**Investigation**

Determine if the service has any known vulnerabilities by reviewing the applicable information in **Monitor > Vulnerabilities > Hosts**.

For additional information, review the Prisma Cloud runtime audit logs, any logs that the service generates, and syslog on the affected host.

**Mitigation**

Resolve any vulnerability and access issues found in the investigation phase.
Access control

Establish and monitor access control measures for cloud workloads and cloud native applications.

- Role-based access control for Docker Engine
- Admission control with Open Policy Agent
Role-based access control for Docker Engine

Prisma Cloud lets you control access to Docker commands based on group membership. Prisma Cloud lets you:

- Secure access to remote Docker Engine instances.
- Control access to Docker commands on a user-by-user basis.

After integrating Prisma Cloud with Active Directory, OpenLDAP, or SAML, you could create a group called Dev Team. Then in Console, you could grant all users in Dev Team permission to remotely run any Docker commands on hosts in the development environment, but deny permission to create, start, or stop containers on hosts in the production environment.

Securing remote access

The following diagram shows how Docker commands are routed from a user's workstation over the network to a host protected by Defender:

![Diagram showing Docker client, Defender, Docker daemon, and Container, with TLS and UNIX socket connections]

The Docker client securely transmits the command over the network to Defender using the Transport Layer Security (TLS) protocol. Defender acts as a proxy to the Docker daemon. If the installed policy permits the command to be executed, it is forwarded to the Docker daemon over the UNIX socket. The UNIX socket is created when the Docker daemon first starts, and it exposes a REST API through which Docker commands can be run.

Controlling access to resources

The following sequence diagram shows how users gain access to Docker resources, and how your access policies are enforced.

In this flow, it is assumed that:

- User Bruce has been added to the AD group Prisma Cloud Devs.
- You have already configured your access policy rules in the Prisma Cloud Console.
1. Bruce logs into Console with his LDAP credentials. He's directed to the single page user view.
2. From the single page user view, he copies a command that installs certs on his machine. These certs identify him as Bruce. Group memberships for the user are embedded in the certificate.
3. Bruce runs the install command on his machine. It copies the certs into the $HOME/.docker directory. He can now use TLS to communicate securely with hosts that run Defender.
4. Bruce runs a Docker command on DevHostA (protected by Defender) from his local machine. He specifies the hostname for DevHostA and the port number where Defender listens. By default, Defender listens for TLS traffic on port 9998.
5. Defender acts as a gateway to the Docker daemon. It uses the certificate to determine the user's identity and group memberships. Defender allows or blocks the command, depending on the access policies specified in Console.
6. In this case, Bruce has the right permissions to run this docker command. The command is forwarded to the docker daemon.
7. The response from the Docker daemon is routed back to Bruce through Defender.
Note that Defender does not talk to the identity provider (IdP). Instead, it relies on the user certificate generated from the initial authentication flow, when the user first tries to log into Console. The validity period for the certificate is controlled by the IdP, which embeds the login expiration into its response.

### Setting Defender's listener type

To enforce role-based access control, Defender's listener type must be set to TCP.

Clients connect to the Docker socket and use the Engine API to manage and control containers on a host. The best known client is the docker command line tool (docker run, docker ps, etc).

In TCP mode, Defender intercepts traffic to the Docker socket and assesses it against the policies you have installed in Console. With this setup, Defender can block Docker commands and prevent them from reaching the Docker socket for execution by the Docker daemon.

In TCP mode, Defender listens for Docker traffic on port 9998 (this value can be configured). Defender runs as a Docker client with non-exclusive access to the Docker socket. Anyone who gains direct access to the Docker daemon will be able to bypass Defender and your policies. To prevent attackers from circumventing Defender, you should lock down your hosts and harden them for least privilege access.

Docker commands should only be run from remote machines through Defender on port 9998. Any user running Docker commands on port 9998 must be authenticated and authorized. Console generates certificates for users to authenticate to Defender. Any command run against Defender must also be explicitly allowed. Prisma Cloud ships with a default deny-all rule that blocks all commands for all users.

You can dynamically change Defender's listening type from Console, even after Defender is installed.

**STEP 1** | Open Console, and go to Manage > Defenders > Manage.

**STEP 2** | Click on a Defender listed in the table to open a dialog with more details.

**STEP 3** | In the **Choose the socket type** drop-down list, select **TCP**.

**STEP 4** | Click **Save**. The socket type for the Defender is updated in the Defenders status table.

### Authentication and identity

Prisma Cloud can authenticate users against its internal local database. The initial admin user created when you first access Console, for example, is a local user. Prisma Cloud can also authenticate users against external services, such as Active Directory or SAML Identity Providers.

Users are identified with client certificates. These certs are automatically generated by Prisma Cloud for each user. Users log into Console with their credentials, then download a script that installs the certs on their machine. Client certs should be installed on any host where the `docker` client can be run.

To install the initial client certs on your host:

**STEP 1** | Open Console.

**STEP 2** | Log in with your credentials.
STEP 3 | Go to Manage > Authentication > User Certificates.

Users with the Access User role are directed to this page by default.

STEP 4 | Install your client certs, which are used to authenticate commands sent from the Docker client through Prisma Cloud.

Copy the curl-bash command under Client certificate installation, then run it on your host. Your client certificate, client private key, and the certificate authority certificate are installed in $HOME/.docker/.

If you’re using custom certificates for authentication, then the above commands only install the certificate authority in the default Docker folder. The other two user certificates must be manually copied to this location.

Configuring Docker client variables

For access control to work, all Docker commands must be routed through Defender. You can configure your environment to shorten the Docker commands that target remote hosts protected by Defender. You should have already installed your client certificates.

To access Docker daemon through Defender, explicitly specify the host and the port of the Defender. For example:

```
$ docker -H <defender_host_address>:9998 run alpine
```

To simplify and shorten the Docker command, set up the following environment variables to route management traffic to Defender by default.

```
$ export DOCKER_HOST=tcp://<defender_host_address>:9998
$ export DOCKER_TLS_VERIFY=1
```

These environment variables can be set on a local machine (such as a dev laptop) that accesses Docker daemon on some remote host (such as a corporate cloud), or they can set directly on the host that runs Defender, for users who do not have root privileges (which should be the majority of the users on such a host).

Creating access control rules

Admins can create policies that control which users can run what commands on what hosts.

For example, an admin could create an access control rule called that limits members of the "Dev team" group to a handful of read-only operations so they can debug issues in the production environment. The admin might decide that docker ps, docker logs, and docker inspect are sufficient for devs to do their job, and he could limit the scope of the rule to hosts named prod*. When this rule is activated, users that are part of the Dev Team group can only run these Docker client commands on production hosts. All other commands are blocked.

Modify the parameters in this example to meet your own specific requirements.

Prerequisites:

- For the purposes of example scenario, you have integrated Prisma Cloud with Active Directory. You could also integrate with OpenLDAP or SAML, or have Prisma Cloud manage your users and groups.
- You have created AD groups for the different types of users that need access to Docker services. This procedure assumes you have a group called Prisma Cloud Devs, and that it has at least one user.

STEP 1 | Set up a user access rule.
1. Log into Console as an admin user.
2. Go to Defend > Access > Docker.
3. Click Add rule.
4. Enter a name for your rule.
5. Set Effect to Allow.
6. Deselect All, then select the Actions to allow:
   - container_list to allow access to the docker ps command.
   - container_logs to allow access to the docker logs command.
   - container_inspect to allow access to the docker inspect command.
7. In the Groups field, delete the wildcard (*) and enter the group(s) for which this rule applies.
   For example, enter Dev team.
8. Click Save.
9. Verify that your new rule is at the top of the list.

Console ships with a default rule that blocks all Docker commands from remote clients.
Rules are enforced according to the order that they are listed in Console. Rules at the top of the list have a higher priority than rules lower down.

STEP 2 | Verify that your policy is being enforced.
1. If you're logged in to Console as an admin user, log out.
2. Log into Console as a user from your group.
3. On the Manage > Authentication > Credentials page, copy the install command for the client certificate.
4. On your local machine, paste the install command into a shell window and run it.
5. Run a Docker command that's not allowed.

```bash
$ docker -H <HOST>:9998 --tlsverify pull nginx
Error response from daemon: [Prisma Cloud] The command 'image_create' denied for user 'bruce@example.com' by rule 'devs_rule'
```

Troubleshooting

**You cannot run Docker commands**
First remove Prisma Cloud from the equation. Verify that you can communicate with Docker locally without Defender in the middle. After you have verified this setup, review the parameters you pass to the docker client.

**Your policies are not being properly enforced.**
Verify your user is in the AD group by following the below steps on the Docker host(s) where you're trying to execute a command:
1. Install ldap-utils:

```bash
$ sudo apt-get install ldap-utils
```
2. Query Active Directory to verify that your user belongs to your AD group. Use the same parameters that you specified in your integration configuration.

```bash
$ ldapsearch \
-x -H [LDAP_URL] \
-D [LDAP_ADMIN_UPN] \
-W \
```
`-b [LDAP_SEARCH_BASE]\n-s sub (&(userPrincipalName=[UPN])(memberof=[LDAP_GROUP_DN]))`

Where:

- **UPN** --
  
  User Principal Name of the user

- **LDAP_GROUP_DN** --

  Full DN of the LDAP group. For example: `CN=group1,DC=USERS,DC=TWISTLOCK,DC=LOCAL`
Admission control with Open Policy Agent

Prisma Cloud provides a dynamic admission controller for Kubernetes that's built on the Open Policy Agent (OPA). In Console, you can manage and compose rules in Rego, which is OPA's native query language. Rules can allow or deny (alert or block) pods. Console pushes your policies to Defender, which enforces them. Decisions made by the system are logged.

There is currently no support for OpenShift or Windows.

Open Policy Agent

The Open Policy Agent is an open source, general-purpose policy engine that lets you consolidate policy enforcement in a single place. OPA can enforce policies in microservices, Kubernetes clusters, CI/CD pipelines, API gateways, and so on. OPA provides a high-level declarative language called Rego, which lets you specify policy as code. The OPA APIs let you offload policy decision-making from your software.

OPA decouples policy decision-making from policy enforcement. When your software needs to make policy decisions, it queries OPA and supplies structured data, such as JSON, as input. The data can be inspected and transformed using OPA's native query language Rego. OPA generates policy decisions by evaluating the query input and against policies and data.

Prisma Cloud operationalizes OPA by:
- Extending Console to manage and compose policies in Rego.
- Integrating OPA’s decision-making library into Defender.
- Connecting Defender’s enforcement capabilities to OPA’s decisions.

Admission webhook

An admission controller is code that intercepts requests to the API server for creating objects. There are two types of admission controllers: built-in and dynamic. Prisma Cloud implements a dynamic admission controller.

Dynamic admission controllers are built as webhooks. After registering to intercept admission requests, they assess requests against policy, and then accept or reject those requests. In Kubernetes terms, these are known as validating admission webhooks.

The Prisma Cloud validating admission webhook handles the API server’s AdmissionReview requests, and returns decisions in an AdmissionReview object. When configuring Prisma Cloud, you'll create a ValidatingWebookConfiguration object, which sets up the Defender service to intercept all create, update, and connect calls to the API server.

The default ValidatingWebookConfiguration provided here sets failurePolicy to Ignore. The failure policy specifies how your cluster handles unrecognized errors and timeout errors from the admission webhook. When set to Ignore, the API request is allowed to continue.

Configuring the webhook

Configure the API server to route AdmissionReview requests to Prisma Cloud.

The ValidatingWebhookConfiguration provided in Console is for API version v1beta. For Kubernetes 1.16 and later, the API version is v1. The v1 object is different than the v1beta object. For the v1 template, see the template section.
Prerequisites:

- You have a running instance of Prisma Cloud Compute Console.
- You have a Kubernetes cluster. Minimum supported version is v1.13.
- Defender has been deployed to your cluster as a DaemonSet. In Console, you can verify Defenders are running and connected under Manage > Defenders > Manage.

STEP 1 | Go to Manage > Defenders > Manage > Defenders.

STEP 2 | Click on Advanced Settings.

1. Enable admission control.
2. Copy the YAML template from here.

   The YAML template provided in the Advanced Settings dialog is designed for Kubernetes v1.13-1.15 only. For Kubernetes v1.16 or later, copy the v1.16 template from here.

3. Click Save.

STEP 3 | Create a file named webhook.yaml, and open it for editing.

STEP 4 | Paste the YAML into your editor, and replace the following fields:

   - TW-CA-BUNDLE - from the daemonset file downloaded in previous section, search for ca.pem and copy the content to it’s right (after the ":").
   - TW-NAMESPACE - replace with the namespace chosen in previous section (where the Defender is deployed) - if you did not change anything, this would be twistlock

STEP 5 | Create the webhook configuration object.

   After creating the object, the Kubernetes API server directs AdmissionReview requests to Defender.

   $ kubectl apply -f webhook.yaml

Kubernetes webhook templates

The ValidatingWebhookConfiguration provided in Console is designed for Kubernetes v1.13-1.15. Kubernetes v1.13-1.15 ships with v1beta of the admissionregistration.k8s.io API.

For Kubernetes 1.16 and later, Kubernetes ships with v1 of the admissionregistration.k8s.io API. The v1 and v1beta ValidatingWebhookConfiguration objects have some differences. Specifically, the v1 object has two additional required fields:

   - webhooks[0].sideEffects: Required value. Must specify one of None, NoneOnDryRun
   - webhooks[0].admissionReviewVersions: Required value. Must specify one of v1, v1beta1 as an array

When creating your ValidatingWebhookConfiguration object, use the right template.

Kubernetes v1.13 to v1.15:

```yaml
apiVersion: admissionregistration.k8s.io/v1beta1  # v1.13 - 1.15
kind: ValidatingWebhookConfiguration
metadata:
  name: "tw-validating-webhook"
webhooks:
  - name: "validating-webhook.twistlock.com"
    namespaceSelector  
    # matchPolicy: Equivalent # v1.15+
    # timeoutSeconds: 2 # v1.14+
```
Validating your setup

Validate that your webhook has been properly set up with one of the predefined admission rules.

The order in which the rules appear is the order in which they are evaluated. Higher rules take precedence over lower rules. Rules can be reordered. Use the hamburger icon to drag and drop rules into the right place.

**STEP 1 |** Navigate to Defend > Access > Admission and verify there exist default admission rules and they are all enabled by default.

**STEP 2 |** Create the following YAML file to test the Twistlock Labs - CIS - Privileged pod created rule.

1. Create the following YAML file: `priv-pod.yaml`

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
labels:
```
Creating custom admission rules

Use Rego syntax to create custom rules. To learn more about the syntax, review the predefined rules that ship with Prisma Cloud. Rules scripts are based on the admission review input JSON structure. For more information, see: https://github.com/kubernetes/api/blob/master/admission/v1beta1/types.go.

Examples

The following examples should give you some ideas about how you can create your own policies by using the Rego language.

Do not allow new namespaces to be created:

```
match["msg": msg] {
  input.request.operation == "CREATE"
  input.request.kind.kind == "Namespace"
  msg := "It's not allowed to create new namespace!"
}
```

Do not allow a specific image (for example nginx) in new pods:

```
match["msg": msg] {
  input.request.operation == "CREATE"
  input.request.kind.kind == "Pod"
  input.request.resource.resource == "pods"
  input.request.object.spec.containers[0].image == "nginx"
  msg := "It's not allowed to use the nginx Image!"
}
```

Do not allow new pods to expose TCP port 80:

```
match["msg": msg] {
  input.request.operation == "CREATE"
  input.request.kind.kind == "Pod"
  input.request.resource.resource == "pods"
  input.request.object.spec.containers[0].ports[0].containerPort == 80
  msg := "It's not allowed to use port 80 (HTTP) with a Pod configuration!"
}
```
Prisma Cloud integrates security into your continuous integration workflows so you can find and fix problems before they enter production. Prisma Cloud’s CI plugins surface vulnerability and compliance issues directly in the build tool every time developers build their container images and serverless functions. Security teams can set policies that only allow compliant and fully remediated images to progress down the pipeline.

> Jenkins plugin
> Jenkins Freestyle project
> Jenkins Maven project
> Jenkins Pipeline project
> Run Jenkins in a container
> Jenkins pipeline on Kubernetes
> CloudBees Core pipeline on Kubernetes
> CI plugin policy
Jenkins plugin

Prisma Cloud provides a Jenkins plugin that lets you incorporate vulnerability and compliance scanning into your continuous integration pipeline. The plugin scans container images and serverless functions.

The Jenkins plugin can downloaded directly from Console (Manage > System > Downloads). It's also delivered with the release tarball that you download from Releases.

In order to interoperate, both Console and the Jenkins plugin must be from the same release.

The Jenkins plugin is built for Jenkins on Linux. To scan images with Jenkins on other operating systems, use a platform-specific twistcli binary.

Build and scan flow

After Jenkins builds a container image or serverless function package, the Prisma Cloud plugin scans it for vulnerabilities and compliance issues.

Prisma Cloud can pass or fail builds, depending on the types of issues discovered, and the policies set in Console. By incorporating scanning into the build phase of the development workflow, developers get immediate feedback about what needs to be fixed. The scan report provides all the information required to fix the vulnerabilities.

The sequence of events is described below:

1. An developer commits a change, which triggers a build.
2. Jenkins builds the container image.
3. Jenkins calls the Prisma Cloud plugin for scanning. The plugin collects data about the image, including the packages and binaries in the image, and submits it to Console for analysis.
4. Console returns a list of vulnerabilities and compliance issues.
5. The Prisma Cloud plugin passes or fails the build depending upon your policy.

For more information about configuring a scan, see: Setting up a Freestyle project, Setting up a Maven project, or Setting up a Pipeline project.

For more information about targeting rules created in Console to the Jenkins plugin, see Set policy in the CI plugins.

6. Scan results can be reviewed in the following locations:
   - Directly in the Jenkins tool, including the project/job page and dashboard view.
   - In Prisma Cloud Console, in the Monitor > Vulnerabilities > [Images | Functions] > CI pages.

When scanning multiple images in a single build, results do not appear correctly in the Jenkins dashboard view or vulnerability trends table/graph. Only trend data for the last image scanned is shown. Instead, go to Console to see scan results for all images in the build.

Installing the Prisma Cloud Jenkins plugin

Install the Jenkins plugin.

The build console output in Jenkins may show the message - "No CA cert was specified, using insecure connection". This message is generated because twistcli, which the Jenkins plugin wraps, checks the Console’s trust chain by default. When twistcli is run directly, the --tlscacert parameter can be passed to specify the signer, so this message is not shown. To
simplify configuration, the Jenkins plugin doesn’t provide this option, hence why the message is shown. The connection between Jenkins and Console is still fully encrypted with TLS.

The Prisma Cloud Jenkins plugin uses the proxy settings specified in your Jenkins HTTP proxy configuration, which can be found in Manage Jenkins > Manage Plugins > Advanced.

Prerequisites:
• Your version of Jenkins meets Prisma Cloud’s minimum requirements.
• You have installed Prisma Cloud Console on a host in your environment.
• Your Jenkins host can reach Prisma Cloud Console over the network.
• We recommend adding a Prisma Cloud user with the CI User role to minimize privileges on Console. For more information, see User roles.

STEP 1 | Validate that the Jenkins host can communicate with Prisma Cloud Console.

STEP 2 | Open the Jenkins top page.

STEP 3 | Install the Prisma Cloud Jenkins plugin.

The Jenkins plugin can downloaded directly from Console (Manage > System > Downloads). It’s also delivered with the release tarball that you download from Releases.

1. Click Manage Plugins (in the left menu bar), and then click the Advanced tab.
2. Scroll down to Upload Plugin, and click Choose File.
3. Navigate to the folder where you unpacked the Prisma Cloud download and select prisma-cloud-jenkins-plugin.hpi.
4. Click Upload.

STEP 4 | Configure the Prisma Cloud plugin.

1. Go to the Jenkins top page, and then click Manage Jenkins > Configure System.
2. Scroll down to the Prisma Cloud section.
3. In the Address field, enter the URL for Prisma Cloud Console.
4. In the User and Password fields, enter the CI role user’s credentials for Prisma Cloud Console.
5. Click Test Connection to validate that the Jenkins plugin can communicate with Prisma Cloud Console.
6. Click Save.
Scan artifacts

When a build completes, you can view the scan results directly in Jenkins. To support integration with other processes and applications in your organization, Prisma Cloud scan reports can be retrieved from several locations.

Full scan reports for the latest build can be retrieved from:

- The project's workspace, in the scan results file (by the name configured in the scan steps).
- The Prisma Cloud API. For more information, see the endpoint for downloading an image's health (GET /api/v1/images/download).

For example, if you use ThreadFix to maintain a consolidated view of vulnerabilities across all your organization's applications, you could create a post-build action which triggers ThreadFix’s Jenkins plugin to grab Prisma Cloud’s scan report from the project workspace and upload it to the ThreadFix server.

To download the scan report from Console using the Prisma Cloud API, use the following command:

```
$ curl \
-H "Accept: application/json, text/plain" \
-H "Content-type: application.json" \
-u username:password \
> scan_report.csv
```

Ignore image creation time

A common stumbling point is the "Ignore Image Build Time" option. This option checks the time the image was created against the time your Jenkins build started. If the image was not created after the start of your current build, the scan is bypassed. The plugin, by default, scans any image generated as part of your build process, but ignores images not created or updated as part of the build.

Keep in mind the nature of Docker creation time in regards to images. If nothing changes in the image, the creation time isn’t updated. This could lead to a scenario where an image is built and scanned in one job, but not scanned in subsequent jobs because the creation time wasn’t updated because the image didn’t change.

Post build cleanup

Most pipelines push images to the registry after passing Prisma Cloud's vulnerability and compliance scan step. Pipelines also have a final cleanup step that removes images from the local Docker cache. If your build fails, and the pipeline is halted, use a post section to clean up the Docker cache. The post section of a pipeline is guaranteed to run at the end of a pipeline's execution.

For more information, see the Jenkins documentation.

What’s next?

Set up a build job and configure Prisma Cloud to scan the Docker image generated from the job.

For more information, see:

- Jenkins Freestyle project
- Jenkins Maven project
- Jenkins Pipeline project

Notifications of build failures can be enabled using existing Jenkins plugins, for example:

- Mailer plugin
- Jira plugin
• Slack plugin
Jenkins Freestyle project

Jenkins Freestyle projects let you create general-purpose build jobs with maximum flexibility.

Setting up a Freestyle project for container images

Create a Freestyle project that builds a Docker image and then scans it for vulnerability and compliance issues.

**STEP 1** | Go to the Jenkins top page.

**STEP 2** | Create a new project.
1. Click *New Item*.
2. In *Enter an item name*, enter a name for your project.
3. Select *Freestyle project*.
4. Click *OK*.

**STEP 3** | Add a build step.
1. Scroll down to the *Build* section.
2. In the *Add build step* drop-down list, select *Execute shell*.
3. In the *Command* text box, enter the following:

```bash
echo "Creating Dockerfile..."
echo "FROM imiell/bad-dockerfile:latest" > Dockerfile
docker build --no-cache -t test/test-image:0.1 .
```

**STEP 4** | Add a build step that scans the container image(s) for vulnerabilities.
1. In the *Add build step* drop-down list, select *Scan Prisma Cloud Images*.
2. In the *Image* field, select the image to scan by specifying the repository and tag.
   
   Use pattern matching expressions. For example, enter `test/test-image*`.
   
   If the image you want to scan is created outside of this build, or if you want to scan
   the image every build, even if the build might not generate an new image, then click
   *Advanced*, and select *Ignore image creation time*. For more information about
   advanced options, see [here](#).

**STEP 5** | Add a post-build action to publish the scan results in Jenkins directly.
This post-build step depends on a file generated by the previous scan build step, which holds the scan results. This step specifically makes the results available for review in the Jenkins build tool. Note that the previous scan step already published the results in Console, and they’re ready to be reviewed there.
1. Scroll down to *Post-build Actions*.
2. In the *Add post-build action* drop-down menu, select *Publish Prisma Cloud analysis results*.
3. In *Scan Result Files*, accept the default.

Scan result files aren’t deleted by the publish step. They stay in the workspace.

**STEP 6** | Click *Save*.

**STEP 7** | Click *Build Now*. 
STEP 8 | After the build completes, examine the results. Scan reports are available in the following locations:

- Prisma Cloud Console: Log into Console, and go to Monitor > Vulnerabilities > Images > CI.
- Jenkins: Drill down into the build job, then click Image Vulnerabilities to see a detailed report.

Setting up a Freestyle project for serverless functions

The procedure for setting up Jenkins to scan serverless functions is similar to the procedure for container images, except you should use the Scan Prisma Cloud Functions build step.
Where:

- **Function Path** — Path to the ZIP archive of the function to scan.
- **Function Name** — (Optional) String identifier for matching policy rules in Console with the functions being scanned. When creating policy rules in Console, you can target specific rules to specific functions by function name. If this field is left unspecified, the plugin matches the function to the first rule where the function name is a wildcard.

- **AWS CloudFormation template file** — (Optional) Path to CloudFormation template file in either JSON or YAML format. Prisma Cloud scans the function source code for AWS service APIs being used, compares the APIs being used to the function permissions, and reports when functions have permissions for APIs they don't need.
Jenkins Maven project

Create a Maven project that builds a Docker image and then scans it for vulnerability and compliance issues.

Configuring Maven

Configure Maven.

**STEP 1 |** Go to the Jenkins top page.

**STEP 2 |** Click Manage Jenkins.

**STEP 3 |** Select Global Tool Configuration.

**STEP 4 |** Scroll down to the Maven section (Not Maven Configuration), and click Add Maven.

Setting up a Maven project for container images

Set up a Jenkins Maven project.

**STEP 1 |** Go to the Jenkins top page.

**STEP 2 |** Create a new project.

1. Click **New Item**.
2. In **Item name**, enter a name for your project.
3. Select **Maven project**.
4. Click **OK**.

**STEP 3 |** Add a build step.

1. Scroll down to the **Pre steps** section.
2. In the **Add pre-build step** drop-down list, select **Execute shell**.
3. In the **Command** text box, enter the following:

   ```
   echo "Creating Dockerfile..."
   ```
echo "FROM imiell/bad-dockerfile:latest" > Dockerfile
echo 'docker build --no-cache -t test/test-image:0.1 .' > build_image.sh
chmod +x build_image.sh

echo "Creating POM file..."
cat > pom.xml << EOF
<?xml version="1.0" encoding="UTF-8"?>
  <modelVersion>4.0.0</modelVersion>
  <groupId>groupId</groupId>
  <artifactId>artifactid</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>projectName</name>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <build>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <executions>
          <execution>
            <phase>generate-sources</phase>
            <goals>
              <goal>exec</goal>
            </goals>
            <configuration>
              <executable>build_image.sh</executable>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
EOF

STEP 4 | Add a build step that scans the container image(s) for vulnerabilities.

1. In the **Add build step** drop-down list, select **Scan Prisma Cloud Images**.
2. In the **Image** field, select the image to scan by specifying the repository and tag.

   Use **pattern matching expressions**. For example, enter `test/test-image*`.

   ❏ **If the image you want to scan is created outside of this build, or if you want to scan the image every build, even if the build might not generate a new image, then click Advanced, and select Ignore image creation time.**

STEP 5 | Add a post-build action so that image scan results in Jenkins directly.

This post-build step depends on a file generated by the previous scan build step, which holds the scan results. This step specifically makes the results available for review in the Jenkins build tool. Note that the previous scan step already published the results in Console, and they’re ready to be reviewed there.

1. Scroll down to **Post-build Actions**.
2. In the **Add post-build action** drop-down menu, select **Publish Prisma Cloud analysis results**.
3. In the **Scan Result Files** field, accept the default.
   Scan result files aren’t deleted by the publish step. They stay in the workspace.

**STEP 6 |** Click **Save**.

**STEP 7 |** Click **Build Now**.

**STEP 8 |** After the build completes, examine the results. Scan reports are available in the following locations:
- Prisma Cloud Console: Log into Console, and go to **Monitor > Vulnerabilities > Images > CI**.
- Jenkins: Drill down into the build job, then click **Image Vulnerabilities** to see a detailed report.

---

**Setting up a Maven project for serverless functions**

The procedure for setting up Jenkins to scan serverless functions is similar to the procedure for container images, except you should use the **Scan Prisma Cloud Functions** build step.
Where:

- **Function Path** — Path to the ZIP archive of the function to scan.
- **Function Name** — (Optional) String identifier for matching policy rules in Console with the functions being scanned. When creating policy rules in Console, you can target specific rules to specific functions by function name. If this field is left unspecified, the plugin matches the function to the first rule where the function name is a wildcard.
- **AWS CloudFormation template file** — (Optional) Path to CloudFormation template file in either JSON or YAML format. Prisma Cloud scans the function source code for AWS service APIs being used, compares the APIs being used to the function permissions, and reports when functions have permissions for APIs they don’t need.

After a build completes, you can view the scan reports in the following locations:

- Prisma Cloud Console: Log into Console, and go to **Monitor > Vulnerabilities > Functions > CI**.
- Jenkins: Drill down into the build job, then click **Vulnerabilities** to see a detailed report.
Jenkins Pipeline project

The Prisma Cloud Jenkins plugin supports Jenkins Pipeline. Jenkins Pipeline lets you implement and integrate continuous delivery pipelines into Jenkins.

The publish build step depends on the results file generated by scan build step. The results file must be accessible when running the publish step. Therefore, it’s not possible to run both stages (scan and publish) on different nodes or in parallel.

For example, a pipeline script that scans a serverless function and publishes the results (assuming the function zip file exists in the current workspace) looks as follows:

```groovy
node('master') {
  stage('Scan') {
    prismaCloudScanFunction
  }
  stage('Publish') {
    prismaCloudPublish
  }
}
```

Setting up a Pipeline project for container images

To set up a Jenkins Freestyle project:

**STEP 1 |** Go to the Jenkins top page.

**STEP 2 |** Create a new project.

1. Click **New Item**.
2. In **Item name**, enter a name for your project.
3. Select **Pipeline**.
4. Click **OK**.

**Enter an item name**

```plaintext
prisma_cloud_pipeline
```

**Freestyle project**

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Maven project**

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

**Pipeline**

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in a single build type.

**Multi-configuration project**

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

**STEP 3 |** Use Jenkin’s Snippet Generator to generate Pipeline Script for the Prisma Cloud steps.
In the **Pipeline** section, click on the **Pipeline syntax** link, which takes you to https://<PRISMA_CLOUD_CONSOLE>/job/docs_issue/pipeline-syntax/.

**STEP 4 |** Generate Pipeline Script for the scan step.

1. In the **Sample Step** drop-down, select **prismaCloudScanImage - Scan Prisma Cloud Images**.
2. In the **Image** field, select the image to scan by specifying the repository and tag.
   
   Specify the repository and tag using an exact match or **pattern matching expressions**. For example, enter `test/test-image*`.

   ✏️ *If the image you want to scan is created outside of this build, or if you want to scan the image every build, even if the build might not generate an new image, then click **Advanced**, and select **Ignore image creation time**.*

3. Click **Generate Pipeline Script**, copy the snippet, and set it aside.

**STEP 5 |** Generate Pipeline Script to publish the scan results in Jenkins directly.

This post-build step depends on a file generated by the previous scan build step, which holds the scan results. This step specifically makes the results available for review in the Jenkins build tool. Note that the previous scan step already published the results in Console, and they're ready to be reviewed there.

1. In the **Sample Step** drop-down, select **prismaCloudPublish - Publish Prisma Cloud analysis results**.
2. In **Scan Result Files**, accept the default.

   Scan result files aren't deleted by the publish step. They stay in the workspace.

3. Click **Generate Pipeline Script**, copy the snippet, and set it aside.

**STEP 6 |** Return to your project configuration page.

**STEP 7 |** Paste both snippets into the script section for your project configuration.

The following example script builds a simple image, and runs the scan and publish steps.

```
pipeline {
    agent any
```
stages {
    stage('Build') {
        // Build an image for scanning
        sh 'echo "FROM imiell/bad-dockerfile:latest" > Dockerfile'
        sh 'docker build --no-cache -t test/test-image:0.1 .'
    }
    stage('Scan') {
        // Scan the image
        prismaCloudScanImage ca: '',
        cert: '',
        dockerAddress: 'unix:///var/run/docker.sock',
        image: 'test/test-image*',
        key: '',
        logLevel: 'info',
        podmanPath: '',
        project: '',
        resultsFile: 'prisma-cloud-scan-results.json',
        ignoreImageBuildTime:true
    }
}
post { // The post section lets you run the publish step regardless
    of the scan results
    always {
        prismaCloudPublish resultsFilePattern: 'prisma-cloud-scan-
        results.json'
    }
}

STEP 8 | Click Save.

STEP 9 | Click Build Now.

STEP 10 | After the build completes, examine the results.

1. The Status page shows a summary of each build step:
   
   ![Stage View](image)

   - Average stage times:
     - Build: 16s
     - Scan: 1s
     - Publish: 151ms
   - Results:
     - Build: 578ms, 589ms
     - Scan: 3s, 3s
     - Publish: 190ms, 204ms

2. Click on a step to view the log messages for that step:
3. Scan step returned result:

   The criteria for passing or failing a scan is determined by the CI vulnerability and compliance policies set in Console. The default CI vulnerability policy alerts on all CVEs detected. The default CI compliance policy alerts on all critical and high compliance issues.

   There are two reasons why prismaCloudScanImage scan step might return a failed result.

   • The scan failed because the scanner found issues that violate your CI policy.
   • Prisma Cloud Compute Jenkins plugin failed to run due to an error.

   In order to understand the reason for the failure, view the step’s log messages, or move to the Jenkins Console Output page. Another option that can help you differentiate the reason for the failure could be to create preliminary steps to the scan step in order to check the Console’s availability, network connectivity, etc.

   Anyhow, although the return value is ambiguous — you cannot determine the exact reason for the failure by just examining the return value — this setup supports automation. From an automation process perspective, you expect that the entire flow will work. If you scan an image, with or without a threshold, either it works or it does not work. If it fails, for whatever reason, you want to fail everything because there is a problem.

4. Scan reports are available in the following locations:

   • Prisma Cloud Console: Log into Console, and go to Monitor > Vulnerabilities > Images > CI.
   • Jenkins: Drill down into the build job, then click Image Vulnerabilities to see a detailed report.
Setting up a Pipeline project for serverless functions

The procedure for setting up Jenkins to scan serverless functions is similar to the procedure for container images, except select `prismaCloudScanFunction: Scan Prisma Cloud Functions` in the snippet generator.
Where:

- **Function Path (functionPath)** — Path to the ZIP archive of the function to scan.
- **Function Name (functionName)** — (Optional) String identifier for matching policy rules in Console with the functions being scanned. When creating policy rules in Console, you can target specific rules to specific functions by function name. If this field is left unspecified, the plugin will use the function zip file name to match against policy.
- **AWS CloudFormation template file (cloudFormationTemplateFile)** — (Optional) Path to CloudFormation template file in either JSON or YAML format. Prisma Cloud scans the function source code for AWS service APIs being used, compares the APIs being used to the function permissions, and reports when functions have permissions for APIs they don't need.
Run Jenkins in a container

Running Jenkins inside a container is a common setup. This article shows you how to set up Jenkins to run in a container so that it can build and scan Docker images.

Setting up and starting a Jenkins container

To set up Jenkins to run in a container:

**Prerequisite:** You have already installed Docker on the host machine.

**STEP 1** | Create the following Dockerfile. It uses the base Jenkins image and sets up the required permissions for the jenkins user.

```
FROM jenkins/jenkins:lts
USER root
RUN apt-get update \
    && apt-get install -y sudo libltdl17 \
    && rm -rf /var/lib/apt/lists/*
RUN echo "jenkins ALL=NOPASSWD: ALL" >> /etc/sudoers
```

**STEP 2** | Build the image.

```
$ docker build -t jenkins_docker .
```

**STEP 3** | Run the Jenkins container, giving it access to the docker socket.

```
$ docker run -d -v /var/run/docker.sock:/var/run/docker.sock: \
    -v $(which docker):/usr/bin/docker -p 8080:8080 jenkins_docker
```

**STEP 4** | Open a browser and navigate to `<JENKINS_HOST>:8080`.

**STEP 5** | Install the Prisma Cloud plugin.

For more information, see [Jenkins plugin](#).
Jenkins pipeline on Kubernetes

Jenkins is fundamentally architected as a distributed system, with a master that coordinates the builds and agents that do the work. The Kubernetes plugin enables deploying a distributed Jenkins build system to a Kubernetes cluster. Everything required to deploy Jenkins to a Kubernetes cluster is nicely packaged in the Jenkins Helm chart. This article explains how to integrate the Prisma Cloud scanner into a pipeline build running in a Kubernetes cluster.

**Key concepts**

A pipeline is a script that tells Jenkins what to do when your pipeline is run. The Kubernetes Plugin for Jenkins lets you control the creation of the Jenkins slave pod from the pipeline, and add one or more build containers to the slave pod to accommodate build requirements and dependencies.

When the Jenkins master schedules the new build, it creates a new slave pod. Each stage of the build is run in a container in the slave pod. By default, each stage runs in the Jenkins slave (jnlp) container, unless other specified. The following diagram shows a slave pod being launched on a worker node using the Java Network Launch Protocol (JNLP) protocol:

A slave pod is composed of at least one container, which must be the Jenkins jnlp container. Your pipeline defines a podTemplate, which specifies all the containers that make up the Jenkins slave pod. You’ll want your podTemplate to include any images that provide the tools required to execute the build. For example, if one part of your app consists of a C library, then your podTemplate should include a container that provides the GCC toolchain, and the build stage for the library should execute within the context of the GCC container.
The Prisma Cloud Jenkins plugin lets you scan images generated in your pipeline.

**The Prisma Cloud scanner can run inside the default Jenkins jnlp slave container only.**
*It cannot be run within the context of a different container (i.e. from within the container statement block).*

### Scripted Pipeline

This section provides a pipeline script that you can use as a starting point for your own script.

**You cannot run the Prisma Cloud scanner inside a container. The following example snippet will NOT work.**

```java
stage('Prisma Cloud Scan') {
    container('jenkins-slave-twistlock') {
        // THIS DOES NOT WORK
        prismaCloudScanImage ca: '', cert: '', ...
    }
}
```

**Instead, run the Prisma Cloud scanner in the normal context:**

```java
stage('Prisma Cloud Scan') {
    // THIS WILL WORK
    prismaCloudScanImage ca: '', cert: '', ...
}
```

**Prerequisites:**
- You have set up a Kubernetes cluster.
- You have installed Prisma Cloud Console. You can install Prisma Cloud inside or outside of the cluster, as long as any cluster node can reach Console over the network.
- You have installed Jenkins in your cluster. The Jenkins Helm chart is the easiest path for bringing up Jenkins in a Kubernetes cluster.
- Install the Prisma Cloud Jenkins plugin.
Pipeline template

The following template can be used as a starting point for your own scripted pipeline. This template is a fully functional pipeline that pulls the nginx:stable-alpine image from Docker Hub, and then scans it with the Prisma Cloud scanner.

While this example shows how to scan container images, you can also call prismaCloudScanFunction to scan your serverless functions.

```
#!/usr/bin/groovy

podTemplate(label: 'prismaCloud-example-builder', // See 1
containers: [
  containerTemplate(
    name: 'jnlp',
    image: 'jenkinsci/jnlp-slave:3.10-1-alpine',
    args: '${computer.jnlpmac} ${computer.name}'
  ),
  containerTemplate(
    name: 'alpine',
    image: 'twistian/alpine:latest',
    command: 'cat',
    ttyEnabled: true
  ),
],
volumes: [ // See 2
  hostPathVolume(mountPath: '/var/run/docker.sock', hostPath: '/var/run/docker.sock'), // See 3
]
)

node ('prismaCloud-example-builder') {
  stage ('Pull image') { // See 4
    container('alpine') {
      sh ""
      curl --unix-socket /var/run/docker.sock \ // See 5
    }
  }

  stage ('Prisma Cloud scan') { // See 6
    prismaCloudImageScan ca: '',
    cert: '',
    dockerAddress: 'unix://var/run/docker.sock',
    image: 'nginx:stable-alpine',
    resultsFile: 'prisma-cloud-scan-results.xml',
    project: '',
    dockerAddress: 'unix://var/run/docker.sock',
    ignoreImageBuildTime: true,
    key: '',
    logLevel: 'info',
    podmanPath: '',
    project: '',
    resultsFile: 'prisma-cloud-scan-results.json',
    ignoreImageBuildTime: true
  }

  stage ('Prisma Cloud publish') {
```
This template has the following characteristics:

- **1** — This podTemplate defines two containers: the required jnlp-slave container and a custom alpine container. The custom alpine container extends the official alpine image by adding the curl package.
- **2** — The docker socket is mounted into all containers in the pod. For more information about the volumes field, see Pod and container template configuration.
- **3** — By default, the docker socket lets the root user or any member of the docker group read or write to it. The default user in the jnlp container is jenkins. The Prisma Cloud plugin functions need access to the docker socket, so you must add the jenkins user to the docker group. The following listing shows the default permissions for the docker socket:

  ```
  $ ls -l /var/run/docker.sock
  srw-rw----  1 root docker   0 May 30 07:58 docker.sock
  ```
- **4** — The first stage of the build pulls down the nginx image. We run the curl command inside the alpine container because the alpine container was specifically built to provide curl. Note that the prismaCloudScanImage and prismaCloudPublish functions cannot be run inside the container('<NAME>') block. They must be run in the default jnlp container context.
- **5** — There is a lot of debate about docker-in-docker, especially with respect to CI/CD pipelines. In most cases, docker-in-docker is not required for build pipelines. In this example, we run docker commands using the API exposed by the docker socket. Alternatively, we could use a container with just the Docker client installed.
- **6** — The second stage runs the Prisma Cloud scanner on the nginx image in the default jnlp container.

> You can run the Prisma Cloud scanner inside a container using the 'containerized' flag. Scanning from inside a container is only required for special situations.

```groovy
stage('Parallel') {
  agent {
    docker {
      image 'ubuntu:latest'
    }
  }
  stages {
    stage('Prisma Cloud Scan') {
      steps {
        prismaCloudScanImage ca: '', cert: '',
        containerized: true, ...
      }
    }
    ...
  }
}
```

When using the containerized mode, image ID won’t be displayed in the scan results (only image name).
CloudBees Core pipeline on Kubernetes

CloudBees Core is the successor to CloudBees Jenkins Platform and CloudBees Jenkins Enterprise. This article explains how to integrate the Prisma Cloud Jenkins plugin with a CloudBees Core build pipeline running in a Kubernetes cluster.

**Key concepts**

Refer to the article on setting up a Jenkins Pipeline in Kubernetes, as the core concepts are the same. In the case of CloudBees Core on Kubernetes, much of the configuration is already done and the pipeline script is simpler because the JNLP Agent/Slave container is launched automatically. The only tricky bit of configuration is determining the group ID (gid) of the docker group on your Kubernetes hosts, and using it to add some YAML to the default JNLP Agent/Slave pod configuration in CloudBees core. This allows a pod running your pipeline to build and scan images using the mapped Docker socket of the underlying hosts.

Integrating Prisma Cloud

After installing the Prisma Cloud Jenkins plugin, configure the default pod template.

**Prerequisites:**

- You have set up a Kubernetes cluster using the Docker runtime and can SSH to nodes (see gid note below).
- You have installed Prisma Cloud Console. You can install Prisma Cloud inside or outside of the cluster, as long as any cluster node can reach Console over the network.
- You have installed CloudBees Core in your cluster. The CloudBees Core Install Guides are very helpful.
- You’ve built or identified an image for your Docker build executor that contains the docker binary. See an example Dockerfile below.
- Install the Prisma Cloud Jenkins plugin.

**STEP 1** Get the docker group ID (GID) used by the hosts in your Kubernetes cluster.

1. SSH to a node in the cluster.
2. Get the docker group GID. Copy it and set it aside for now.

   ```
   $ sudo grep docker /etc/group
   ```

**STEP 2** Log into the CloudBees Core console, and navigate to `<CLOUDBEES_CONSOLE>/cjoc/view/All/`.

**STEP 3** Click on kubernetes shared cloud.
STEP 4 | In the left navigation bar, click on Configure.

STEP 5 | Scroll down to the Kubernetes pod template section. You'll notice a pod template named default-java with a single container named jnlp.

STEP 6 | Scroll to the bottom of the section. In Raw yaml for the Pod, enter the following snippet, replacing <GID> with the docker GID for your environment.

```yaml
spec:
  securityContext:
    fsGroup: <GID>
```
STEP 7 | Grant all containers in the pod access to the underlying host's Docker socket (unless you do this manually in the pipeline script).
1. Scroll up to the Volumes section.
2. Add a Host Path Volume to the pod template.
3. In both Host path and Mount path, enter /var/run/docker.sock.

STEP 8 | Add a second container to the pod template.
In addition to the JNLP agent/slave, you'll also want to spin up a container with the docker binary inside of it. Use the official docker image from DockerHub and name it build, although you could use any image with the docker client command installed in it. The docker client will use the Docker socket mounted from the underlying host.
1. Scroll up the Container Template section.
2. Click Add Container.
3. In Name, enter build.
4. In Docker image, enter docker.
5. In Working directory, enter /home/jenkins.
6. In Command to run, enter /bin/sh -c.
7. In Arguments to pass to the command, enter cat.
8. Enable Allocate pseudo-TTY.

STEP 9 | Your CloudBees Core pod template config page should look like the following screenshot.
Pipeline template

The following template can be used as a starting point for your own scripted pipeline. This template illustrates how to build a new Docker image and then scan it with the Prisma Cloud scanner. Because the pod template includes a container named 'build' that has the `docker` client command, you can use it in step (1) to build an image.

```json

{ node {

stage ('Build image') { // See 1
    container('build') {
        sh ""
            mkdir myproj
            cd myproj
            echo 'FROM alpine:latest' > Dockerfile
            docker build -t myalpine:latest .
        ""
    }
}

stage ('Prisma Cloud scan') { // See 2
    prismaCloudScanImage
        ca: '',
        cert: '',
        image: 'myalpine:latest',
        resultsFile: 'prisma-cloud-scan-results.xml',
        project: '',
        dockerAddress: 'unix:///var/run/docker.sock',
        ignoreImageBuildTime: true,
        key: '',
        logLevel: 'true',
        timeout: 10,
        podmanPath:''
    }

stage ('Prisma Cloud publish') { // See 3
    prismaCloudPublish resultsFilePattern: 'prisma-cloud-scan-results.xml'
}
}
}
```

This template has the following characteristics:

- **1** — The first stage of the pipeline builds a new container image from a one-line Dockerfile inside the 'build' container specified in the pod config. Note that the `prismaCloudScanImage` and `prismaCloudPublish` functions cannot be run inside the `container('<NAME>')` block. They must be run in the default context.
- **2** — The second stage runs the Prisma Cloud scanner on our newly built image in the default JNLP Agent/Slave container named 'jnlp'.
- **3** — The third stage publishes the scan results to the Prisma Cloud Console.
CI plugin policy

Prisma Cloud lets you centrally define your CI policy in Console. These policies establish security gates at build-time. Use policies to pass or fail builds, and surface security issues early during the development process.

There are two types of policies you can use to target your CI pipeline: vulnerability policies and compliance policies. CI rules have the same parameters as the rules for registries and deployed components, letting you evenly enforce policy in all phases of the app lifecycle.

Prisma Cloud offers the following CI tools:

- A native Jenkins plugin.
- A stand-alone, statically compiled binary, called `twistcli`, that can be integrated with any CI/CD tool.

Vulnerability policy

For more information about the parameters in vulnerability management rules, see here.

Vulnerability rules that target the build tool can allow specific vulnerabilities by creating an exception and setting the effect to 'ignore'. Block them by creating an exception and setting the effect to 'fail'. For example, you could create a vulnerability rule that explicitly allows CVE-2018-1234 to suppress warnings in the scan results.

Rules take effect as soon as they are saved.

Compliance policy

Prisma Cloud’s compliance checks are based on the Center for Internet Security (CIS) Docker Benchmarks. We also provide numerous checks from our lab. You can also implement your own checks using custom checks or XCCDF.

Compliance rules that target the build tool can permit specific compliance issues by setting the action to 'ignore'. They cannot 'fail' a build.

Rules take effect as soon as they are saved.
Firewalls

Prisma Cloud provides layer 4 monitoring and layer 7 firewalling for containers and hosts.

- Web-Application and API Security (WAAS)
- Deploying WAAS
- Cloud Native Network Firewall (CNNF)
Web-Application and API Security (WAAS)

WAAS (Web-Application and API Security, formerly known as CNAF, Cloud Native Application Firewall) is a web application firewall (WAF) designed for HTTP-based web applications deployed directly on hosts, as containers, application embedded or serverless functions. WAFs secure web applications by inspecting and filtering layer 7 traffic to and from the application.

WAAS enhances the traditional WAF protection model by deploying closer to the application, easily scaling up or down and allowing for inspection of "internal" traffic (east-to-west) from other micro-services as well as inbound traffic (north-to-south).

For containerized web applications, WAAS binds to the application’s running containers, regardless of the cloud, orchestrator, node, or IP address where it runs, and without the need to configure any complicated routing. For non-containerized web applications, WAAS simply binds to the host where the application runs.

Highlights of WAAS’s capabilities:

- **OWASP Top-10 Coverage** - protection against most critical security risks to web applications, including injection flaws, broken authentication, broken access control, security misconfigurations, etc.
- **API Protection** - WAAS is able to enforce API traffic security based on definitions/specs provided in the form of Swagger or OpenAPI files.
- **Access Control** - WAAS controls access to protected applications using Geo-based, IP-based or HTTP Header-based user defined restrictions.
- **File Upload Control** - WAAS secures application file uploads by enforcing file extension rules.
- **Detection of Unprotected Web Applications** - WAAS detects unprotected web applications and flags them in the radar view.
- **Penalty Box for Attackers** - WAAS supports a 5 minutes ban of IPs triggering one of it’s protections to slow down vulnerability scanners and other attackers probing the application.

Architecture

WAAS is deployed via Prisma Compute Defenders which operate as a transparent HTTP proxy, evaluating client requests against security policies before relaying the requests to your application. Defenders are deployed into the environment in which the web applications run. WAAS’s management console is independent of the Defenders and can be self-hosted or provided as a service (SaaS):
When a firewall is deployed, Defender reroutes traffic bound for your web application to WAAS for inspection. If a connection is secured with TLS, Defender decrypts the traffic, examines the content, and then re-encrypts it.
Legitimate requests are passed to the target container or host. Requests triggering one or more WAAS protections generate a WAAS "event audit" and an action is taken based on the preconfigured action (see "WAAS Actions" below).

WAAS's event audits can be further explored in the "Monitor" section of Prisma Compute's management console (Monitor > Events). In addition, event audits are registered in the Defender's syslog thus allowing for integration with third-party analytics engines or SIEM platforms of choice.

**WAAS Actions**

Requests that trigger a WAAS protection are subject to one of the following actions:

- **Alert** - The request is passed to the protected application and an audit is generated for visibility.
- **Prevent** - The request is denied from reaching the protected application, an audit is generated and WAAS responds with an HTML page indicating the request was blocked.
- **Ban** - All requests originating from the same IP to the protected application are denied for a time period of 5 minutes following the last detected attack (Penalty Box).

*WAAS implements state, which is required for banning user sessions by IP address. Because Defenders do not share state, any application that is replicated across multiple nodes must enable IP stickiness on the load balancer.*

**Operation**

**Deploying WAAS**

WAAS is enabled by adding a new WAAS rule. Whenever new policies are created, or existing policies are updated, Prisma Cloud immediately pushes them to all the resources to which they apply.

To deploy WAAS, create a new WAAS rule, select the resources on which to apply the rule, define your web application and select the protections to enable. For containerized web applications, Prisma Cloud creates a firewall instance for each container instance. For legacy (non-containerized web applications), Prisma Cloud creates a firewall for each host specified in the configuration.
Supported Protocols, Message Parsers and Decoders

**Supported Protocols**

- HTTP 1.0, 1.1, 2.0 - full support of all HTTP methods
- TLS 1.0, 1.1, 1.2, 1.3
- WebSockets Passthrough

**Supported Message Parsers and Decoders**

- GZip, deflate content encoding
- HTTP Multipart content type
- URL Query, x-www-form-urlencoded, JSON and XML parameter parsing
- URL, HTML Entity, JS, BASE64 decoding
- Overlong UTF-8

**Protection Capabilities**

WAAS provides a rich set of capabilities to protect your web application from attacks.

**Detection of Unprotected Web Applications**

Once a day, the Defender scans its environment for unprotected web applications and marks them on the radar view.

**OWASP Top-10 Protection**

**SQL injection**

An SQL injection (SQLi) attack occurs when an attacker inserts an SQL query into the input fields of a web application. A successful attack can read sensitive data from the database, modify data in the database, or run arbitrary commands.

WAAS parses and tokenizes input streams (request data) and then detects malicious attempts to inject unauthorized SQL queries.

**Cross Site Scripting**

Cross-Site Scripting (XSS) is a type of injection attack, in which malicious JavaScript snippets are injected into otherwise benign and trusted web sites. Attackers try to trick the browser into switching to a Javascript context, and executing arbitrary code.

WAAS parses and tokenizes input streams (request data) and then searches for matching fingerprints of known malicious attack patterns.

**Command & Code Injection**

Command injection is a form of attack in which attackers attempt to run arbitrary commands on the web application’s host. Code injection is a form of attack in which code is injected and interpreted by the application or other runtimes. Command and code payloads are either injected as part of HTTP requests or included from local or remote files (also known as File Inclusion).
WAAS inspects all HTTP requests sent to the application and protects against all types of injection attacks as well as local file inclusions (see more on Local File Inclusion below).

**Prisma Cloud architecture facilitates defense in-depth via multiple protection layers.**

Enabling **Runtime Protection** in addition to WAAS would allow profiling of the application and identifying any anomalies resulting from command or code injections (e.g. unexpected new processes or DNS queries).

### Local File Inclusion

Local File Inclusion is a form of attack in which attackers attempt to gain unauthorized access to locally stored sensitive files on the web application host. Such access attempts are often made using directory traversal attacks or exploiting file inclusion vulnerabilities in the application.

WAAS inspects all HTTP requests sent to the application for local file inclusion attacks aiming at sensitive system files as well as other various traversal attempts.

### Attack Tool & Vulnerability Scanners

Vulnerability scanners are automated tools that scan web applications for known security vulnerabilities and misconfiguration.

Web crawlers are automated tools designed to systematically access and enumerate the content of web applications. Crawling can lead to data breaches by exposing resources that should not be publicly available, or revealing opportunities for hacking by exposing software versions, environment data, and so on.

WAAS is continuously updated with new signatures of widely used web attack arsenal, crawlers and penetration testing tools.

### API Protection

WAAS is able to enforce API security based on specifications provided in the form of Swagger or OpenAPI files. WAAS also allows for manual API definition. E.g. paths, allowed HTTP methods, parameter names, input types, value ranges, etc. Once defined, users can choose WAAS actions to apply for requests which do not comply with the API’s expected behavior.

### Security Misconfigurations

#### Shellshock

Shellshock is a unique privilege escalation vulnerability that permits remote code execution. In unpatched versions of the bash shell interpreter, the Shellshock vulnerability lets attackers create environment variables with specially crafted values that contain code. As soon as the shell is invoked, the attacker’s code is executed.

WAAS checks for requests that are crafted to exploit the Shellshock vulnerability.

For more information about Shellshock, see CVE-2014-6271.

#### Malformed Request Protection

WAAS validates the structure of HTTP requests, automatically blocking those that are malformed.

Examples of malformed requests include:

- HTTP GET requests with a body.
- HTTP POST requests without a `Content-Length` header.

### Cross-site Request Forgery

Cross-site request forgery (CSRF) attacks trick the victim's browser into executing unwanted actions on a web application in which the victim is currently authenticated. WAAS mitigates CSRF attacks by intercepting responses and setting the 'SameSite' cookie attribute value to 'strict'. The 'SameSite' attribute
prevents browsers from sending the cookie along with cross-site requests. It only permits the cookie to be sent along with same-site requests.

There are several techniques for mitigating CSRF, including synchronizer (anti-CSRF) tokens, which developers must implement as part of your web application. The synchronizer token pattern generates random challenge tokens associated with a user’s session. These tokens are inserted into forms as a hidden field, to be submitted along with your forms. If the server cannot validate the token, the server rejects the requested action.

The SameSite cookie attribute works as a complementary defense against CSRF, and helps mitigate against things such as faulty implementation of the synchronizer token pattern.

- When the SameSite attribute is not set, the cookie is always sent.
- With SameSite attribute set to strict, the cookie is never sent in cross-site requests.
- With SameSite attribute set to lax, the cookie is only sent on same-site requests or top-level navigation with a safe HTTP method, such as GET.

It is not sent with cross-domain POST requests or when loading the site in a cross-origin frame. It is sent when you navigate to a site by clicking on a `<a href=…>` link that changes the URL in your browser’s address bar.

Currently, the following browsers support the SameSite attribute:

- Chrome 61 or later.
- Firefox 58 or later.

For more information about the SameSite attribute, see https://tools.ietf.org/html/draft-west-first-party-cookies-07

**Clickjacking**

Web applications that permit their content to be embedded in a frame are at risk of clickjacking attacks. Attackers can exploit permissive settings to invisibly load the target website into their own site and trick users into clicking on links which they never intended to click.

WAAS modifies all response headers, setting the `X-Frame-Options` response header value to `SAMEORIGIN`. The `SAMEORIGIN` directive only permits a page to be displayed in a frame on the same origin as the page itself.

**Access Control**

WAAS can control which applications and end-users can communicate with the protected web application.

**IP-based Access Control**

Administrators can create user-defined Network IP lists and name them e.g. "Office Branches", "Tor and VPN exit nodes", "Business Partners", etc. Network lists can be specified in:

- **Denied inbound IP Sources** - WAAS applies selected action (Alert or Prevent) for IP addresses in network lists
- **IP Exception List** - Traffic originating from IP addresses listed in this category will not be inspected by any of the protections defined in this policy.

*We strongly advise users to practice caution when adding network lists to the IP Exception List as protections would not apply for traffic originating from those IP addresses.*

**Country-based Access Control**

Country codes can be specified in:

- **Denied Inbound Source Countries** - WAAS applies selected action (Alert or Prevent) for requests originating from the specified country.
• **Allowed Inbound Source Countries**  - Requests originating from specified countries would be forwarded to the application (pending inspection). WAAS will apply action of choice (Alert or Prevent) for all other requests not originating from specified countries.

*Origin country is determined by the IP address associated with the request.*

**HTTP Header-based Access Control**

WAAS lets you block or allow requests that contain specific values in HTTP headers. Specify a header and a value to match. The value can be a full or partial string match. Standard **pattern matching** is supported. Pattern matching for this value is same as throughout the product.

Header fields consist of a name, followed by a colon, and then the field value. When decoding field values, WAAS treats commas as delimiters. For example, the *Accept-Encoding* request header advertises which compression algorithm the client supports.

```
Accept-Encoding: gzip, deflate, br
```

WAAS rules do not support exact matching when the value in a multi-value string contains a comma because WAAS treats all commas as delimiters. To match this type of value, use wildcards. For example, consider the following header:

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.108 Safari/537.36
```

To match it, specify the following wildcard expression in your WAAS rule:

```
Mozilla/5.0*
```

**File Uploads**

Attackers may try to upload malicious files (e.g. malware) to your systems. WAAS protects your applications against malware dropping by restricting uploads to just the files that match any allowed content types. All other files will be blocked.

Files are validated both by their extension and their **magic numbers**. Built-in support is provided for the following file types:

- Audio: aac, mp3, wav.
- Compressed archives: 7zip, gzip, rar, zip.
- Documents: odf, pdf, Microsoft Office (legacy, Ooxml).
- Images: bmp, gif, ico, jpeg, png.
- Video: avi, mp4.

WAAS rules let you explicitly allow additional file extensions. These lists provide a mechanism to extend support to file types with no built-in support, and as a fallback in case Prisma Cloud’s built-in inspectors fail to correctly identify a file of a given type. Any file with an allowed extension is automatically permitted through the firewall, regardless of its ‘magic number’.

**Intelligence Gathering**

Error messages give attackers insight into the inner workings of your application. It is therefore important to prevent information leakage.

The following controls limit the exposure of sensitive information.
Brute force protection

WAAS limits the number of POST requests per minute, per IP. If a threshold of more than thirty POST requests is exceeded in a short interval, the source IP address is banned for 5 minutes.

The brute force protection threshold is fixed and cannot be changed by users. This prevents attackers from guessing passwords and flooding your application with unnecessary traffic.

*WAAS implements state, which is required for banning user sessions by IP address. Because Defenders do not share state, any application that is replicated across multiple nodes must enable IP stickiness on the load balancer.*

"Brute-Force Protection" and "Track Response Error Codes" protections share the same count of 30 requests per minute, per IP, per policy. For example, an IP address accessing endpoints protected under the same policy, would get banned for 5 minutes when sending 20 POST requests and receiving 10 error responses from the server, as it would effectively meet the block threshold (20 POST + 10 errors = 30).

Track Response Error Codes

Many failures in rapid succession can indicate that an automated attack is underway. WAAS applies rate-based rules to mitigate these types of attacks. Any HTTP response with a status code equal or greater than 400 is considered as a failure and would be included in the error rate counting. If a threshold of more than thirty errors per minute, per IP address is exceeded, the source IP address is blocked for 5 minutes. The response error codes rate threshold is fixed and cannot be changed by users. If an attacker tries to access non-existing URLs that are known administration pages for various web application frameworks, the source IP address will be immediately blocked for 5 minutes.

*WAAS implements state, which is required for banning user sessions by IP address. Because Defenders do not share state, any application that is replicated across multiple nodes must enable IP stickiness on the load balancer.*

"Brute-Force Protection" and "Track Response Error Codes" Protection share the same count of 30 requests per minute, per IP, per policy. For example, an IP address accessing endpoints protected under the same policy, would get banned for 5 minutes when sending 20 POST requests and receiving 10 error responses from the server, as it would effectively meet the block threshold (20 POST + 10 errors = 30).

Remove Server Fingerprints

By gathering information about the software type and version used by the web application, attackers may learn about potentially known weaknesses and bugs and exploit them.

Eliminating unnecessary headers makes it more difficult for attackers to identify the frameworks that underpin your application.

Response headers that advertise your application’s web server and other server details should be scrubbed. WAAS automatically removes unnecessary headers, such as `X-Powered-By`, `Server`, `X-AspNet-Version`, and `X-AspNetMvc-Version`. 
Detect Information Leakage

WAAS detects situations where the contents of critical files, such as `/etc/shadow`, `/etc/passwd`, and private keys, are contained in responses. WAAS will also detect when responses contain directory listings, output from `php_info()` function calls, and other similar data leakage cases of potentially risky information.
Deploying WAAS

WAAS (Web-Application and API Security) can secure both containerized and non-containerized web applications. To deploy WAAS, create a new rule, and declare the entity to protect.

Although the deployment method varies slightly depending on the type of entity you’re protecting, the steps, in general, are:

1. Define rule resource.
2. Define application scope.
3. Enable relevant protections.

Understanding WAAS rule resources and application scope

The WAAS rule engine is designed to let you tailor the best-suited protection for each part of your deployment. Each rule has two scopes:

- Rule resources.
- Application list.

Rule Resources

This scope defines, for each type of deployment, a combination of one or more elements to which WAAS should attach itself in order to protect the web application:

- **For containerized applications** - Containers, images, namespaces, cloud account IDs, hosts.
- **For non-containerized applications** - Host on which the application is running.
- **For containers protected with App-Embedded Defender** - App ID.
- **For serverless functions** - Function name.

Application List

This scope defines the protected application’s endpoints within the deployment as a combination of one or more of the following:

- **Port (Required)** - For containerized applications, the internal port on which the application is listening. For all other types, the externally facing port.
- **HTTP hostname** - Default setting is set to " (wildcard indicating all hostnames)
- **Base path** - Lets you apply protection policy on certain paths of the application (e.g. "/admin", "/admin/*", etc.)
- **TLS** - TLS certificate to be used when expecting encrypted inbound traffic.

To better illustrate, consider the following deployment scenario for a web application running on-top of an NGINX cluster:
In this example, different policies apply for different parts of the application. The steps for deploying a WAAS rule to protect the above described web application would be as follows:
1. **Define rule resources** - The rule will apply to all containers created from the nginx image.

### Create new WAAS rule

<table>
<thead>
<tr>
<th>Rule name</th>
<th>Example Web Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notes</td>
<td>This is an example used for WAAS documentation</td>
</tr>
<tr>
<td>Containers</td>
<td>Specify a container</td>
</tr>
<tr>
<td>Images</td>
<td>nginxlatest Specify an image</td>
</tr>
<tr>
<td>Cloud Accounts IDs</td>
<td>Specify an account ID</td>
</tr>
<tr>
<td>Labels</td>
<td>Specify a label</td>
</tr>
<tr>
<td>Namespaces</td>
<td>Specify a namespace</td>
</tr>
<tr>
<td>Hosts</td>
<td>Specify a host</td>
</tr>
</tbody>
</table>

2. **Define protection policy for 'login', 'search' and 'product' endpoints** - Set OWASP Top 10 protection to "Prevent" and geo-based access control to "Alert".

3. **Define protection policy for the application's API endpoints** - Set OWASP Top 10 and API protection to "Prevent" and HTTP header-based access control to "Alert".

Once the policy is defined, the rule overview shows the following rule resource and application definitions:
• **Rule Resources** - Protection is applied to all NGINX images
• **Apps List** - We deployed two policies each covering a different endpoint in the application (defined by HTTP hostname, port and path combinations)

## Deploying WAAS

### Deploying WAAS for Containers

To deploy WAAS for containerized web applications, create a new rule, specify the image name, define application endpoints and select protections. WAAS only needs to be applied to images that transmit and receive HTTP/HTTPS traffic.

**STEP 1** | Open Console, and go to **Defend > WAAS**.

**STEP 2** | Select the **Container** tab.

**Defend / WAAS**

<table>
<thead>
<tr>
<th>Container</th>
<th>Host</th>
<th>App Embedded</th>
<th>Serverless</th>
<th>Network Lists</th>
</tr>
</thead>
</table>

**STEP 3** | Click **Add Rule**.

**STEP 4** | Enter a **Rule Name** and **Notes** (Optional) for describing the rule.

**STEP 5** | **Define Rule Resources**.

The rule resource section defines for each type of deployment a combination of image names and one or more elements to which WAAS should attach itself in order to protect the web application:
Applying a rule to all images using a wild card (*) is invalid - instead, only specify your web application images.

**STEP 6** | Click **Add New App**.

**STEP 7** | In the **App Definition** tab, specify the endpoints in your web application that should be protected. Each defined application can have multiple protected endpoints. If you have a Swagger or OpenAPI file, click **Import**, and select the file to load. Otherwise, skip to the next step to manually define your application's endpoints.

<table>
<thead>
<tr>
<th><strong>App Definition</strong></th>
<th><strong>App Firewall</strong></th>
<th><strong>Access Control</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Import Open API/Swagger</strong></td>
<td></td>
<td><strong>Import</strong></td>
</tr>
</tbody>
</table>

**STEP 8** | If you do not have a Swagger or OpenAPI file, manually define each endpoint by specifying the host, port, and path.

1. In the **General App Setup** tab, click **Add Endpoint**.
2. Specify endpoint details:

<table>
<thead>
<tr>
<th>Port</th>
<th>Path</th>
<th>TLS</th>
</tr>
</thead>
</table>

There is no data to show

3. Enter **Port (required)**
   
   Specify the TCP port listening for inbound HTTP traffic.

4. Enter **HTTP Hostname** (optional, wildcards supported).
   
   HTTP host names are specified in the form of [hostname]:[external port].

   External port is defined as the TCP port on the host, listening for inbound HTTP traffic. If the the value of the external port is "80" for non-TLS endpoints or "443" for TLS endpoints it can be omitted. Examples: ".example.site", "docs.example.site", "www.example.site:8080", etc.

5. Enter **Path** (optional, wildcards supported):
   
   Base path for WAAS to match on, when applying protections.

   Examples: "/admin", "/" (root path only), "/\", "/v2/api", etc.
6. If your application uses TLS, set **TLS** to **On**. WAAS must be able to decrypt and inspect HTTPS traffic to function properly. To facilitate that, after creating all endpoints, upload your server’s certificate and private key - concatenate public cert and private key (e.g. `cat server-cert.pem server-key > certs.pem`).

7. If your application uses HTTP/2, set **HTTP/2** to **On**.

8. Click **Create Endpoint**

9. If your application requires **API Protection**, select the “API Protection” tab and define for each path allowed methods, parameters, types, etc. See detailed definition instructions in the **API Protection** section below.

**STEP 9 |** Continue to **App Firewall** tab, select **WAAS protections** to enable and assign them with **WAAS Actions**.

**STEP 10 |** Continue to **Access Control** tab and select **WAAS Access Controls** to enable.

**STEP 11 |** Click **Save**.

**STEP 12 |** You should be redirected to the **Rule Overview** page.

Select the created new rule to display **Rule Resources** and for each application a list of **protected endpoints** and enabled protections.

**STEP 13 |** Test protected endpoint using the following **cURL Test Commands**

**Deploying WAAS for hosts**

To deploy WAAS to protect a host running a non-containerized web application, create a new rule, specify the host(s) where it run, define application endpoints and select protections.

**STEP 1 |** Open Console, and go to **Defend > WAAS**.

**STEP 2 |** Select the **Host** tab
Defend / WAAS

Container  |  Host  |  App Embedded  |  Serverless  |  Network Lists

STEP 3 | Click Add Rule.

STEP 4 | Enter a Rule Name and Notes (Optional) for describing the rule.

STEP 5 | Define Rule Resources.

The rule resource section defines the hosts to which WAAS should attach itself in order to protect the web application:

Enter the rule name

Enter notes

* Specify a host

Applying a rule to all hosts using a wild card (*) is invalid and a waste of resources. WAAS only needs to be applied to hosts that run applications that transmit and receive HTTP/HTTPS traffic.

STEP 6 | Click Add New App.

STEP 7 | In the App Definition tab, specify the endpoints in your web application that should be protected. Each defined application can have multiple protected endpoints. If you have a Swagger or OpenAPI file, click Import, and select the file to load. Otherwise, skip to the next step to manually define your application’s endpoints.
STEP 8 | If you don’t have a Swagger or OpenAPI file, manually define each endpoint by specifying the host, port, and path.

1. In the General App Setup tab, click on Add Endpoint

2. Specify endpoint details:
3. Enter **Port (required)**.

   Specify TCP port, in the container, listening for inbound HTTP traffic.

4. Enter **HTTP Hostname** (optional, wildcards supported).

   HTTP host names are specified in the form of [hostname]:[external port].

   External port is defined as the TCP port on the host, listening for inbound HTTP traffic. If the value of the external port is "80" for non-TLS endpoints or "443" for TLS endpoints it can be omitted.

   Examples: ".example.site", ".docs.example.site", "www.example.site:8080", etc.

5. Enter **Path** (optional, wildcards supported):

   Base path for WAAS to match on when applying protections.

   Examples: "/admin/", "/" (root path only), "/", "/v2/api/", etc.

6. If your application uses TLS, set **TLS** to On. WAAS must be able to decrypt and inspect HTTPS traffic to function properly. To facilitate that, after creating all endpoints, upload your server's certificate and private key - concatenate public cert and private key (e.g. `cat server-cert.pem server-key > certs.pem`).

7. If your application uses HTTP/2, set **HTTP/2** to On.

8. Click **Create Endpoint**

9. If your application requires **API Protection**, select the "API Protection" tab and define for each path allowed methods, parameters, types, etc. See detailed definition instructions in the **API Protection** section below.

**STEP 9** | Continue to **App Firewall** tab, select **WAAS protections** to enable and assign them with **WAAS Actions**.

**STEP 10** | Continue to **Access Control** tab and select **WAAS Access Controls** to enable.

**STEP 11** | Click **Save**.

**STEP 12** | You should be redirected to the **Rule Overview** page.

   Select the created new rule to display **Rule Resources** and for each application a list of **protected endpoints** and **enabled protections**.
STEP 13 | Test protected endpoint using the following cURL Test Commands

**Deploying WAAS for Containers Protected By App-Embedded Defender**

In some environments, Prisma Cloud Defender must be embedded directly inside the container it is protecting. This type of Defender is known as App-Embedded Defender. App-Embedded Defender can secure these types of containers with all WAAS protection capabilities.

The only difference is that App Embedded Defender runs as a reverse proxy to the container it’s protecting. As such, when you set up WAAS for App-Embedded, you must specify the exposed external port where App-Embedded Defender can listen, and the port (not exposed to the Internet) where your web application listens. WAAS for App-Embedded forwards the filtered traffic to your application's port - unless an attack is detected and you set your WAAS for App Embedded rule to **Prevent**.

When testing your Prisma Cloud-protected container, be sure you update the security group's inbound rules to permit TCP connections on the external port you entered in the WAAS rule. This is the exposed port that allows you to access your web application's container. To disable WAAS protection, disable the WAAS rule, and re-expose the application’s real port by modifying the security group’s inbound rule.

To embed App-Embedded WAAS into your container or Fargate task:

**STEP 1 |** Open Console, and go to **Defend > WAAS**.

**STEP 2 |** Select the **App Embedded** tab.

**STEP 3 |** Click **Add Rule**.
STEP 4 | Enter a **Rule Name** and **Notes** (Optional) for describing the rule.

STEP 5 | Define **Rule Resources**.

The rule resource section defines the App IDs to which WAAS should attach itself in order to protect the web application:

**Name**

Enter the rule name

**Notes**

Enter notes

**Specify an app ID**

STEP 6 | Click **Add New App**.

STEP 7 | In the App Definition tab, specify the endpoints in your web application that should be protected. Each defined application can have multiple protected endpoints. If you have a Swagger or OpenAPI file, click Import, and select the file to load. Otherwise, skip to the next step to manually define your app’s endpoints.

**App Definition**

| App Firewall | Access Control |

Import Open API/Swagger

**STEP 8** | If you don’t have a Swagger or OpenAPI file, manually define each endpoint by specifying the host, port, and path.

1. In the **General App Setup** tab, click on **Add Endpoint**.
2. Specify endpoint details:

<table>
<thead>
<tr>
<th>Port</th>
<th>External Port</th>
<th>Path</th>
<th>TLS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Add [host]:[external port]</td>
<td>Port</td>
<td>Add [internal port]</td>
</tr>
<tr>
<td></td>
<td>Application path</td>
<td>External port</td>
<td>External port</td>
</tr>
<tr>
<td></td>
<td>HTTP/2</td>
<td></td>
<td>Off</td>
</tr>
</tbody>
</table>

3. Enter Port (required)

Specify TCP port, in the container, listening for inbound HTTP traffic

4. Enter External Port (required).

External port is the TCP port for the App-Embedded Defender to listen on for inbound HTTP traffic.

5. Enter HTTP Hostname (optional, wildcards supported).

HTTP host names are specified in the form of [hostname]:[external port].

External port is defined as the TCP port on the host, listening for inbound HTTP traffic. If the the value of the external port is "80" non-TLS endpoints or "443" for TLS endpoints it can be omitted. Examples: "*.example.com", "docs.example.com", "www.example.com:8080", etc.

6. Enter Path (optional, wildcards supported):
Base path for WAAS to match on when applying protections.

Examples: "/admin/*", "/" (root path only), "/v2/api/*", etc.

7. If your application uses TLS, set TLS to On. WAAS must be able to decrypt and inspect HTTPS traffic to function properly. To facilitate that, after creating all endpoints, upload your server’s certificate and private key - concatenate public cert and private key (e.g. cat server-cert.pem server-key > certs.pem).

8. If your application uses HTTP/2, set HTTP/2 to On.
9. Click Create Endpoint
10. If your application requires API Protection, select the "API Protection" tab and define for each path allowed methods, parameters, types, etc. See detailed definition instructions in the API Protection section below.

STEP 9 | Continue to App Firewall tab, select WAAS protections to enable and assign them with WAAS Actions.

STEP 10 | Continue to Access Control tab and select WAAS Access Controls to enable.

STEP 11 | Click Save.

STEP 12 | You should be redirected to the Rule Overview page.

Select the new rule to display Rule Resources and for each application a list of protected endpoints and enabled protections.

STEP 13 | Test protected container using the following cURL Test Commands

Deploying WAAS for serverless functions

When Serverless Defender is embedded in a function, it offers built-in web application firewall (WAF) capabilities, including protection against:

- SQL injection (SQLi) attacks
- Cross-site scripting (XSS) attacks
- Command injection (CMDi) attacks
- Local file system inclusion (LFI) attacks
- Code injection attacks

Some WAAS protections are not available for WAAS serverless deployment.

Prerequisites: You already embedded Serverless Defender into your function.

**STEP 1** | Open Console and go to Defend > WAAS > Serverless.

**STEP 2** | Click Add rule.

**STEP 3** | Enter a rule name.

**STEP 4** | Select Alert or Prevent.

**STEP 5** | Select the protections to enable.

**STEP 6** | Enter the functions to protect.
Use pattern matching to precisely target your rule.

**WAAS Actions**

HTTP requests that trigger WAAS protections are subject to one of the following actions:

- **Alert** - Request is passed to the protected application and an audit is generated for visibility.
- **Prevent** - Request is denied from reaching the protected application, an audit is generated and WAAS responds with an HTML banner indicating the request was blocked.
- **Ban** - All requests originating from the same IP address to the protected application are denied for a time period of 5 minutes since the last detected attack (Penalty Box).

WAAS implements state, which is required for banning user sessions by IP address. Because Defenders do not share state, any application that is replicated across multiple nodes must enable IP address stickiness on the load balancer.
## WAAS protections

### Firewall settings

<table>
<thead>
<tr>
<th>Protection</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Injection</td>
<td>Disable, Alert</td>
</tr>
<tr>
<td>Cross-Site Scripting (XSS)</td>
<td>Disable, Alert</td>
</tr>
<tr>
<td>OS Command Injection</td>
<td>Disable, Alert</td>
</tr>
<tr>
<td>Code Injection</td>
<td>Disable, Alert</td>
</tr>
<tr>
<td>Local File Inclusion</td>
<td>Disable, Alert</td>
</tr>
<tr>
<td>Attack Tools &amp; Vulnerability Scanners</td>
<td>Disable, Alert</td>
</tr>
<tr>
<td>Shellshock Protection</td>
<td>Disable, Alert</td>
</tr>
<tr>
<td>Malformed HTTP Request</td>
<td>Disable, Alert</td>
</tr>
<tr>
<td>Prisma Cloud Advanced Threat Protection</td>
<td>Disable, Alert</td>
</tr>
<tr>
<td>API Protection</td>
<td>Disable, Alert</td>
</tr>
<tr>
<td>Detect Information Leakage</td>
<td>Disable, Alert</td>
</tr>
<tr>
<td>Cross Site Request Forgery Protection</td>
<td>On</td>
</tr>
<tr>
<td>Clickjacking Prevention</td>
<td>On</td>
</tr>
<tr>
<td>Brute-Force Prevention</td>
<td>Off</td>
</tr>
<tr>
<td>Track Server Error Response Codes</td>
<td>Off</td>
</tr>
<tr>
<td>Remove Server Fingerprints</td>
<td>On</td>
</tr>
</tbody>
</table>

### OWASP Top 10 protection

WAAS offers protection for the critical security risks described in the OWASP Top Ten list.

#### SQL injection

An SQL injection (SQLi) attack occurs when an attacker inserts an SQL query into the input fields of a web application. A successful attack can read sensitive data from the database, modify data in the database, or run arbitrary commands.
WAAS parses and tokenizes input streams (request data) and then detects malicious attempts to inject unauthorized SQL queries.

**Cross Site Scripting**

Cross-Site Scripting (XSS) is a type of injection attack, in which malicious JavaScript snippets are injected into otherwise benign and trusted web sites. Attackers try to trick the browser into switching to a Javascript context, and executing arbitrary code.

WAAS parses and tokenizes input streams (request data) and then searches for matching fingerprints of known malicious attack patterns.

**Command & Code Injection**

Command injection is a form of attack in which attackers attempt to run arbitrary commands on the web application’s host. Code injection is a form of attack in which code is injected and interpreted by the application or other runtimes. Command and code payloads are either injected as part of HTTP requests or included from local or remote files (also known as File Inclusion).

WAAS inspects all HTTP requests sent to the application and protects against all types of injection attacks as well as local file inclusions.

Prisma Cloud architecture facilitates defense in-depth via multiple protection layers. Enabling Runtime Protection in addition to WAAS would allow profiling of the application and identifying any anomalies resulting from command or code injections (e.g. unexpected new processes or DNS queries).

**Local File Inclusion**

Local File Inclusion is a form of attack in which attackers attempt to gain unauthorized access to locally stored sensitive files on the web application host. Such access attempts are often made using directory traversal attacks or exploiting file inclusion vulnerabilities in the application.

WAAS inspects all HTTP requests sent to the application for local file inclusion attacks aiming at sensitive system files as well as other various traversal attempts.

**Attack Tool & Vulnerability Scanners**

Vulnerability scanners are automated tools that scan web applications for known security vulnerabilities and misconfiguration.

Web crawlers are automated tools designed to systematically access and enumerate the content of web applications. Crawling can lead to data breaches by exposing resources that should not be publicly available, or revealing opportunities for hacking by exposing software versions, environment data, and so on.

WAAS is continuously updated with new signatures of widely used web attack arsenal, crawlers and penetration testing tools.

**API Protection**

WAAS is able to enforce API security based on specifications provided in the form of Swagger or OpenAPI files. WAAS also allows for manual API definition. E.g. paths, allowed HTTP methods, parameter names, input types, value ranges, etc. Once defined, users can choose WAAS actions to apply for requests which do not comply with the API’s expected behavior.

**Importing API Definition From Swagger or OpenAPI**

1. Enter App Definition Tab.
2. Click on Import.
3. Select definition file to load.
4. Select API Protection Tab.
5. Review path and parameter definitions
6. Enter App Firewall Tab.
7. Assign API Protection protection relevant action.

Manual API Definition

1. Enter App Definition Tab.
2. Click Add Endpoint and enter API HTTP hostnames and base paths.

   Base path in the endpoint definition should always end with a * e.g. "/**", "/api/v2/**". If not configured that way, API protection would not apply to sub-paths defined in the API protection tab.

1. Select API Protection Tab.
You can create the App definition by importing an Open API/Swagger or by adding manually. Importing from a file will overwrite previous manually added API Entries.

2. Click **Add Path**
3. Enter **Resource Path** (e.g. `/product`)
   Paths entered in this section are additional subpaths to the base path defined in the previous endpoint section. For example, if in the endpoint definition hostname was set to "www.example.com", base path set to "/api/v2/*" and in the API Protection tab resource path set to "/product" - full protected resource would be www.example.com/api/v2/product.
4. Select allowed **HTTP Methods**.
5. For each allowed HTTP method, define parameters by selecting the method from **Parameters for** dropdown list.
6. For each HTTP method add allowed parameters:
   1. Click **Add Parameter**
   2. Enter parameter definition

7. Enter **App Firewall Tab**
8. Assign **API Protection** protection relevant action

   To apply actions on requests that do not include mandatory parameters, make sure to set the **Required** toggle switch to **On** for all mandatory parameters.

### Security Misconfigurations

#### Shellshock

Shellshock is a unique privilege escalation vulnerability that permits remote code execution. In unpatched versions of the bash shell interpreter, the Shellshock vulnerability lets attackers create environment variables with specially crafted values that contain code. As soon as the shell is invoked, the attacker’s code is executed.
WAAS checks for requests that are crafted to exploit the Shellshock vulnerability. For more information about Shellshock, see CVE-2014-6271.

**Malformed Request Protection**

WAAS validates the structure of HTTP requests, automatically blocking those that are malformed.

Examples of malformed requests include:

- HTTP GET requests with a body.
- HTTP POST requests without a `Content-Length` header.

**Cross-site Request Forgery**

Cross-site request forgery (CSRF) attacks trick the victim's browser into executing unwanted actions on a web application in which the victim is currently authenticated. WAAS mitigates CSRF attacks by intercepting responses and setting the 'SameSite' cookie attribute value to 'strict'. The 'SameSite' attribute prevents browsers from sending the cookie along with cross-site requests. It only permits the cookie to be sent along with same-site requests.

There are several techniques for mitigating CSRF, including synchronizer (anti-CSRF) tokens, which developers must implement as part of your web application. The synchronizer token pattern generates random challenge tokens associated with a user's session. These tokens are inserted into forms as a hidden field, to be submitted along with your forms. If the server cannot validate the token, the server rejects the requested action.

The SameSite cookie attribute works as a complementary defense against CSRF, and helps mitigate against things such as faulty implementation of the synchronizer token pattern.

- When the SameSite attribute is not set, the cookie is always sent.
- With SameSite attribute set to strict, the cookie is never sent in cross-site requests.
- With SameSite attribute set to lax, the cookie is only sent on same-site requests or top-level navigation with a safe HTTP method, such as GET.

It is not sent with cross-domain POST requests or when loading the site in a cross-origin frame. It is sent when you navigate to a site by clicking on a `<a href=...>` link that changes the URL in your browser's address bar.

Currently, the following browsers support the SameSite attribute:

- Chrome 61 or later.
- Firefox 58 or later.


**Clickjacking**

Web applications that permit their content to be embedded in a frame are at risk of clickjacking attacks. Attackers can exploit permissive settings to invisibly load the target website into their own site and trick users into clicking on links which they never intended to click.

WAAS modifies all response headers, setting the `X-Frame-Options` response header value to `SAMEORIGIN`. The `SAMEORIGIN` directive only permits a page to be displayed in a frame on the same origin as the page itself.

**Intelligence Gathering**

Error messages give attackers insight into the inner workings of your application. It is therefore important to prevent information leakage.

The following controls limit the exposure of sensitive information.
**Brute Force Protection**

WAAS limits the number of POST requests per minute per IP. If a threshold of more than thirty POST requests is exceeded in a short interval, the source IP address is banned for 5 minutes.

The brute force protection threshold is fixed and cannot be changed by users. This prevents attackers from guessing passwords and flooding your application with unnecessary traffic.

> WAAS implements state, which is required for banning user sessions by IP address. Because Defenders do not share state, any application that is replicated across multiple nodes must enable IP stickiness on the load balancer.

> "Brute-Force Protection" and "Track Response Error Codes" protections share the same count of 30 requests per minute per IP, per policy. For example, an IP address accessing endpoints protected under the same policy, would get banned for 5 minutes when sending 20 POST requests and receiving 10 error responses from the server, as it would effectively meet the block threshold (20 POST + 10 errors = 30).

**Track Response Error Codes**

Many failures in rapid succession can indicate that an automated attack is underway. WAAS applies rate-based rules to mitigate these types of attacks. Any HTTP response with a status code equal or greater than 400 is considered as a failure and would be included in the error rate counting. If a threshold of more than thirty errors per minute, per IP address is exceeded, the source IP address is blocked for 5 minutes. The response error codes rate threshold is fixed and cannot be changed by users. If an attacker tries to access non-existing URLs that are known administration pages for various web application frameworks, the source IP address will be immediately blocked for 5 minutes.

> WAAS implements state, which is required for banning user sessions by IP address. Because Defenders do not share state, any application that is replicated across multiple nodes must enable IP stickiness on the load balancer.

> "Brute-Force Protection" and "Track Response Error Codes" Protection share the same count of 30 requests per minute, per IP, per policy. For example, an IP address accessing endpoints protected under the same policy, would get banned for 5 minutes when sending 20 POST requests and receiving 10 error responses from the server, as it would effectively meet the block threshold (20 POST + 10 errors = 30).

**Remove Server Fingerprints**

By gathering information about the software type and version used by the web application, attackers may learn about potentially known weaknesses and bugs and exploit them.

Eliminating unnecessary headers makes it more difficult for attackers to identify the frameworks that underpin your application.

Response headers that advertise your application's web server and other server details should be scrubbed. WAAS automatically removes unnecessary headers, such as X-Powered-By, Server, X-AspNet-Version, and X-AspNetMvc-Version.
Detect Information Leakage

WAAS detects situations where the contents of critical files, such as /etc/shadow, /etc/passwd, and private keys, are contained in responses. WAAS will also detect when responses contain directory listings, output from php_info() function calls, and other similar data leakage cases of potentially risky information.

WAAS Access Controls

WAAS allows for control over how applications and end-users communicate with the protected web application.

Network Lists

Network Lists allow administrators to create and maintain named IP address lists e.g. "Office Branches", "Tor and VPN Exit Nodes", "Business Partners", etc. List entries are composed of IPv4 addresses or IP CIDR blocks.

To access Network Lists, open Console, go to Defend > WAAS and select the Network List tab.
Lists can be updated manually or via batch importing of entries from a CSV file. Once defined, Network Lists can be referenced and used in IP-based access control.

To export lists in CSV format, click export CSV.

*When importing IP addresses or IP CIDR blocks from a CSV file, first record value should be set to "ip" (case sensitive).*

*IPv6 entries are currently not supported.*
Network Controls

App Firewall  |  Access Control

HTTP Headers  |  File Uploads

**Network Controls**

**IP-based access control**

Network lists can be specified in:

- **Denied inbound IP Sources** - WAAS applies selected action (Alert or Prevent) for IP addresses in network lists.
- **IP Exception List** - Traffic originating from IP addresses listed in this category will not be inspected by any of the protections defined in this policy.

We strongly advise users to practice caution when adding network lists to the IP Exception List as protections will not be applied for traffic originating from these IP addresses.

**Country-Based Access Control**

Specify country codes, ISO 3166-1 alpha-2 format, in one of the following categories (mutually exclusive):

- **Denied Inbound Source Countries** - WAAS applies selected action (Alert or Prevent) for requests originating from the specified countries.
- **Allowed Inbound Source Countries** - Requests originating from specified countries will be forwarded to the application (pending inspection). WAAS will apply action of choice (Alert or Prevent) on all other requests not originating from the specified countries.

Country of origin is determined by the IP address associated with the request.
HTTP Header Controls

<table>
<thead>
<tr>
<th>HTTP Header Name</th>
<th>Value</th>
<th>Required</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Header name**

  - **Allowed**
  - **Blocklisted**

  - **Alert**
  - **Prevent**

  - **Comma separated values**

  - **Off**

WAAS lets you block or allow requests which contain specific strings in HTTP headers by specifying a header name and a value to match. The value can be a full or partial string match. Standard pattern matching is supported.

If the **Required** toggle is set to **On** WAAS will apply the defined action on HTTP requests in which the specified HTTP header is missing. When the **Required** toggle is set to **Off** no action will be applied for HTTP requests missing the specified HTTP header.

HTTP Header fields consist of a name, followed by a colon, and then the field value. When decoding field values, WAAS treats all commas as delimiters. For example, the **Accept-Encoding** request header advertises which compression algorithm the client supports.

- **Accept-Encoding**: gzip, deflate, br

WAAS rules do not support exact matching when the value in a multi-value string contains a comma because WAAS treats all commas as delimiters. To match this type of value, use wildcards. For example, consider the following header:

- **User-Agent**: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.108 Safari/537.36
To match it, specify the following wildcard expression in your WAAS rule:

Mozilla/5.0*

**File Upload Controls**

**File Uploads**

- **Extensions**
  - List of allowed extensions without leading dot (e.g., jpg, docx, zip)
  - Action for all others

<table>
<thead>
<tr>
<th>Compressed archives</th>
<th>Documents</th>
<th>Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>7zip</td>
<td>odf</td>
<td>bmp</td>
</tr>
<tr>
<td>gzip</td>
<td>Office legacy</td>
<td>gif</td>
</tr>
<tr>
<td>rar</td>
<td>Office Open XML</td>
<td>ico</td>
</tr>
<tr>
<td>zip</td>
<td>pdf</td>
<td>jpeg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>png</td>
</tr>
</tbody>
</table>

Attackers may try to upload malicious files (e.g. malware) to your systems. WAAS protects your applications against malware dropping by restricting uploads to just the files that match any allowed content types. All other files will be blocked.

Files are validated both by their extension and their **magic numbers**. Built-in support is provided for the following file types:

- **Audio**: aac, mp3, wav.
- **Compressed archives**: 7zip, gzip, rar, zip.
- **Documents**: odf, pdf, Microsoft Office (legacy, Ooxml).
- **Images**: bmp, gif, ico, jpeg, png.
- **Video**: avi, mp4.

WAAS rules let you explicitly allow additional file extensions. These lists provide a mechanism to extend support to file types with no built-in support, and as a fallback in case Prisma Cloud's built-in inspectors fail to correctly identify a file of a given type. Any file with an allowed extension is automatically permitted through the firewall, regardless of its 'magic number'.

**cURL Test Commands**

Below are curl-based tests that can be used to verify endpoints have been properly defined. Make sure all changes are saved prior to running these tests. The method for verifying test results differs according to the selected action:

- **Alert** - Go to Monitor > Events to see alerts logged by Prisma Cloud relating to this policy violation.
- **Prevent** - Commands return output similar to the following:

```
HTTP/1.1 403 Forbidden
Date: Wed, 15 Jul 2020 12:51:50 GMT
Content-Type: text/html; charset=utf-8
```

In the following examples, replace `<http_hostname>` with your endpoint’s hostname and `<external_port>` with the web facing port of your application. For testing HTTP header access control, also replace `<http_header_name>` with the header name set in the rule and `<http_header_value>` with set values.

### SQL injection:

```
curl -I http://<http_hostname>:<external_port>/\?id\=%27%20OR%20%271
```

### Cross-site scripting:

```
curl -I http://<http_hostname>:<external_port>/\?id\=\<script\>alert \((1)\)/script\>
```

### OS command injection:

```
curl -I http://<http_hostname>:<external_port>/\?id\=/bin/sh/
```

### Code injection:

```
curl -I http://<http_hostname>:<external_port>/\?id\=phpinfo()
```

### Local file inclusion:

```
curl -I http://<http_hostname>:<external_port>/\?id\=../etc/passwd
```

### Attack tools and vulnerability scanners:

```
```

### Shellshock protection:

```
```

### Malformed HTTP request:

```
curl -s -i -X GET -o /dev/null -D -d '"test":"test"'} http://<http_hostname>:<external_port>/
```

### HTTP header access controls:

```
curl -H '"header_Name": "header_value"'} http://<http_hostname>:<external_port>/
```
Cloud Native Network Firewall (CNNF)

Cloud Native Network Firewall (CNNF) is a Layer 4 network monitoring tool. CNNF automatically discovers how entities in your environment communicate, and shows the communication mesh on Radar. Radar has a container view, which shows the network topology for your containerized apps. Radar also has a host view, which shows the network topology for hosts.

Architecture

Defender monitors how your containers and hosts connect to each other in real-time.

Defender inspects connections before they’re set up. After a connection is established, traffic flows directly between source and destination without any further oversight from Defender.

Defender adds iptables rules to observe TCP’s three-way handshake. The three-way handshake sets up new connections using SYN messages. For each pod or container IP address, Defender adds an iptables rule with the target set to NFQUEUE. NFQUEUE is an iptables target which delegates the decision of how to handle a packet to a userspace program (in this case Defender). When SYN messages arrive, Defender evaluates them to track connections. From this vantage point, Defender can see all connections.
The following screenshot shows the mesh Radar draws for a typical microservices app. Radar draws directed edges for each connection, and displays the associated port number. An instance count for each node shows how many copies of the image are running as containers. Black bubbles mean the runtime model is in enforcement mode. Blue bubbles mean the runtime model is in learning mode.

Network objects

You might have resources that interact with external, non-containerized services. For example, a payment gateway might pass information to an external service to verify transactions.

Network objects represent sources and destinations. Network objects can represent container images, subnets, and hosts. You can create new network objects representing a range of IP addresses or a single IP. Radar shows any connections established to the network object:

To create a network object, go to Defend > Firewalls > Cloud Native Network Firewall, click Add Network Object, and specify an IP address or subnet.

Enabling CNNF

CNNF runs in one of two modes: Disabled or Enabled.

- **Disabled**

  CNNF displays limited traffic flow data on Radar, including outbound connections to the Internet and connections local to the node itself. By default, CNNF ships in the disabled state.

- **Enabled**
CNNF monitors all connections, including connections across hosts and connections to any configured network objects.

**STEP 1** |  Open Console.

**STEP 2** |  Go to Defend > Firewalls > Cloud Native Network Firewall.

**STEP 3** |  Enable CNNF for hosts and containers.
Prisma Cloud integrates with the secrets management tools, such as Hashicorp Vault, CyberArk Enterprise Password Vault, AWS Secrets Manager, and Microsoft Azure Key Vault, to ensure the safe distribution of secrets. Compliance checks let you detect and prevent unsafe usage of secrets.

- Secrets manager
- Integrate with secrets stores
- Inject secrets into containers
- Injecting secrets: end-to-end example
Secrets manager

Containers often require sensitive information, such as passwords, SSH keys, encryption keys, and so on. You can integrate Prisma Cloud with HashiCorp Vault or CyberArk Enterprise Password Vault to securely distribute secrets from those stores to the containers that need them.

Enterprise secret stores reduce the risk of data breaches that could occur when sensitive information is littered across an organization, often in places where they should not be, such as email inboxes, source code repositories, developer workstations, and Dropbox. Secret stores provide a central and secure location for managing and distributing secrets to the apps that need them. They give you a way to account for all the secrets in your organization with audit trails that show how they are being used.

Orchestrators, such as Docker Swarm and Kubernetes, offer their own built-in secret stores with support for creating secrets, listing them, deleting them, and injecting them into containers. However, if you already have an enterprise secrets store, then you probably want to extend it to handle to your container environment. Utilizing the orchestrator’s built-in secrets management capabilities when you already have an enterprise secrets store is unappealing because it represents another silo of sensitive information that needs to be carefully secured. It undermines the principal benefit of a secrets store, which is managing all sensitive data in a single location.

Theory of operation

When a user or orchestrator starts a container, Defender injects the secrets you specify into the container. In order for secret injection to work, all Docker commands must be routed through Defender, so be sure to download your client certs and setup your environment variables.

There are several moving parts in the solution:
A. Secret values are fetched, encrypted, and stored in Console's database when a rule is created or modified. Secret values in Console's database are periodically synced with the secrets store to provide resiliency in the case of connectivity outages and to optimize performance. All secrets cached on disk, both in Defender and in Console, are protected with 256 bit AES encryption. If you change a secret's value in the secrets store and you need it synced immediately, you can click the refresh button in the Console UI to refetch all secrets from their configured stores.

1. Operator (or orchestrator) starts a container with docker run.

2. Defender assesses the command against the policy installed in Console.

3. The secret is returned to Defender. Defender starts the container using the Docker API, which is exposed through the Docker daemon's local UNIX socket, and injects the secret into the container.

Capabilities

The Prisma Cloud secrets manager has the following capabilities:

- Supports integration with HashiCorp Vault and CyberArk Enterprise Password Vault.
- Manages the distribution of secrets from the secret store to your containers through policies. In Console, you create the rules that control which secrets get injected into which containers.
- Injects secrets into containers as either environment variables or files.
- Secrets injected as environment variables are presented in-the-clear from within the container. They are redacted from the outside to prevent exposure by the docker inspect command.
- Secrets injected as files are provided from an in-memory filesystem (on /run/secrets/<SECRET_NAME>) that is mounted into the container when it is created. When the container is stopped, the secrets directory is unmounted and deleted.

Best practices

There are a number of ways that secrets can be compromised.

**Defender is bypassed.** If Defender is bypassed, an attacker can execute Docker commands directly against the Docker daemon's local UNIX socket, and he will be able to expose your secrets. Be sure that your hosts are secured with least privilege access so that users can only run docker commands through Defender.

Limit lower privileged users to monitoring commands, such as docker ps and docker inspect. Prisma Cloud automatically encrypts secrets injected as environment variables when accessed from docker inspect. Restrict commands such as docker exec and docker run to just the operators that need them because these commands can reveal secrets injected into a container by giving the user shell access inside the container, where variables are in the clear. For example, docker exec printenv on a running container, or docker run <IMAGE_ID> printenv on an image, can reveal environment variables that are otherwise encrypted with docker inspect. The following diagram shows one way to grant access to Docker functions based on a user's role. This is the way that Docker Datacenter Universal Control Plane (UCP) grants permissions, and you can implement the same scheme with Prisma Cloud's access control rules.
Integrate with secrets stores

To inject secrets into your containers, you must first integrate Prisma Cloud with your secrets manager, and then set up rules for injecting specific secrets into specific containers.

Prisma Cloud can integrate with the following secrets managers:

- AWS Secrets Manager
- AWS Systems Manager Parameters Store
- Azure Key Vault
- CyberArk Enterprise Password Vault
- HashiCorp Vault (versions 0.9.x and older, and versions 0.10 and later)

Refresh interval

By default, the refresh interval is disabled. That means if you change a secret’s value in the secrets store, you must force Prisma Cloud to update its list of values. In Console, go to Defend > Access > Secrets and click Refresh secrets to force Prisma Cloud to fetch the latest values of all secrets from their configured stores.

You can also configure Prisma Cloud to periodically retrieve the latest values of all the secrets from their stores. In Console, go to Manage > Authentication > Secrets, click Edit next to the Secrets refresh interval field, and specify an integer value in hours. Setting the refresh interval to 0 disables automatic periodic refreshes.
Secrets Stores

Integrate Prisma Cloud with the supported secrets management stores.

AWS Secrets Manager

You can integrate Prisma Cloud with AWS Secrets Manager. First, configure Prisma Cloud to access AWS Secrets Manager, then create rules to inject the relevant secrets into the relevant containers.

Prerequisites:

- The service account Prisma Cloud uses to access the secrets store must have the following permissions:
  - secretsmanager:GetSecretValue
  - secretsmanager:ListSecrets
- You have created a secret in AWS Secrets Manager. Automatic rotation must be disabled. Prisma Cloud supports the key-value secret type only. When storing a new secret, select Other type of secrets, then Secret key/value.

### Select secret type

- Credentials for RDS database
- Credentials for other database
- Other type of secrets (e.g. API key)

### Specify the key/value pairs to be stored for this secret

<table>
<thead>
<tr>
<th>Secret key/value</th>
<th>Plaintext</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysecret</td>
<td>12345</td>
</tr>
</tbody>
</table>

**STEP 1** | Open Prisma Cloud Console.

**STEP 2** | Integrate Prisma Cloud with the secrets store.

  1. Go to Manage > Authentication > Secrets, and click Add store.
2. Enter a name for the store. This name is used when you create rules to inject secrets into specific containers.
3. For **Type**, select **AWS Secrets Manager**, then fill out the rest of the form, including your credentials.
4. Fill out the rest of the form, specifying how to connect to the Secrets Manager.
5. Click **Add**.

   After clicking **Add**, Prisma Cloud tries connecting to your secrets manager. If successful, the dialog closes, and an entry is added to the table. Otherwise, connection errors are displayed directly in the configuration dialog.

   Next, inject a secret into a container.

### AWS Systems Manager Parameters Store

You can integrate Prisma Cloud with AWS Systems Manager Parameters Store. First configure Prisma Cloud to access the Parameters Store, then create rules to inject the relevant secrets into the relevant containers.

**Prerequisites:**

- The service account Prisma Cloud uses to access the Parameters Store must have the following permissions. These permissions are part of pre-existing policy named AmazonSSMReadOnlyAccess. For more information, see [Configure User Access for Systems Manager](#).
  - `ssm:Get*`
  - `ssm:List*`
- You have created a secret in your Parameters Store. Prisma Cloud supports all parameter types. Note, however, that StringList is injected "as-is". For example, if the value you specify for parameter of type StringList is `twistlock,test,value`, then the injected environment variable would look like this:

```bash
ENV_VAR=twistlock,test,value
```
STEP 1 | Open Prisma Cloud Console.

STEP 2 | Integrate Prisma Cloud with the store.

1. Go to Manage > Authentication > Secrets, and click Add store.
2. Enter a name for the store. This name is used when you create rules to inject secrets into specific containers.
3. For Type, select AWS Systems Manager Parameters Store.
4. Fill out the rest of the form, specifying how to connect to the store.
5. Click Add.

After clicking Add, Prisma Cloud tries connecting to your store. If it is successful, the dialog closes, and an entry is added to the table. Otherwise, any connection errors are displayed directly in the configuration dialog.

Next, inject a secret into a container.

Azure Key Vault

You can integrate Prisma Cloud with Azure Key Vault. First configure Prisma Cloud to access your Key Vault, then create rules to inject the relevant secrets into their associated containers.

Prerequisites: You have created a secret in Key Vault.
STEP 1 | Create an Azure servicePrincipal in your Azure AD Tenant

1. Use AZ CLI to create a servicePrincipal and obtain the json credential file.
2. Authenticate to your Azure tenant.

   $ az login

3. Create a servicePrincipal

   $ az ad sp create-for-rbac

4. Save the resulting json output.

   ```json
   {
   "appId": "xxxxxxxx-xxxx-xxxx-xxxx-x",
   "displayName": "azure-cli-2018-11-01-xx-xx-xx",
   "password": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
   "tenant": "xxxxxxxxxxxxxxxxxxxxxxxxxx"
   }
   ```

5. In the Azure Key Vault, add the servicePrincipal to the Access Policies with the following permissions:

   - secrets/get permission
   - secrets/list permission

STEP 2 | In the Prisma Cloud Console, go to Manage > Authentication > Secrets.

STEP 3 | Click Add store.

1. Enter a name for the vault. This name is used when you create rules to inject secrets into specific containers.
2. For Type, select Azure Key Vault.
3. For Address, enter https://<vault-name>.vault.azure.net. This address can be found in the Azure Key Vault’s properties in the DNS Name element.
4. In Credential, click Add new.
5. Enter a name for the credentials.
6. In Type, select Azure.
7. In Service Key, enter the JSON credentials returned from the az ad sp create-for-rbac command.
8. Click Save.
9. Click Add.

   After adding the new store, Prisma Cloud tries connecting to your vault. If it is successful, the dialog closes, and an entry is added to the table. Otherwise, any connection errors are displayed directly in the configuration dialog.

   Next, inject a secret into a container.

CyberArk Enterprise Password Vault

You can integrate Prisma Cloud with CyberArk Enterprise Password Vault. First configure Prisma Cloud to access CyberArk Enterprise Password Vault, then create rules to inject the relevant secrets into the relevant containers.

STEP 1 | In Console, go to Manage > Authentication > Secrets.

STEP 2 | Click Add store.
1. Enter a name for the vault. This name is used when you create rules to inject secrets into specific containers.
2. For Type, select CyberArk Enterprise Password Vault.
3. Fill out the rest of the form, specifying how to connect to your vault.
4. Click Add.

After clicking Add, Prisma Cloud tries connecting to your vault. If it is successful, the dialog closes, and an entry is added to the table. Otherwise, any connection errors are displayed directly in the configuration dialog.

Next, inject a secret into a container.

HashiCorp Vault

You can integrate Prisma Cloud with HashiCorp Vault. Prisma Cloud supports the K/V Secrets Engine v2 in Vault 0.10.x, and K/V Secrets Engine v1 in Vault 0.9.x and older. Prisma Cloud does not support Secrets Engine v1 in Vault 0.10.x.

First configure Prisma Cloud to access HashiCorp Vault, then create rules to inject the relevant secrets into the relevant containers.

STEP 1 | In Console, go to Manage > Authentication > Secrets.

STEP 2 | Click Add store.

1. Enter a name for the vault. This name is used when you create rules to inject secrets into specific containers.
2. For Type, select HashiCorp Vault. Choose the version that matches the version of Vault installed in your environment.
3. Fill out the rest of the form, specifying how to connect to your vault.
4. Click Add.

After clicking Add, Prisma Cloud tries connecting to your vault. If it is successful, the dialog closes, and an entry is added to the table. Otherwise, any connection errors are displayed directly in the configuration dialog.

Next, inject a secret into a container.
Inject secrets into containers

To inject secrets into your containers, first integrate Prisma Cloud with your secrets manager, and then set up rules for injecting specific secrets into specific containers.

Use the same procedure for injecting secrets in a Kubernetes cluster. Set up your rules to target specific containers, images, or labels. Make sure Kubernetes uses dockerd and Prisma Cloud is running in local socket mode.

Injecting secrets into containers

After integrating your secrets store with Prisma Cloud, specify which secrets should be injected into which containers. To do this, create the appropriate rules in Console.

Secrets can be injected as environment variables or as files.

For secrets injected as environment variables: if there is a collision between a predefined environment variable and an injected secret, the value of the environment variable will always be the value of the secret. For security reasons, secrets injected as environment variables are only exposed to the container’s main process and children of the main process.

For secrets injected as files: they can be found in /run/secrets/<SECRET_NAME>, where the contents of the file contain the secret’s value. By default, secrets can only be read by root users in the container space. If you run your containers as non-root users, configure the injection rule to make the secrets readable by all users. Prisma Cloud can set the access permissions of the injected secrets file to read-only for the ‘others’ class of users. For more information about access permissions and ‘others’, see the chmod man page.

Secrets injection currently only works with image labels, not container or host labels.

Prerequisite: You’ve already created a secret in your store or vault.

STEP 1 | In Console, go to Defend > Access > Secrets.

STEP 2 | Click Add new secrets rule.

STEP 3 | Specify a name for your rule.

STEP 4 | Specify how your secret(s) should be injected. You can choose between environment variables and files.

If you choose files, you can select how the files are injected into the container. By default, the files are readable by root users only. If your containers run as non-root users, select All Users. The All Users option makes the files readable by any user by setting read permission for the others class of users.

STEP 5 | Create a list of secrets from your store that you want to inject into your container(s). Under Add secret:

1. Specify a secret name.

When you inject secrets as environment variables, this field specifies the environment variable name.

When you inject secrets as files, this field specifies the file name.
2. Specify the store where the secret is stored. The drop-down list contains any store that you integrated with Prisma Cloud.
3. Specify the secret's path and key.
4. Click **Add Secret**. It is added to the list of secrets.
5. Repeat steps a through d for as many secrets that must be included in your rule.

**STEP 6** | Specify a filter for your rule. By default, all secrets are injected into all containers on all hosts.

**STEP 7** | Click **Add**.

**STEP 8** | Verify that your secrets are properly injected.

For example, assuming your rule targets the alpine container and secrets are injected as environment variables, run the following commands:

(Default rules target all resources in the environment. The **Containers**, **Images**, **Hosts**, and **Labels** fields are set to wildcards. If your rule is set up this way, then your secrets will be injected into the alpine container.)

```bash
$ docker run -it alpine:latest /bin/sh 
/ # printenv
```

If your secrets are injected as files, and you left **Target directory** unspecified in your rule, then your secrets are injected into `/run/secrets/`, where `<SECRET-NAME>` is the name of the injected file, as specified in your rule.

```bash
$ docker run -it alpine:latest /bin/sh 
/ # cat /run/secrets/<SECRET-NAME>
```
Injecting secrets: end-to-end example

This article presents a step-by-step guide for testing Prisma Cloud’s secret manager. You will set up HashiCorp Vault, store a secret in it, inject the secret into a running container, then validate that it can be seen from within the container.

Setting up Vault

Set up HashiCorp Vault in development mode.

**STEP 1** | Download Vault from [https://www.vaultproject.io/downloads.html](https://www.vaultproject.io/downloads.html).

**STEP 2** | Unzip the package, then copy the vault executable to a directory in your `PATH`.

**STEP 3** | Verify that vault is installed. Run the following command:

```
$ vault -help
```

**STEP 4** | Start Vault in development mode.

```
$ vault server -dev -dev-listen-address='<VAULT_HOST_IPADDR>:8200'
```

===> WARNING: Dev mode is enabled!

In this mode, Vault is completely in-memory and unsealed. Vault is configured to only have a single unseal key. The root token has already been authenticated with the CLI, so you can immediately begin using the Vault CLI.

The only step you need to take is to set the following environment variables:

```
export VAULT_ADDR='http://10.240.0.53:8200'
```

The unseal key and root token are reproduced below in case you want to seal/unseal the Vault or play with authentication.

Unseal Key: Hb0dBfYh3ieHMrmf28ohu5xh0DKfmP4aNa8JS5/jNaWQ=
Root Token: 29e3e12b-09b4-af6c-6e87-cbd9fbc51bd

Storing a secret in HashiCorp Vault

Store a secret in Vault.

**STEP 1** | Open a shell and ssh to the host running Vault.

**STEP 2** | Set the Vault address in your environment.

```
$ export VAULT_ADDR='http://<VAULT_HOST_IPADDR>:8200'
```

**STEP 3** | Create a secret.
For Vault 0.10 or later:

```bash
vault kv put secret/mySecret1 "pass=1234567"
```

For Vault 0.9.x or older:

```
$ vault write secret/mySecret1 "pass=1234567"
```

**STEP 4 |** Read the secret back to validate it was properly stored.

For Vault 0.10 or later:

```
$ vault kv get secret/mySecret1
```

For Vault 0.9.x or older:

```
$ vault read secret/mySecret1
```

**Integrating Prisma Cloud and Vault**

Follow the steps in [Integrating Prisma Cloud with HashiCorp Vault](#).

**Creating a rule in Console**

Follow the steps in [Injecting secrets into containers](#).

**Validating the secret is injected**

Start a container and verify that your secret is properly injected.

+ The same procedure can be followed for injecting secrets in Kubernetes cluster. Make sure Kubernetes uses dockerd and Prisma Cloud runs in local socket mode.

**Prerequisites:** Defender must be running on the machine where you start your container.

**STEP 1 |** Start a container:

```
$ docker run -ti ubuntu /bin/bash
```

**STEP 2 |** Validate your secrets have been injected into this container.

If you injected your secrets as environment variables, run:

```
# printenv
```

If you injected your secrets as files, run:

```
# ls /run/secrets
# cat /run/secrets/<SECRET_NAME>
```

**STEP 3 |** Exit the shell inside the container.

```
# exit
```
STEP 4 | If your secrets are injected as environment variables, validate that they are encrypted when you run docker inspect.

Start a container, and run it in the background:

```bash
$ docker run -dit ubuntu /bin/bash <CONTAINER_ID>
```

```bash
$ docker inspect <CONTAINER_ID>
```
Alerts

Prisma Cloud lets you surface critical policy breaches by sending alerts to any number of channels. Alerts ensure that significant events are put in front of the right audience at the right time.

- Alert mechanism
- AWS Security Hub
- Cortex XSOAR alerts
- Email alerts
- Google Cloud Pub/Sub
- Google Cloud Security Command Center
- IBM Cloud Security Advisor
- JIRA Alerts
- PagerDuty alerts
- ServiceNow alerts
- Slack Alerts
- Webhook alerts
Alert mechanism

Prisma Cloud lets you surface critical policy breaches by sending alerts to any number of channels. Alerts ensure that significant events are put in front of the right audience at the right time.

Alerts are built on the following constructs:

- **Alert profile** --
  Specifies which events should be sent to which channel. You can create any number of alert profiles, where each profile gives you granular control over which audience should receive which notifications.

- **Alert channel** --
  Messaging medium over which alerts are sent. Prisma Cloud supports email, JIRA, Slack, PagerDuty, and others.

- **Alert trigger** --
  Events that require further scrutiny. Alerts are raised when the rules that make up your policy are violated. When something in your environment violates a rule, an audit is logged, and an alert is sent to any matching alert profile (channel, audience). Prisma Cloud can be configured to notify the appropriate party when an entire policy, or even a specific rule, is violated.

You can also set up alerts for Defender health events. These events tell you when Defender unexpectedly disconnects from Console. Alerts are sent when a Defender has been disconnected for more than 6 hours.

Not all triggers are available for all channels. For example, new JIRA issues can only be opened when vulnerability rules are triggered.

Triggers

Most alerts trigger on a policy violation. When policy is the trigger, you can optionally choose to trigger on specific rules rather than the entire policy. Vulnerability, compliance, and cloud discovery alerts work differently.

_Vulnerability alerts that arise from registry scans only trigger for the 50 most recent images, as sorted by last modified date. The limit is designed to contain Console resource consumption in large environments._

Vulnerability alerts

The number of known vulnerabilities in a resource is not static over time. As the Prisma Cloud Intelligence Stream is updated with new data, new vulnerabilities might be uncovered in resources that were previously considered clean. The first time a resource (image, container, host, etc) enters the environment, Prisma Cloud assesses it for vulnerabilities. If a vulnerability violates a rule in the policy, and the rule has been configured to trigger an alert, an alert is dispatched. Thereafter, every resource is periodically rescanned. Additional alerts are dispatched only when new vulnerabilities that match your alert profile settings are detected. With vulnerability alerts, you get one, and only one, alert for each vulnerability detected (aggregated by scan).

Compliance alerts

Alerts for compliance issues work a little differently. The resources in your system are either compliant or non-compliant. When your system is non-compliant, Prisma Cloud sends an alert. As long as there are non-compliant resources, Prisma Cloud sends an alert every scan interval (default 24 hours). Compliance alerts list each failed check, and the number of resources that failed the check in the latest scan and the previous
scan. For detailed information about exactly which resources are non-compliant, use Compliance Explorer. The following screenshot shows an example compliance email alert:

<table>
<thead>
<tr>
<th>Container compliance</th>
<th>Previous</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>52 Verify SELinux security options, if applicable</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>53 Restrict Linux kernel capabilities within containers</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>51 Verify AppArmor profile, if applicable</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>54 Do not use privileged containers</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image compliance</th>
<th>Previous</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>421 Image contains banned processes</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>41 Image should be created with a user</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>422 Image contains malware</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>406 Add HEALTHCHECK instruction to the container image</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>420 Image is not updated to latest</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host compliance</th>
<th>Previous</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 Use the updated Linux Kernel</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>15 Keep Docker up to date</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>16 Only allow trusted users to control Docker daemon</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>11 Create a separate partition for containers</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

For example:

- **Scan period 1**: You have non-compliant container named *crusty_pigeon*. You’ll be alerted about the container compliance issues.
- **Scan period 2**: Container *crusty_pigeon* is still running. It’s still non-compliant. You’ll be alerted about the same container compliance issues.

### Cloud discovery alerts

Cloud discovery alerts warn you when new cloud native resources are discovered in your environment so that you can inspect and secure them with Prisma Cloud. Cloud discovery alerts are available on the email channel only. For each new resource discovered in a scan, Prisma Cloud lists the cloud provider, region, project, service type (i.e. AWS Lambda, Azure AKS) and resource name (my-aks-cluster).

### Limitations

For runtime audits, there’s a limit of 50 runtime audits per aggregation period (seconds, minutes, hours, days) for all alert providers.
AWS Security Hub

AWS Security Hub aggregates, organizes, and prioritizes security alerts from multiple AWS services and AWS Partner Network solutions, including Prisma Cloud, to give you a comprehensive view of security across your environment.

Permissions

The minimum required permissions policy to integrate Prisma Cloud with AWS Security Hub is AWSSecurityHubFullAccess. Whether using IAM users, groups, or roles, be sure the entity Prisma Cloud uses to access AWS Security Hub has this minimum permissions policy.

This procedure shows you how to set up integration with an IAM user (configured as a service account). In AWS IAM, create a service account that has the AWSSecurityHubFullAccess permissions policy. You will need the service account’s access key ID and secret access key to integrate with Prisma Cloud.

Enabling AWS Security Hub

STEP 1 | Log into your AWS tenant and enter Security Hub in the Find services search, then select Security Hub.

STEP 2 | Click Enable Security Hub.

Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to Manage > Alerts.

STEP 2 | In Aggregate audits every, specify the maximum rate that alerts should be sent.

You can specify Second, Minute, Hour, Day.

Sending alerts to Security Hub

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:
(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, along with the type of issue, priority, assignee, and so on.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.

Create new alert profile

Create a new alert profile.

STEP 1 | In Manage > Alerts, click Add profile.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In Provider, select AWS Security Hub.

Configure the channel

Configure the channel.

STEP 1 | In Region, select your region.

STEP 2 | Enter your Account ID, which can be found in the AWS Management Console under My Account > Account Settings.
STEP 3 | Provide your credentials, which Prisma Cloud uses to integrate with AWS Security Hub. You can either:

- Set Use IAM Role to On to use the role assigned to the EC2 instance that runs Console.
- Select a Credential, which holds the access key and secret for a service account. Select an existing credential or create a new one.

STEP 4 | (Optional) Toggle the Use AWS STS to On. In Role ARN, specify the role to use. For more information about AWS STS, see AWS Security Token Service.

STEP 5 | Click Send Test Alert to test the connection. An alert is sent immediately.

Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under Alert Types, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click Edit.

STEP 3 | To specify specific rules that should trigger an alert, deselect All rules, and then select any individual rules.

Alert types

- Access
- Cloud Native App Firewall
- Cloud Native Network Firewall
- Container and Image vulnerabilities
- Container Runtime

Alert on

- All rules

Select rules to be alerted on

- Default - alert on suspicious runtime behavior

- Defender Health
- Host App Firewall
- Host Runtime
- Host vulnerabilities
- Incident
- Kubernetes Audits
- RASP App Firewall
- RASP Runtime
- Serverless

STEP 4 | Click Save.
Cortex XSOAR alerts

Cortex XSOAR is a security orchestration, automation, and response (SOAR) platform. Prisma Cloud can send alerts, vulnerabilities, and compliance issues to XSOAR when your policies are violated. Prisma Cloud can be configured to send data when an entire policy, or even specific rules, are violated.

Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

**STEP 1** | Open Console, and go to Manage > Alerts.

**STEP 2** | In Aggregate audits every, specify the maximum rate that alerts should be sent. You can specify Second, Minute, Hour, Day.

Alert providers

- Not applicable to vulnerability, compliance and cloud discovery alerts.

| Audit aggregation period | Second | Minute | Hour | Day |

Send alerts to XSOAR

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

1. **Alert settings** — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, along with the type of issue, priority, assignee, and so on.

2. **Alert triggers** — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.

If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

**STEP 1** | In Manage > Alerts, click Add profile.

**STEP 2** | Enter a name for your alert profile.
Configure the channel

In **Provider**, select **Demisto**.

**Configure the channel.**

**STEP 1 |** In **Console Name**, choose the console name XSOAR should use to access your Prisma Cloud console.

**STEP 2 |** Copy the **Console URL** and save it for creating the integration in XSOAR.

**STEP 3 |** Copy the **Project Name**, and save it for creating the integration in XSOAR (the project name would only appear if you are using the projects feature).

**STEP 4 |** Copy the **CA certificate** and save it for creating the integration in XSOAR.

Configure the triggers

Configure how the alert is triggered.

**STEP 1 |** Under **Alert Types**, check the boxes types of events that should trigger an alert.

**STEP 2 |** For additional configuration options, click **Edit**.

**STEP 3 |** To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.

**Alert types**

- Access
- Cloud Native App Firewall
- Cloud Native Network Firewall
- Container and Image vulnerabilities
- Container Runtime **Edit**

Alert on

- All rules

Select rules to be alerted on

- Default - alert on suspicious runtime behavior
- Defender Health
- Host App Firewall
- Host Runtime
- Host vulnerabilities
- Incident
- Kubernetes Audits
- RASP App Firewall
- RASP Runtime
- Serverless

**STEP 4 |** Click **Save**.
Configure XSOAR

Create a new Prisma Cloud Compute integration in XSOAR.

**STEP 1** | Log into Cortex XSOAR.

**STEP 2** | Go to **Settings > Integrations**.

**STEP 3** | Search for **Prisma Cloud Compute** and click **Add instance**.

**STEP 4** | Under the **Settings**:

1. **Name**: Enter the name for the integration.
2. Check the **Fetch incidents** checkbox.
3. **Prisma Cloud Compute Console URL and Port**: Paste the URL of the console that you copied from Prisma Cloud.

4. (optional) **Prisma Cloud Compute Project Name**: Enter the name of the project in Prisma Cloud.

5. **Credentials**: Enter the Prisma Cloud username that XSOAR should use to communicate with your Prisma Cloud console.

6. **Password**: Enter the password for the username you provided.

7. **Prisma Cloud Compute CA Certificate**: Paste the CA Certificate you copied from Prisma Cloud, or enter your own CA Certificate (if using a custom certificate to access your Prisma Cloud console).

**STEP 5** | Click **Test** to check the connection to Prisma Cloud console.

**STEP 6** | Click **Done** to save the integration.

**STEP 7** | Go to **Incidents** to see the alerts received from Prisma Cloud.
Email alerts

Prisma Cloud can send email alerts when your policies are violated. Audits in Monitor > Events are the result of a policy violation. Prisma Cloud can be configured to notify the appropriate party by email when an entire policy, or even specific rules, are violated.

Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

**STEP 1** | Open Console, and go to Manage > Alerts.

**STEP 2** | In Aggregate audits every, specify the maximum rate that alerts should be sent.

You can specify Second, Minute, Hour, Day.

Sending email alerts

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

1. **Alert settings — Who should get the alerts, and on what channel?** Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, along with the type of issue, priority, assignee, and so on.

2. **Alert triggers — Which events should trigger an alert to be sent?** Specify which of the rules that make up your overall policy should trigger alerts.
If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

**STEP 1** | In Manage > Alerts, click Add profile.

**STEP 2** | Enter a name for your alert profile.

**STEP 3** | In Provider, select Email.

Configure the channel

Configure the channel.

**STEP 1** | In SMTP address, specify the hostname for your outgoing email server.

**STEP 2** | In Port, specify the port for email submissions.

**STEP 3** | In Credential, create the credentials required to access the email account that sends alerts. This isn't a required field.
   1. Click Add new.
   2. Select Basic authentication.
   3. Enter a username and password.
STEP 4 | If you’re using SMTPS (your SMTP connection is secured by SSL), set SSL to On.

STEP 5 | Set up your recipients.
   1. Click Add recipient, and enter an email address. Every email alert profile must have at least one recipient, even if you’re using alert labels.
   2. (Optional) Specify recipients using alert labels.

STEP 6 | Click Send Test Alert to test the connection to your SMTP server.

Configure the triggers
Configure how the alert is triggered.

STEP 1 | Under Alert Types, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click Edit.

STEP 3 | To specify specific rules that should trigger an alert, deselect All rules, and then select any individual rules.

Alert types
- Access
- Cloud Native App Firewall
- Cloud Native Network Firewall
- Container and Image vulnerabilities
- Container Runtime

Alert on
- All rules
- Default - alert on suspicious runtime behavior

- Defender Health
- Host App Firewall
- Host Runtime
- Host vulnerabilities
- Incident
- Kubernetes Audits
- RASP App Firewall
- RASP Runtime
- Serverless

STEP 4 | Click Save.
Google Cloud Pub/Sub

Google Cloud Pub/Sub is a durable, scalable event ingestion and delivery system. It provides asynchronous messaging that decouples senders from receivers, and enables highly available communication between independently written applications.

Prisma Cloud can send alerts to Google Cloud Pub/Sub topics, where a topic is a message feed.

Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

**STEP 1** | Open Console, and go to Manage > Alerts.

**STEP 2** | In Aggregate audits every, specify the maximum rate that alerts should be sent.

You can specify Second, Minute, Hour, Day.

Sending alerts to Google Cloud Pub/Sub

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

1. **Alert settings — Who should get the alerts, and on what channel?** Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, along with the type of issue, priority, assignee, and so on.

2. **Alert triggers — Which events should trigger an alert to be sent?** Specify which of the rules that make up your overall policy should trigger alerts.
If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

Prerequisite: You’ve set up a Cloud Pub/Sub topic.

STEP 1 | In Manage > Alerts, click Add profile.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In Provider, select GCP Pub/Sub.

Configure the channel

Configure the channel.

STEP 1 | In Credential, click Add new or select an existing service account.

To create a new GCP credential, see here.

STEP 2 | Enter a Cloud Pub/Sub topic.

STEP 3 | Click Send Test Alert to test the connection.

Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under Alert Types, check the boxes types of events that should trigger an alert.
STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.

```plaintext
Alert types

- [ ] Access
- [ ] Cloud Native App Firewall
- [ ] Cloud Native Network Firewall
- [ ] Container and Image vulnerabilities
- [x] Container Runtime

Alert on
- [ ] All rules

**Select rules to be alerted on**
- [ ] Default - alert on suspicious runtime behavior

- [ ] Defender Health
- [ ] Host App Firewall
- [ ] Host Runtime
- [ ] Host vulnerabilities
- [ ] Incident
- [ ] Kubernetes Audits
- [ ] RASP App Firewall
- [ ] RASP Runtime
- [ ] Serverless

STEP 4 | Click **Save**.
Google Cloud Security Command Center

Prisma Cloud can be configured as a security source that provides security findings to Google Cloud Security Command Center (SCC). This lets you see all security tool findings in a single place.

Prisma Cloud is a registered Google Cloud Platform Marketplace partner.

Configuring Google Cloud Security Command Center

In Google Cloud Platform (GCP), create a service account in your project that has the Cloud Security Command Center API enabled. You will need the service account keys, API, and Organization ID to enable this feature.

You should have already enabled and onboarded Prisma Cloud as a Security Source in Google Security Command Center. Prisma Cloud supports the alpha and beta versions of Google Security Command Center. The following instructions show how to configure the beta version.

STEP 1 | Log into your GCP tenant and select the project that has the Cloud Security Command Center API enabled.

STEP 2 | Go to IAM & admin > Service accounts.

STEP 3 | Click Create Service Account.

STEP 4 | Enter a name and description for the service account.
STEP 5 | Grant this service account access to project (optional) click continue. Do not grant a role to the account at this time.

STEP 6 | Grant user account to this service account click create key.

STEP 7 | Set key type to JSON, and click create. Save the downloaded JSON key.

STEP 8 | Go to the project's APIs & Services > Credentials.

STEP 9 | Click Create credentials > API key.
STEP 10 | Save the API key. We recommended that you restrict the key to the Cloud Security Command Center API.

STEP 11 | Go to the Google tenant's organizational IAM & admin.

   This setting is configured at the organizational level, not the project level.

STEP 12 | In the IAM window click +Add.
STEP 13 | Paste in the name of the service account that has been created.

STEP 14 | Select Role: Security Center > Security Center Editor.

Enter one or more members below. Then select a role for these members to grant them access to your resources. Multiple roles allowed. Learn more

```
New members
twistlock-gcss@com
```

### Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

**STEP 1 |** Open Console, and go to Manage > Alerts.

**STEP 2 |** In Aggregate audits every, specify the maximum rate that alerts should be sent.

You can specify Second, Minute, Hour, Day.
Alert providers

Not applicable to vulnerability, compliance and cloud discovery alerts.

Audit aggregation period  

Sending alerts to Google Cloud SCC

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

1. **Alert settings — Who should get the alerts, and on what channel?** Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, along with the type of issue, priority, assignee, and so on.

2. **Alert triggers — Which events should trigger an alert to be sent?** Specify which of the rules that make up your overall policy should trigger alerts.

If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.
STEP 1 | In Manage > Alerts, click Add profile.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In Provider, select Security Center.

Configure the channel

Configure the channel.

STEP 1 | In Credential, click Add new or select an existing service account.

To create a new GCP credential, see here.

STEP 2 | In Source Name, enter the resource path for a source that’s already been created.

The source name has the following format:

organizations/<organization_id>/sources/<source_id>

Where organization_id and source_id are numeric identifiers. For example:

organizations/111122222444/sources/43211234

STEP 3 | Click Send Test Alert to test the connection.

Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under Alert Types, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click Edit.

STEP 3 | To specify specific rules that should trigger an alert, deselect All rules, and then select any individual rules.
Alert types

- Access
- Cloud Native App Firewall
- Cloud Native Network Firewall
- Container and Image vulnerabilities
- Container Runtime

Select rules to be alerted on
- Default - alert on suspicious runtime behavior

STEP 4 | Click Save.
IBM Cloud Security Advisor

IBM Cloud Security Advisor is a centralized security dashboard. Prisma Cloud can be configured to send security findings to your service dashboard.

Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

**STEP 1** | Open Console, and go to **Manage > Alerts**.

**STEP 2** | In **Aggregate audits every**, specify the maximum rate that alerts should be sent. You can specify **Second, Minute, Hour, Day**.

Alert providers

- Not applicable to vulnerability, compliance and cloud discovery alerts.

<table>
<thead>
<tr>
<th>Audit aggregation period</th>
<th>Second</th>
<th>Minute</th>
<th>Hour</th>
<th>Day</th>
</tr>
</thead>
</table>

Sending alerts to Security Advisor

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) **Alert settings — Who should get the alerts, and on what channel?** Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, along with the type of issue, priority, assignee, and so on.

(2) **Alert triggers — Which events should trigger an alert to be sent?** Specify which of the rules that make up your overall policy should trigger alerts.
If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

**STEP 1** | In Manage > Alerts, click Add profile.

**STEP 2** | Enter a name for your alert profile.

**STEP 3** | In Provider, select IBM Cloud Security Advisor.

Configure the channel

Configure the channel.

**STEP 1** | In Credential, click Add new or select an existing service account.

To create a new IBM Cloud credential, see here.

**STEP 2** | Copy the configuration URL, and set it aside. You'll need it for the next step.

**STEP 3** | Go to the Security Advisor dashboard, and then follow the steps in Prisma Cloud partner integration to complete the setup process.

**STEP 4** | Click Send Test Alert to test the connection.

Configure the triggers

Configure how the alert is triggered.
STEP 1 | Under Alert Types, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click Edit.

STEP 3 | To specify specific rules that should trigger an alert, deselect All rules, and then select any individual rules.

Alert types

- Access
- Cloud Native App Firewall
- Cloud Native Network Firewall
- Container and Image vulnerabilities
- Container Runtime

Alert on
- All rules

Select rules to be alerted on
- Default - alert on suspicious runtime behavior

- Defender Health
- Host App Firewall
- Host Runtime
- Host vulnerabilities
- Incident
- Kubernetes Audits
- RASP App Firewall
- RASP Runtime
- Serverless

STEP 4 | Click Save.
JIRA Alerts

Prisma Cloud continually scans your environment for vulnerabilities using the threat data in the Intelligence Stream. Prisma Cloud can open JIRA issues when new vulnerabilities are detected in your environment. This mechanism lets you implement continuous vulnerability assessment and remediation by hooking directly into the developer's workflow.

New JIRA issues are opened when new vulnerabilities are found. Issues are opened on a per-image basis. Each JIRA issue lists the new vulnerabilities discovered, and a list of vulnerabilities that have already been reported but were still detected.

JIRA issues are opened based on policy. For example, an issue would be created when all of the following conditions are met:

- You have a rule that alerts on critical vulnerabilities,
- The rule is associated with your JIRA alert profile,
- The Prisma Cloud scanner finds a critical vulnerability in an image in your environment.

The following screenshot shows an example JIRA issue opened by Prisma Cloud.
Intelligent issue routing

You can leverage image labels to intelligently route alerts to the right team, and eliminate manual ticket triage. For example, if team-a is responsible for image-a, and a vulnerability is found in image-a, you could set up the alert to flow directly to team-a's JIRA queue.

Intelligent routing depends on a Prisma Cloud feature called alert labels, where you define labels that Prisma Cloud should watch. When rules trigger, Prisma Cloud extracts the value of the label from the resource, and applies it to the next phase of alert processing. For JIRA alerts, you can use labels to specify the JIRA project key, JIRA labels, and JIRA issue assignee.
For example, if you have an image with the following labels:

group=front-end-group
team=client-team
business-app=my-business-app

You could configure Prisma Cloud to open issues about this specific image in the JIRA project defined by the *group* label.

**Configuring alert frequency**

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

**STEP 1** | Open Console, and go to Manage > Alerts.

**STEP 2** | In Aggregate audits every, specify the maximum rate that alerts should be sent.

You can specify Second, Minute, Hour, Day.

**Alert providers**

* Not applicable to vulnerability, compliance and cloud discovery alerts.

Audit aggregation period  

| Second | Minute | Hour | Day |

**Integrating Prisma Cloud with JIRA**

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

1. **Alert settings — Who should get the alerts, and on what channel?** Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, along with the type of issue, priority, assignee, and so on.

2. **Alert triggers — Which events should trigger an alert to be sent?** Specify which of the rules that make up your overall policy should trigger alerts.
If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

STEP 1 | In Manage > Alerts, click Add profile.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In Provider, select JIRA.

Configure the channel

Configure the channel.

STEP 1 | In Base URL, specify the location of your JIRA service.

STEP 2 | In Credential, create the credentials required to access the account.
   1. Click Add new.
   2. Select Basic authentication.
   3. Enter a username and password.
   4. Click Save.

STEP 3 | In CA certificate, enter a copy of the CA certificate in PEM format.
STEP 4 | In **Project key**, enter a project key.

Alternatively, you can dynamically specify the project key based on a label. When an alert fires, the project key is taken from the label of the resource that triggered the action. To do so, click **Select labels**,... and choose a label that you know will contain the project key. If there are no labels in the drop-down list, go to **Manage > Alerts > Alert Labels**, and define them.

STEP 5 | Enter an issue type.

STEP 6 | Enter a priority.

STEP 7 | Enter a comma delimited list of JIRA labels to apply to the issue.

You can dynamically define the list from a label. Click **Select labels**,... and select one or more labels.

STEP 8 | Enter an assignee for the new issue.

You can dynamically define the assignee from a label. Click **Select labels**,... and select one or more labels.

STEP 9 | Click **Send Test Alert** to test the connection.

Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.

**Alert types for JIRA**

- Container and Image vulnerabilities
- Host vulnerabilities

**Select rules to be alerted on**

- Default - ignore Twistlock components
- Default - alert all components

STEP 4 | Click **Save**.
PagerDuty alerts

You can configure Prisma Cloud to route alerts to PagerDuty. When Prisma Cloud detects anomalies, it generates alerts. Alerts are raised when the rules that make up your policy are violated.

Configuring PagerDuty

Create a new Prisma Cloud service, and get an integration key.

STEP 1 | Log into PagerDuty.

STEP 2 | Go to Configuration > Services.

STEP 3 | Click New Service.

STEP 4 | Under General Settings:
   1. Name: Enter Prisma Cloud.

STEP 5 | Under Integration Settings:
   1. Integration Type: Select Use our API directly, the select Events API v2.
   2. Integration Name: Enter Prisma Cloud.
Add a Service

A service may represent an application, component or team you wish to open incidents against.

General Settings

- Name: Twistlock
- Description: Add a description for this service (optional)

Integration Settings

Integrations can open and resolve incidents. Once a service is created, it can have multiple integrations.

- Integration Type
  - Select a tool: We integrate with dozens of monitoring systems. This may involve configuration steps in your monitoring tool.
  - Integrate via email: If your monitoring tool can send email, it can integrate with PagerDuty using a custom email address.
  - Use our API directly: If you’re writing your own integration, use our Events API. More information is in our developer documentation.
  - Don’t use an integration: If you only want incidents to be manually created. You can always add additional integrations later.

Integration Name: Twistlock

STEP 6 | **Click Add Service.** You’re taken to **Integrations** tab for the Prisma Cloud service.

STEP 7 | **Copy the Integration Key**, and set it aside. You’ll use it to configure the integration in Prisma Cloud Console.

<table>
<thead>
<tr>
<th>Name</th>
<th>Integration Key</th>
<th>Type</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twistlock</td>
<td>d157f241b7b545089ccad05b38ae7b91</td>
<td>Events API v2</td>
<td></td>
</tr>
</tbody>
</table>

Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each
alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

**STEP 1** | Open Console, and go to Manage > Alerts.

**STEP 2** | In Aggregate audits every, specify the maximum rate that alerts should be sent.

You can specify Second, Minute, Hour, Day.

### Sending alerts to PagerDuty

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

1. **Alert settings — Who should get the alerts, and on what channel?** Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, along with the type of issue, priority, assignee, and so on.

2. **Alert triggers — Which events should trigger an alert to be sent?** Specify which of the rules that make up your overall policy should trigger alerts.
If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

**STEP 1** | In Manage > Alerts, click Add profile.

**STEP 2** | Enter a name for your alert profile.

**STEP 3** | In Provider, select PagerDuty.

Configure the channel

Configure the channel.

**STEP 1** | In Routing Key, enter the integration key you copied from PagerDuty.

**STEP 2** | In Summary, enter a brief description, which will appear in the PagerDuty dashboard alongside your alerts.

**STEP 3** | For Severity, select the urgency of the alert.

**STEP 4** | Click Send Test Alert to validate the integration.

If the integration is set up properly, you will see a sample alert in PagerDuty. In the PagerDuty dashboard, click Alerts.

Configure the triggers

Configure how the alert is triggered.
STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.

STEP 4 | Click **Save**.
ServiceNow alerts

ServiceNow is a workflow management platform. It offers a number of security operations applications. You can configure Prisma Cloud to route alerts to ServiceNow’s Security Incident Response application.

Prisma Cloud audits are mapped to a ServiceNow security incident as follows:
- Audits and incidents are mapped to individual ServiceNow security incidents.
- Vulnerabilities are aggregated by resource (currently image) and mapped to individual ServiceNow security incidents. ServiceNow short description fields list the resource. ServiceNow long description fields list the details of each finding.

<table>
<thead>
<tr>
<th>ServiceNow security incident</th>
<th>Field description</th>
<th>Prisma Cloud audit data</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>The current state of the security incident. Upon security incident creation, this field defaults to Draft.</td>
<td>Draft (automatically set by ServiceNow)</td>
</tr>
<tr>
<td>Priority</td>
<td>Select the order in which to address this security incident, based on the urgency. If this value is changed after the record is saved, it can affect the Business impact calculation.</td>
<td>Vulnerabilities: Max severity from the image’s new vulnerabilities. ServiceNow’s priorities map one-to-one to Prisma Cloud priorities (Critical - Critical, High - High, Medium - Medium, Low - Low). Incidents and audits: runtime audits priority set in the alert profile.</td>
</tr>
<tr>
<td>Business impact</td>
<td>Select the importance of this security incident to your business. The default value is Non-critical. If, after the security incident record has been saved, you change the value in the Priority and/or Risk fields, the Business impact is recalculated.</td>
<td>Automatically calculated by ServiceNow</td>
</tr>
<tr>
<td>Assignment group</td>
<td>The group to which this security incident is assigned.</td>
<td>Assignment group set in the alert profile</td>
</tr>
<tr>
<td>Assigned to</td>
<td>The individual assigned to analyze this security incident.</td>
<td>Assignee set in the alert profile</td>
</tr>
<tr>
<td>Short description</td>
<td>A brief description of the security incident.</td>
<td>Vulnerabilities: Prisma Cloud Compute vulnerabilities for image &lt;image name&gt; Compliance: Prisma Cloud Compute compliance issues for image/container/host &lt;image/container/host name&gt; Incidents and audits: Prisma</td>
</tr>
</tbody>
</table>
### Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

**STEP 1** | Open Console, and go to Manage > Alerts.

**STEP 2** | In Aggregate audits every, specify the maximum rate that alerts should be sent. You can specify **Second, Minute, Hour, Day**.

---

<table>
<thead>
<tr>
<th>ServiceNow security incident</th>
<th>Field description</th>
<th>Prisma Cloud audit data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Cloud Compute audit - &lt;audit type&gt;</td>
</tr>
<tr>
<td>Category</td>
<td>Set to &quot;None&quot;</td>
<td></td>
</tr>
<tr>
<td>Sub-category</td>
<td>Set to &quot;None&quot;</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
<td>Vulnerabilities:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CVEs IDs list (with each CVE's details)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Collections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compliance:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Summary of compliance issues count by severity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Compliance issues list (with each issue's details)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Collections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Incidents and audits:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Description: &lt;audit description&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Related resource: &lt;resource name&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Project: &lt;Project&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Time created: &lt;audit creation date&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Then add all the other fields this type of Incident/Audit has</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note that the Project field will specify Central Console even when projects aren't enabled.</td>
</tr>
</tbody>
</table>
Alert providers

Alert profiles specify which events trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

1. **Alert settings — Who should get the alerts, and on what channel?** Configure Prisma Cloud to integrate with ServiceNow and specify the people or places where alerts should be sent. You can specify assignees and assignment groups.

2. **Alert triggers — Which events should trigger an alert to be sent?** Specify which of the rules that make up your overall policy should trigger alerts.

Sending findings to ServiceNow
Create new alert profile

Create a new alert profile.

**STEP 1** | In **Manage > Alerts**, click **Add profile**.

**STEP 2** | Enter a name for your alert profile.

**STEP 3** | In **Provider**, select **ServiceNow**.

Configure the channel

Configure Prisma Cloud to send alerts to ServiceNow, then validate the setup by sending a test alert.

Prerequisites: You’ve created a service account in ServiceNow with a base role of web_service_admin.

**STEP 1** | In **Application**, select **Security Incident Response**.
STEP 2 | In **URL**, specify the base URL of your ServiceNow tenant.
   
   For example, `https://ena03291.service-now.com`

STEP 3 | In **Credential**, click **Add New**.
   
   1. In **Type**, select **Basic authentication**.
   
      This is currently the only auth method supported.
   
   2. Enter a username and password.

STEP 4 | (Optional) In **Assignee**, enter the name of a user in ServiceNow that will be assigned the security incident.
   
   This value isn't case sensitive.

STEP 5 | (Mandatory) In **Assignment Group**, enter the name of a group in ServiceNow that will be assigned the security incident. The default value is **Security Incident Assignment**.
   
   If **Assignment Group** is set without specifying **Assignee**, the first user from the group is set on the security incident (ServiceNow's logic).
   
   If the **Assignee** set in the profile isn't a part of the **Assignment Group**, the security incident won't be created (ServiceNow's logic).

STEP 6 | (Optional) In **CA certificate**, enter a CA certificate in PEM format. Relevant only for on-premises deployments of ServiceNow.

STEP 7 | Click **Send Test Alert**. If everything looks good, and you get an alert in ServiceNow, save the profile.

### Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.
Alert types

☐ Access
☐ Cloud Native App Firewall
☐ Cloud Native Network Firewall
☐ Container and Image vulnerabilities
☐ Container Runtime

Alert on
☐ All rules

Select rules to be alerted on
☐ Default - alert on suspicious runtime behavior

☐ Defender Health
☐ Host App Firewall
☐ Host Runtime
☐ Host vulnerabilities
☐ Incident
☐ Kubernetes Audits
☐ RASP App Firewall
☐ RASP Runtime
☐ Serverless

STEP 4 | Click Save.
ServiceNow alerts

ServiceNow is a workflow management platform. It offers a number of security operations applications. You can configure Prisma Cloud to route alerts to ServiceNow’s Vulnerability Response application.

To integrate Prisma Cloud with ServiceNow, you’ll need to create a ServiceNow endpoint to consume findings from the Prisma Cloud scanner. The endpoint is created using ServiceNow’s Scripted REST API mechanism.

Each vulnerability found by the Prisma Cloud scanner is mapped to a ServiceNow vulnerable item. Scanner data is mapped to vulnerable items as follows:

> Vulnerable items contain all CVEs reported by the Prisma Cloud scanner only if the corresponding CVEs also exist in ServiceNow’s vuln DB. If a CVE doesn’t exist in ServiceNow, the Vulnerability (Reference) field won’t list it.

<table>
<thead>
<tr>
<th>ServiceNow vulnerability item field</th>
<th>Field description</th>
<th>Prisma Cloud scanner data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>The scanner that found this vulnerable item.</td>
<td>Prisma Cloud Compute</td>
</tr>
<tr>
<td>Vulnerability (Reference)</td>
<td>ID of the vulnerability associated with this vulnerable item.</td>
<td>Reference to CVE ID (if exists in ServiceNow’s vulnerabilities DB)</td>
</tr>
<tr>
<td>State</td>
<td>This field defaults to Open, but you can change it to Under Investigation if the vulnerability is ready for immediate remediation.</td>
<td>Open (automatically set by ServiceNow)</td>
</tr>
<tr>
<td>Assignment group</td>
<td>Group selected to work on this vulnerability group.</td>
<td>Assignment group set in the alert profile</td>
</tr>
<tr>
<td>Assigned to</td>
<td>Individual from the selected assignment group that works on this vulnerability.</td>
<td>Assignee set in the alert profile</td>
</tr>
<tr>
<td>Created</td>
<td>The date this vulnerable item was created in your instance.</td>
<td>Creation date of the vulnerable item (automatically set by ServiceNow)</td>
</tr>
<tr>
<td>Additional comments</td>
<td>Any relevant information.</td>
<td>Vulnerabilities:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Image name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Severity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Package</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Package version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Fix status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Collections</td>
</tr>
</tbody>
</table>
Configuring ServiceNow

Create a ServiceNow endpoint to collect findings from the Prisma Cloud scanner.

**Prerequisites:** Prisma Cloud Console is running.

**STEP 1 |** In ServiceNow, create a Scripted REST API. Name it **Prisma Vulnerabilities Report**. For more information, see the official documentation [here](#).

**STEP 2 |** Create a new resource in your scripted REST service.

**API definition = Prisma Vulnerabilities Report**

<table>
<thead>
<tr>
<th>Request Headers</th>
<th>Query Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources</td>
<td>Name ▼ Search</td>
</tr>
</tbody>
</table>

**STEP 3 |** In **Name**, enter **report_findings**.

**STEP 4 |** In **HTTP method**, select **POST**.

**STEP 5 |** Download the script that implements the endpoint from Prisma Cloud Console.

1. Log into Prisma Cloud Console.
2. Go to **Manage > Alerts > Add Profile**.
3. Click **Add Profile**.
4. In **Provider**, select **ServiceNow**.
5. In **Application**, select **Vulnerability Response**.
6. In **Scripted REST API**, click **Copy**.
7. In ServiceNow, paste the script into **Script**.

**STEP 6 |** Click **Submit** to create the resource.

**STEP 7 |** Construct the URL for your resource (endpoint), then copy it, and set it aside. You’ll need when you configure Prisma Cloud to send findings to ServiceNow.

The format for the base URL is: `https://<SERVICENOW>/<BASE_API_PATH>`

For example: `https://ena03291.service-now.com/api/paan/prisma_vulnerabilities_report`

Where:

- **SERVICENOW** — URL for your ServiceNow instance.
- **BASE_API_PATH** — Path to the scripted API service you just created.
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

**STEP 1** | Open Console, and go to **Manage > Alerts**.

**STEP 2** | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.
You can specify **Second, Minute, Hour, Day**.

Sending findings to ServiceNow

Alert profiles specify which events trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) **Alert settings — Who should get the alerts, and on what channel?** Configure Prisma Cloud to integrate with ServiceNow and specify the people or places where alerts should be sent. You can specify assignees and assignment groups.

(2) **Alert triggers — Which events should trigger an alert to be sent?** Specify which of the rules that make up your overall policy should trigger alerts. For the Vulnerability Response application, you can send vulnerability and compliance alerts only.
Create new alert profile

Create a new alert profile.

**STEP 1** | In Manage > Alerts, click Add profile.

**STEP 2** | Enter a name for your alert profile.

**STEP 3** | In Provider, select ServiceNow.

Configure the channel

Configure Prisma Cloud to send alerts to ServiceNow, then validate the setup by sending a test alert.

**Prerequisites:** You’ve created a service account in ServiceNow with a base role of web_service_admin.

**STEP 1** | In Application, select Vulnerability Response.
STEP 2 | In **Scripted API URL**, enter the url of the vulnerabilities reporting api defined in ServiceNow (see ServiceNow config above). e.g. https://ven03718.service-now.com/api/paan/prisma_vulnerabilities_report

STEP 3 | In **Credential**, click **Add New**.
   1. In **Type**, select **Basic authentication**.
      This is currently the only auth method supported.
   2. Enter a username and password.

STEP 4 | (Optional) In **Assignee**, enter the name of a user in ServiceNow that will be assigned the Vulnerable Items.
   The assignee name isn’t case-sensitive.

STEP 5 | (Optional) In **Assignment Group**, enter the name of a group in ServiceNow that will be assigned the Vulnerable Items.

STEP 6 | (Optional) In **CA certificate**, enter a CA certificate in PEM format. Relevant only for on-premises deployments of ServiceNow.

STEP 7 | Click **Send Test Alert**. If everything looks good, and you get an alert in ServiceNow, save the profile.

Configure the triggers
Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.

### Alert triggers

- **Container and Image** [Edit]
- **Vulnerabilities**

- **Alert on**
  - [ ] All rules

- **Select rules to be alerted on**
  - [ ] Default - ignore Twistlock components
  - [ ] Default - alert all components

STEP 4 | Click **Save**.

---

© 2020 Palo Alto Networks, Inc.
Slack Alerts

Prisma Cloud lets you send alerts to Slack channels and users.

Configuring Slack

To integrate Prisma Cloud with Slack, you must enable incoming webhooks. Prisma Cloud uses incoming webhooks to post messages to Slack.

STEP 1 | Log into the page where you manage apps for your Slack workspace.

STEP 2 | In the Search App Directory box, enter Incoming Webhooks, and hit Return.

STEP 3 | Click on the result.

STEP 4 | Click the green Add Configuration button.

STEP 5 | Enter the channel where you want Prisma Cloud to post.

STEP 6 | Click Add Incoming Webooks Integration.

STEP 7 | Copy the Webhook URL and set it aside. You will use it when configuring Prisma Cloud.

Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to Manage > Alerts.

STEP 2 | In Aggregate audits every, specify the maximum rate that alerts should be sent.
   You can specify Second, Minute, Hour, Day.

Alert providers

Not applicable to vulnerability, compliance and cloud discovery alerts.

Audit aggregation period

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Second</td>
<td>Minute</td>
<td>Hour</td>
</tr>
</tbody>
</table>

Sending alerts to Slack

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example,
configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, along with the type of issue, priority, assignee, and so on.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.

Create new alert profile

Create a new alert profile.

**STEP 1** | In Manage > Alerts, click Add profile.

**STEP 2** | Enter a name for your alert profile.

**STEP 3** | In Provider, select Slack.

Configure the channel

Configure the channel.

**STEP 1** | In Incoming Webhook URL, enter the URL you generated in the previous section.

**STEP 2** | Specify how to route alerts. Enter values for one or both of the following fields.

1. In Channels, enter the Slack channel where you want to post alerts.
2. In Users, enter the Slack users to whom you want to send alerts.

**STEP 3** | Click Send Test Alert to test the connection.
Configure the triggers

Configure how the alert is triggered.

**STEP 1** | Under **Alert Types**, check the boxes types of events that should trigger an alert.

**STEP 2** | For additional configuration options, click **Edit**.

**STEP 3** | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.

**Alert types**

- [ ] Access
- [ ] Cloud Native App Firewall
- [ ] Cloud Native Network Firewall
- [ ] Container and Image vulnerabilities
- [X] Container Runtime  
  
  **Alert on**
  - [ ] All rules

  **Select rules to be alerted on**
  - [ ] Default - alert on suspicious runtime behavior

- [ ] Defender Health
- [ ] Host App Firewall
- [ ] Host Runtime
- [ ] Host vulnerabilities
- [ ] Incident
- [ ] Kubernetes Audits
- [ ] RASP App Firewall
- [ ] RASP Runtime
- [ ] Serverless

**STEP 4** | Click **Save**.
Webhook alerts

Prisma Cloud offers native integration with a number of services, including email, JIRA, and Slack. When no native integration is available, webhooks provide a mechanism to interface Prisma Cloud’s alert system with virtually any third party service.

A webhook is an HTTP callback. When an event occurs, Prisma Cloud notifies your web service with an HTTP POST request. The request contains an JSON body that you configure when you set up the webhook. A webhook configuration consists of:

- URL,
- Custom JSON body,
- Username,
- Password,
- CA Certificate.

Custom JSON body

You can customize the body of the POST request with values of interest. The content of the JSON object in the request body is defined using predefined macros. For example:

```
{
    "type": "ContainerRuntime",
    "host": "host1",
    "details": "/bin/cp changed binary /bin/busybox MD5:XXXXXXXX"
}
```

When an event occurs, Prisma Cloud replaces the macros in your custom JSON with real values, and then submits the request.

```
{
    "type": "ContainerRuntime",
    "host": "host1",
    "details": "/bin/cp changed binary /bin/busybox MD5:XXXXXXXX"
}
```

All supported macros are described in the following table. Not all macros are applicable to all alert types.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#type</td>
<td>Audit alert type. For example, 'Container Runtime'.</td>
</tr>
<tr>
<td>#time</td>
<td>Audit alert time. For example, 'Jan 21, 2018 UTC'.</td>
</tr>
<tr>
<td>#container</td>
<td>Impacted container.</td>
</tr>
<tr>
<td>#image</td>
<td>Impacted image.</td>
</tr>
<tr>
<td>#host</td>
<td>Hostname for the host where the audit occurred.</td>
</tr>
<tr>
<td>#fqdn</td>
<td>Fully qualified domain name for the host where the audit occurred.</td>
</tr>
<tr>
<td>#function</td>
<td>Serverless function where the audit occurred.</td>
</tr>
<tr>
<td>Rule</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>#region</td>
<td>Region where the audit occurred. For example 'N. Virginia'.</td>
</tr>
<tr>
<td>#runtime</td>
<td>Language runtime in which the audit occurred. For example, 'python3.6'.</td>
</tr>
<tr>
<td>#appID</td>
<td>Serverless or Function name.</td>
</tr>
<tr>
<td>#rule</td>
<td>Rule which triggered the alert.</td>
</tr>
<tr>
<td>#message</td>
<td>Associated alert message.</td>
</tr>
<tr>
<td>#aggregated</td>
<td>All fields in the audit message as a single JSON object.</td>
</tr>
<tr>
<td>#rest</td>
<td>All subsequent alerts that occurred during the aggregation period, in JSON format.</td>
</tr>
</tbody>
</table>

### Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

**STEP 1 |** Open Console, and go to Manage > Alerts.

**STEP 2 |** In Aggregate audits every, specify the maximum rate that alerts should be sent.

You can specify Second, Minute, Hour, Day.

### Alert providers

Not applicable to vulnerability, compliance and cloud discovery alerts.

### Sending alerts to a webhook

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

1. **Alert settings — Who should get the alerts, and on what channel?** Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, along with the type of issue, priority, assignee, and so on.

2. **Alert triggers — Which events should trigger an alert to be sent?** Specify which of the rules that make up your overall policy should trigger alerts.
If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert channel

Create a new alert channel.

**Prerequisites:** You have a service to accept Prisma Cloud's callback. For purely testing purposes, consider PostBin or RequestBin.

**STEP 1 |** In Manage > Alerts, click *Add profile*.

**STEP 2 |** Enter a name for your alert profile.

**STEP 3 |** In Provider, select *Webhook*.

Configure the channel

Configure the channel.

**STEP 1 |** In *Webhook incoming URL*, enter the endpoint where Prisma Cloud should submit the alert.

**STEP 2 |** In *Custom JSON*, Enter the structure of the JSON payload that your web application is expecting.

For more details about the type of data in each field, click *Show macros*.
STEP 3 | (Optional) In **Credential**, specify a basic auth credential if your endpoint requires authentication.

STEP 4 | (Optional) In **CA Certificate**, enter a CA cert in PEM format.

> *When using a CA cert to secure communication, only one-way SSL authentication is supported. If two-way SSL authentication is configured, alerts will not be sent.*

STEP 5 | Click **Send Test Alert** to test the connection. An alert is sent immediately.

**Configure the triggers**

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.

**Alert types**

- Access
- Cloud Native App Firewall
- Cloud Native Network Firewall
- Container and Image vulnerabilities
- Container Runtime
- Defender Health
- Host App Firewall
- Host Runtime
- Host vulnerabilities
- Incident
- Kubernetes Audits
- RASP App Firewall
- RASP Runtime
- Serverless

STEP 4 | Click **Save**.
Prisma Cloud creates and stores audit event records (audits) for all major subsystems. Audits can be reviewed in Monitor > Events, or they can be retrieved from the Prisma Cloud API. If you have a centralized syslog collector, you can integrate Prisma Cloud with your existing infrastructure by configuring Prisma Cloud to send all audit events to syslog in RFC5424-compliant format.

- Event viewer
- Host activity
- Administrative activity audit trail
- Annotate audit event records
- Delete audit logs
- Syslog and stdout integration
- Log rotation
- Throttling audits
- Prometheus
- Kubernetes auditing
Event viewer

Prisma Cloud creates and stores audit event records (audits) for all major subsystems. Audits can be reviewed in Monitor > Events, or they can be retrieved from the Prisma Cloud API. If you have a centralized syslog collector, you can integrate Prisma Cloud with your existing infrastructure by configuring Prisma Cloud to send all audit events to syslog in RFC5424-compliant format.

When you’re reviewing audits in a dialog, the list of audits isn’t updated in real-time. To retrieve all the latest data, close the dialog. If the Refresh button is decorated with a red indicator, click it to refresh the view with the latest data, then reopen the dialog.

Access audits

Access to any container resource protected by Defender is logged and aggregated in Console. You can also configure Prisma Cloud to record audits for sudo, SSH, and other events that are executed on hosts protected by Defender. This audit trail links access to system components to individual users. Access events can be viewed in Console under Monitor > Events.

Runtime audits

Prisma Cloud records an audit every time a runtime sensor (process, network, file system, and system call) detects activity that deviates from the sum of the predictive model plus any runtime rules you’ve defined. For example, a file system audit event is emitted when Prisma Cloud detects malware in a container. Runtime events for containers can be viewed in Console under Monitor > Events. Runtime events for hosts can be viewed in Console under Monitor > Events.

Firewall audits

Web Application and API Security (WAAS) is a layer 7 filtering engine that ensures only safe, clean traffic ever reaches your web app. Audits are generated when WAAS detects an attack, such as SQL injection or cross-site scripting. WAAS audits can be viewed under Monitor > Events.

Admin activity

All Prisma Cloud administrative activity can viewed under Manage > View Logs.

Prisma Cloud limits viewing of audit trails to those with a job-related need. To view audit events, you must log into Console. Only users with Administrator, Operator, Defender Manager, or Auditor roles can view audit data in Console. Similarly, only users with the above-mentioned roles can retrieve audit data from the Prisma Cloud API.
Host activity

Prisma Cloud lets you audit security-related activity on hosts protected by Defender.

Runtime rules specify the type of activity to capture. The default host runtime rule, *Default - alert on suspicious runtime behavior*, assesses interactive user activity. You can create additional runtime rules to control which type of events are captured on which hosts.

The following types of activity can be assessed and captured.

- **Docker** — Docker commands that alter state: create, run, exec, commit, save, push, login, export, kill, start, stop, and tag.
- **Read-only Docker events** — When you configure Prisma Cloud to capture Docker commands, you can optionally capture commands that simply read state. These include `docker ps` and `docker images`.
- **New sessions spawned by sshd** — Self-explanatory.
- **Commands run with sudo or su** — Self-explanatory.
- **Log activity from background apps** — Processes run by services on the host that could raise security concerns. Activities include: service restart, service install, service modified, cron modified, system update, system reboot, package source modified, package source added, iptables changed, secret modified, accounts modified, and sensitive files modified.

Whereas Defender’s runtime system surfaces suspect activity by sifting through events, Defender’s *forensics* system presents a raw list of all spawned processes.

Enabling audits for local events

To enable audits for host activity, create a new host runtime rule. After making your changes, you can view all audits in *Monitor > Events* with the *Host Activities* filter.

Auditing begins after a rule is created. Any events that occurred before the rule was created are not recorded.

**STEP 1** | Open Console.

**STEP 2** | Go to *Defend > Runtime > Host Policy*.

**STEP 3** | Click *Add rule*, and give it a name.

**STEP 4** | In *Hosts*, specify the hosts for which this rule applies.

**STEP 5** | In the *Activities* tab, enable the events for which you want audits.

**STEP 6** | Click *Save*. 
Administrative activity audit trail

All Prisma Cloud administrative activities are logged.

Changes to any settings (including previous and new values), changes to any rules (create, modify, or delete), and all logon activity (success and failure) are logged. For every event, both the user name and source IP are captured.

For login activity, the following events are captured:

- Every login attempt from the login page, including failures.
- Every failed attempt to authenticate to the API. Successfully authenticated calls to the API are not recorded.

The full set of log data is available to anyone with a user role of auditor or higher.

To view the administrative history, open Console, then go to Manage > View Logs > History.

Settings and rule events show how a configuration has changed. Both the API endpoint, and a diff of the previous and current JSON objects are shown. The following screenshot shows the changes to a vulnerability rule:

```
Device:
  envVers: [],
  readonly: false,
  vulnerabilities: [
  {   
      "block": true,
      "id": 46,
      "modified": "2017-12-24T14:54:54.054Z",
      "name": "No critical vulns",
      "namespace": [],
      "owner": "ian",
      "previousName": "...
    },

    "modified": "2017-12-24T14:54:58.058Z",
    "minSeverity": 9,
    "minSeverity": 7
  }
},

"effect": "block",
"group": [

"a"
],

"modified": "2017-12-24T14:54:29.037Z",
"modified": "2017-12-24T14:54:57.410Z",
"minSeverity": 9,
"namespace": [],
"owner": "ian",
"previousName": "...
```

Use the API reference to determine what has changed and how to interpret the meaning of the change. The /api/v1/policies/cve endpoint creates and modifies vulnerability rules. An id of 46 specifies how to handle vulnerabilities in OS packages. And minSeverity specifies the threshold for taking the action specified in block. In this case, user ian has changed the threshold for blocking containers with vulnerable OS packages from 9 (Critical) to 7 (High).
Annotate audit event records

Prisma Cloud lets you surface and display designated labels in events and reports. For example, you might already use labels to classify resources according to team name or cost center. With alert labels, you can specify which of these key-value pairs are appended to events (audits, syslog, alerts) and reports.

Labels are key-value string pairs that can be attached to objects such as images, containers, or pods. In Console, specify a list of labels that contain the metadata you want to append to Prisma Cloud events. When an event fires, if the associated object has any of the specified labels, they are appended to the event.

Specifying labels to append to Prisma Cloud events

Specify which labels to append to Prisma Cloud events.

STEP 1 | Open Console.
STEP 2 | Go to Manage > Alerts > Alert Labels.
STEP 3 | Click Add label.
STEP 4 | Enter the name of the label to be appended to Prisma Cloud events.
STEP 5 | Click Save.

Email alerts

The contents of a label can be used as a dynamic target for email alerts. Specify the labels that contain a comma delimited list of email addresses, and when an event fires, the recipients will be notified.

Before setting up your email alerts, be sure you've specified a list of labels to be appended to Prisma Cloud events, where at least one label contains a comma-delimited list of email addresses.

Configure email alerts

JIRA alerts

The contents of a label can be used to dynamically specify project keys, JIRA labels, and assignees for new JIRA issues.

Before setting up your JIRA alerts, be sure you've specified a list of labels to be appended to Prisma Cloud events, where the labels contain the type of information you need to dynamically route JIRA issues to the right team.

Configure JIRA alerts
Delete audit logs

Delete audits from the log using the Prisma Cloud API.

Delete all access audit events

Deleting audit log entries is done through API calls only.

Path

DELETE /api/v1/audits/access?type=[type]

Description

Deletes all access events of a specific type. In case type is not provided all access audits for every type will be removed. The possible 'types' for this command are:

- *docker*: Docker access audit
- *kubernetes*: Kubernetes access audit (to Kubernetes master)
- *swarm*: same as Kubernetes but for Docker Swarm
- *sshd*: SSH audit to host
- *sudo*: sudo commands audit on host

Status codes

- **200** - no error
- **400** - bad request was provided

Example request

```
curl -X DELETE -u admin:<Password> 'https://<localhost>:8443/api/v1/audits/access?type=docker'
```

Example response

```
{}
```

Delete access audit event

Deleting audit log entries is done through API calls only.

Path

DELETE /api/v1/audits/access/[id]

Description

Deletes an access event with specific id.

Status codes

- **200** - no error
- **400** - bad request was provided
Example request

```bash
curl -X DELETE -u admin:<Password>  'https://<localhost>:8443/api/v1/audits/access/580fd342b8aaba1000ec47be'
```

Example response

```
{}
```

*The current set up enables user to delete entries at the access layer for each runtime sensor. To learn more on API calls, see the [API reference](#).*
Syslog and stdout integration

You can configure Prisma Cloud to send audit event records (audits) to syslog and/or stdout.

Syslog integration must be turned on manually. Open Console, go to Manage > Alerts > Logging, then set Syslog to Enabled. Prisma Cloud connects to the syslog socket on /dev/log. Stdout integration can be enabled from the same tab.

When you enable syslog or stdout integration, you can optionally enable verbose output. Verbose output records vulnerability and compliance issues in your environment. It also records all process activity.

In general, enabling verbose output is not recommended because of the substantial overhead. You can retrieve this data much more efficiently from the Prisma Cloud API. Nevertheless, sometimes this capability is expressly required for integration with SIEM tools.

Do not enable both syslog and stdout on hosts with systemd. With systemd, anything sent to stdout gets logged to syslog. With both syslog and stdout enabled, you would get duplicate messages in syslog.

Sending syslog messages to a network endpoint

Writing to /dev/log sends logs to the local host’s syslog daemon. The syslog daemon can then be optionally configured to forward those logs to a remote syslog or SIEM server. If you don’t have access to the underlying host, you can configure Prisma Cloud Console to send log messages directly to your remote system.

In most cases, you won’t need to specify a network endpoint in order to send syslog messages to your SIEM tool. If you already have log collectors on your hosts, simply enable syslog. Your log collectors will stream Prisma Cloud syslog messages to your SIEM tool.

Some things to keep in mind:

- Console sends logs directly to your remote server. When configuring Console with the remote server, validate that the address you enter is actually reachable from the host where Console runs. Otherwise, you risk losing log messages.
- Because Console sends messages directly to your remote server, and not through the local syslog daemon, you don’t get some of syslog’s built-in benefits, such as buffering, which protects against network outages and service failures.
- The classic syslog implementation sends logs over UDP. This is considered a bad practice if your logs have any value. UDP is connectionless. Packets are sent to their destination without confirming that they were received. TCP’s stateful connections and retransmission capabilities make it more appropriate for shuttling logs to a SIEM.

STEP 1 | Log into Console.

STEP 2 | Go to Manage > Alerts > Logging.

STEP 3 | Set Syslog to Enabled.

STEP 4 | In Send syslog messages over the network to, click Edit, and then specify a destination.
Appending custom strings to syslog messages

You can configure Prisma Cloud Compute to append a custom string to all Console and Defender syslog messages.

Custom strings are set in the event message as a key-value pair, where the key is "id", and the value is your custom string. The following screenshot shows a Defender event, where the custom string is "koko".

Configuring a custom string is useful when you have multiple Prisma Cloud Compute deployments (i.e. multiple Compute Consoles) and you’re aggregating all messages in a single log management system. The custom string serves as a marker that lets you correlate specific events to specific deployments.

**STEP 1** | Open Console.

**STEP 2** | Go to Manage > Alerts > Logging.

**STEP 3** | Set Syslog to Enabled.

**STEP 4** | For Identifier, click Edit, and enter a string.

**Events**

Both Console and Defender emit messages. The data emitted to syslog and stdout is exactly the same.

**Console events**

Console syslog messages are tagged as Twistlock-Console in the logs.

**Image scan**

Records when Prisma Cloud scans an image.

Example image scan message:

```
Jul 30 18:51:32 aqsa-root Twistlock-Console[1]:
  time="2019-07-30T18:51:32.214136319Z"
  type="scan_summary"
  log_type="image"

image_id="sha256:cd14cecfdb3a657ba7d05bea026e7ac8b9abafc6e5c66253ab327c7211fa6281"
image_name="aqsa/internal:tag5"
vulnerabilities="297"
compliance="1"
```

**Container scan**

Records when Prisma Cloud scans a container.

Example container scan message:

```
Jul 30 22:06:15 aqsa-root Twistlock-Console[1]:
  time="2019-07-30T22:06:15.804842461Z"
  type="container_scan"
  log_type="container"
```
**Host scan**

Records when Prisma Cloud scans a host. Defenders scan the hosts they run on.

Example host scan:

```
Jul 30 22:09:53 aqsa-root Twistlock-Console[1]:
time="2019-07-30T22:09:53.390680962Z"
type="scan_summary"
log_type="host"
hostname="aqsa-root.c.cto-sandbox.internal"
vulnerabilities="89"
compliance="17"
```

**Code repository scan**

Records when Prisma Cloud scans a code repository.

Example scan:

```
Jul 7 23:34:09 ip-172-31-55-106 Twistlock-Console[1]:
time="2020-07-07T23:34:09.25109843Z"
type="scan_summary"
last_update_time="2020-07-07 23:21:00.203 +0000 UTC"
log_type="code_repository_scan"
source="github"
repository_name="jerryso/apper"
vulnerable_files="1"
vulnerabilities="25"
collections="All"
```

**Individual compliance issues**

Records a compliance finding. These messages are tagged with `log_type="compliance"`, and are generated as a byproduct of container scans, image scans, host scans, and registry scans.

Compliance issues are only recorded when **Detailed output for vulnerabilities and compliance** is enabled in **Manage > Alerts > Logging** (to see this option, syslog must be enabled).

A syslog entry is generated for each compliance issue. This can result in a significant amount of data, which is why verbose output is disabled by default.

You must have a rule that alerts on compliance issues for an entry to be written to syslog. It might just be the **Default - alert all components** rule, or another custom rule. This option does not simply log all compliance issues irrespective of the rules that are in place.

Example image compliance issue:

```
Jul 30 22:18:53 aqsa-root Twistlock-Console[1]:
time="2019-07-30T22:18:53.23838464Z"
type="image_scan"
log_type="containerCompliance"
compliance_id="41"
```
**Example container compliance issue:**

| type="container_scan" | log_type="containerCompliance" |
| compliance_id="526" | severity="medium" |
| description="(CIS_Docker_CE_v1.1.0 - 5.26) Check container health at runtime" | rule="Default - alert on critical and high" |
| host="aqsa-root.c.cto-sandbox.internal" | container_id="22b745b2220f3f128a1cf57d2fffd3f28a02ba380930ebf83fca9f26d4d2b8aa4" |
| container_name="serene_cray" |

**Example host compliance issue:**

| type="host_scan" | log_type="containerCompliance" |
| compliance_id="6518" | severity="high" |
| description="(CIS_Linux_1.1.0 - 5.1.8) Ensure at/cron is restricted to authorized users" | rule="Default - alert on critical and high" |
| host="aqsa-root.c.cto-sandbox.internal" |

**Individual vulnerability issues**

Records a vulnerability finding. These messages are tagged with `log_type="vulnerability"`, and are generated as a byproduct of image scans, host scans, and registry scans.

Vulnerability issues are only recorded when Detailed output for vulnerabilities and compliance is enabled in Manage > Alerts > Logging.

A syslog entry is generated for each vulnerability for each package. This can result in a significant amount of data, which is why verbose output is disabled by default.

For example, consider a rule that raises an alert when vulnerabilities of medium severity or higher are found in an image. If there are eleven packages that violate this rule, there will be eleven syslog entries, one for each package.

You must have a rule that alerts on vulnerabilities for an entry to be written to syslog. It might just be the Default - alert all components rule, or another custom rule. This option does not simply log all vulnerability data irrespective of the rules that are in place.

**Example image vulnerability issue:**

Example registry image vulnerability issue:

```
Jul 30 22:03:56 aqsa-root Twistlock-Console[1]:
time="2019-07-30T22:03:56.930640366Z"
type="registry_scan"
log_type="vulnerability"
vulnerability_id="410"
description="Image contains vulnerable Python components"
cve="CVE-2019-11236"
severity="medium"
package="urllib3"
package_version="1.24.1"
vendor_status="fixed in 1.24.3"
rule="test"
host="aqsa-root.c.cto-sandbox.internal"
image_id="sha256:196601f91030425db810fa57104b041e414b9b963923ad574e74700c3ea8213e"
image_name="weaveworksdemos/user-db:0.4.0"
```

Example host vulnerability issue:

```
Jul 30 22:09:53 aqsa-root Twistlock-Console[1]:
time="2019-07-30T22:09:53.390181271Z"
type="host_scan"
log_type="vulnerability"
vulnerability_id="46"
description="Image contains vulnerable OS packages"
cve="CVE-2017-8845"
severity="low"
package="lzo2"
package_version="2.08-1.2"
vendor_status="deferred"
rule="Default - alert all components" host="aqsa-root.c.cto-sandbox.internal"
```

Admin activity

Changes to any settings (including previous and new values), changes to any rules (create, modify, or delete), and all logon activity (success and failure) are logged. For every event, both the user name and source IP are captured.

Example admin activity audit:

```
Jul 30 21:58:16 aqsa-root Twistlock-Console[1]:
```
Defender events

Defender syslog messages are tagged as **Twistlock-Defender** in logs.

Container runtime audit

Activity that breaches your runtime rules or the automatically generated allow lists in your models generates audits. The *log_type* field specifies the runtime sensor that detected the anomaly (filesystem, processes, syscalls, or network).

Example container runtime audit: The following process audit shows that busybox was unexpectedly launched, and an alert was raised.

```
Jul 30 22:41:25 aqsa-root Twistlock-Defender[13460]:
type="container_runtime_audit"

container_id="73c2e8267f9b80ea152403c36c377476d24e43e211bb098300a317b3d1c472e4"
container_name="/dreamy_rosalind"
image_id="sha256:94e814e2efa8845d95b2112d54497fbad173e45121ce9255b93401392f538499"
image_name="ubuntu:18.04"
effect="alert"
msg="High rate of reg file access events, reporting aggregation started;
last event: /usr/lib/apt/methods/gpgv wrote a suspicious file to /tmp/apt.conf.2ZH7tP.
Command: /usr/lib/apt/methods/gpgv"
log_type="filesystem"
```

Host runtime audit

Activity that breaches your runtime rules or the automatically generated allow lists in your host services models generates audits.

Example host runtime audit:

```
Jul 30 22:47:12 aqsa-root Twistlock-Defender[13460]:
type="host_runtime_audit"
service_name="ssh"
effect="alert"
msg="Outbound connection by /usr/lib/apt/methods/http to an unexpected port: 80 IP: 91.189.91.26. Low severity audit, event is automatically added to the runtime model"
log_type="network"
```

Access audit

Docker commands run on hosts protected by Defender.

With user access events, you can determine who performed an action, and on which resource.
For example:
- [Bruce] [started container X] in the [DEV environment] (allowed).
- [Bruce] [stopped container Y] in the [PROD environment] (denied).

All Docker commands issued to the Docker daemon are intercepted and inspected by Defender to determine if they comply with the policy set in Console.

The following diagram illustrates how Defender operates on the management plane:

1. Bruce, a developer, issues a command, docker -H.
2. Defender checks the command against the policies defined in the Console. If the command is allowed, Defender forwards it to the Docker daemon for execution. If the command is denied, the user is notified.
3. An event is recorded in syslog.

Access audits have the following fields:
- type=access_audit
- user=[String] Identity of the person who ran the command
- action=[String] Docker command requested - API invoked
- action_type=[String] Action type
- allow=[Boolean] true/false - Action was allowed or not.
- rule=[String] Rule matched

Example:

```
Jul 30 23:02:23 aqsa-root Twistlock-Defender[13460]:
time="2019-07-30T23:02:23.179494498Z"
type="access_audit"
user="aqsa"
action="docker_ping"
action_type="docker"
allow="true"
rule="Default - allow all"
```

**App firewall audit (WAAS)**

All events associated with WAAS (Web-Application and API Security) rules for container, hosts and app-embedded generate audits.
WAAS serverless events are not registered in the syslog. Events audits will be registered to the syslog in future releases.

WAAS Container and Host rule audits are written to the Defender host’s syslog. WAAS App-Embedded rule audits are written to the console’s host’s syslog.

Message fields for WAAS audit would change based on the deployment type as follows:

- container_id=[String] Container id in which the event triggered
- container_name=[String] Container name on which the action was performed
- image_name=[String] Image name on which the action was performed
- custom_labels=[String] User-defined Alert Labels (Manage > Alerts > Alert Labels)
- hostname=[String] Host in which the event triggered
- app_id=[String] app_id in which the event triggered
- time=[String] Request timestamp
- type=[String] Type of app_firewall_audit
- effect=[String] “alert”, “prevent”, “ban”
- msg=[String] Audit message detailing the event
- log_type=[String] Attack Type
- source_ip=[String] Source IP address from the request originated
- source_country=[String] Country associated with source IP address
- connecting_ips=[CSV] List of IPs included in the X-Forwarded-For header
- request_method=[String] HTTP Request Method
- request_user_agents=[String] User-agent string parsed from the User-Agent header
- request_host=[String] HTTP hostname in the request
- request_url=[String] Request url
- request_path=[String] Request path
- request_query=[String] Request query string
- request_header_names=[String] Ordered list of HTTP request headers
- response_header_names=[String] Ordered list of HTTP response headers
- status_code=[String] HTTP response status code in the server response

In addition, message structure is subject for the following changes:

- Fields containing empty values are omitted from the message i.e. if a HTTP message does not contain a query field the request_query field will not be present in the message.

Example:

```
Jul 16 20:10:16 cnaf-nightly-build Twistlock-Defender[1947]:
time="2020-07-16T20:10:16.706085135Z"
type="container_app_firewall_audit"

container_id="0a16b4e4dbefc6ef8cc6a08d038e775a8523ad053416730f01eafbf2dee2e693"
container_name="/nginx"
```
Process activity audit

Records all processes spawned in a container.

Process audits are only recorded when Detailed output of all runtime process activity is enabled in Manage > Alerts > Logging.

Note that process activity that breaches your runtime policy is separately audited. For more information, see the container runtime audit section.

This audit has the following fields:

- type=process
- pid=Process ID
- path=Path to the executable in the container file system
- md5=MD5 checksum for the executable file
- interactive=Whether the process was spawned from a shell session: true or false
- container-id=Container ID

Example: This audit shows that busybox was spawned in the container with ID 8c5b3fe0037d.

```
Jul 30 22:06:03 aqsa-root Twistlock-Defender[13460]:
time="2019-07-30T22:06:03.515319204Z"
type="process"
pid="20859"
path="/bin/df"
md5="aeec137ba6012f093e483efb57a521ab"
interactive=false
container_id="3491b03544a51c60e176e54a5077161f14db3850bf069cf7a096db028e9981de"
```

Incidents

Incidents are logical groupings of events, related by context, that reveal known attack patterns.

Example container incident:

```
Jul 30 22:41:24 aqsa-root Twistlock-Defender[13460]:
type="incident"
```
Example host incident:

Mar  5 00:26:42 itay-ThinkPad-P50 Twistlock-Defender[22797]:
  time="2018-03-05T00:26:42.894707831+02:00"
  type="incident"
  service_name="http-service"
  host="itay-ThinkPad-P50"
  incident_category="serviceViolation"
  audit_ids="5a9c72a223d020590de74db5"

Rate limiters

Depending on your configuration, Prisma Cloud can produce a lot of logs, especially in environments with many hosts, images, and containers. By default, most syslog daemons throttle logging with a rate limiter.

If you have a large environment (hundreds of Defenders with tens of images per host) AND you have configured Prisma Cloud for verbose syslog output, you will need to tune the rate limiter. Otherwise, you might find that logs are missing.

For example, on RHEL 7, you must tune both systemd-journald’s `RateLimitInterval` and `RateLimitBurst` settings and rsyslog’s `imjournalRatelimitInterval` and `imjournalRatelimitBurst` settings. For more information about RedHat settings, see How to disable log rate-limiting in Red Hat Enterprise Linux 7.

Truncated log messages

Very long syslog events can get truncated. For example, changing settings in Console generates management_audits events, which show a diff between old settings and new settings. For policies changes, the diff can be big. Linux log managers limit the number of characters logged per line, and so long messages, such as management audits, can be truncated.

If you've got truncated log messages, increase the log manager’s default string size limit. There are several types log managers, but rsyslog is popular with most distributions. For rsyslog, the default log string size is 1024 characters per line. To increase it, open `/etc/rsyslog.conf` and set the maximum message size:

```
$MaxMessageSize 20k
```
Log rotation

Both Console and Defender call `log-rotate` every 30 minutes. The options passed to log-rotate are described below.

**Defender**
The default path for Defender’s log file is `/var/lib/twistlock/log/defender.log`.

It is configured as follows:

- Truncate the original log file in place after creating a copy, instead of moving the old log file. *(copytruncate)*
- Have 10 backup files rotated. If rotation exceeds 10 files, the oldest rotated file is deleted. *(rotate 10)*
- Don’t generate an error in case a log file doesn’t exist. *(missingok)*
- Don’t rotate the log in case it’s empty. *(notifempty)*
- Rotate the log only if its size is 100M or more. *(size 100M)*
- Compress the rotated logs. *(compress)*

**Console**
The default path for Console’s log file is `/var/lib/twistlock/log/console.log`.

It is configured as follows:

- Truncate the original log file in place after creating a copy, instead of moving the old log file. *(copytruncate)*
- Have 10 backup files rotated. If rotation exceeds 10 files, the oldest rotated file is deleted. *(rotate 10)*
- Don’t generate an error in case a log file doesn’t exist. *(missingok)*
- Don’t rotate the log in case it’s empty. *(notifempty)*
- Rotate the log only if its size is 100M or more. *(size 100M)*
- Compress the rotated logs. *(compress)*

**DB logs**
We log CRITICAL/ERROR messages to enable critical DB diagnostics.

*This is automatically done by Prisma Cloud and is non-configurable.*
Throttling audits

When your runtime models aren't completely tuned, you can get a barrage of false positives. It's difficult for operators to parse through so many audits, especially when most of it is noise. And the volume and rate of audits can degrade your system.

To address the problem, Console presents a cross-section of the most important audits, while dropping redundant audits. Prisma Cloud collects, collates, and throttles audits on a per-profile (model) basis, with a maximum of 100 audits per profile, sorted by recency. Every audit is categorized by Type and Attack Type, where a Type can have one or more Attack Types. For example, the Network Type has the following Attack Types (not a complete list):

<table>
<thead>
<tr>
<th>Type</th>
<th>Attack Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>Feed DNS</td>
<td>DNS query of a high risk domain based on data in the Intelligence Stream.</td>
</tr>
<tr>
<td>Network</td>
<td>Unexpected Listening Port</td>
<td>Container process is listening on an unexpected port.</td>
</tr>
<tr>
<td>Network</td>
<td>etc.</td>
<td>etc.</td>
</tr>
</tbody>
</table>

When there's a large number of incoming audits, Prisma Cloud temporarily applies throttling. When more than five audits of the same Attack Type are received over a short period of time, those audits are dropped. A running count of all audits (dropped and not dropped) is updated periodically. If no audits are received after a grace period, throttling is disabled. Throttling is reset every 24 hours. That is, if throttling is applied for all day 0, and five audits of a given attack have already been received, then no new audits for that Attack Type are displayed for 24 hours. At the 24 hour period mark, throttling is disabled, and any new audits are collected, collated, and presented, until throttling is reapplied.

Throttling is applied to audits in the following systems:

- Monitor > Events > Container Audits
- Monitor > Events > Host Audits
- Monitor > Events > Cloud Native App Firewall
- Monitor > Events > WAAS for Hosts

Note that a comprehensive list of audits can always be found in the Defender logs. If syslog and/or stdout integration is enabled, all audits are always emitted there too. Finally, if you set up alerts on all container runtime rules, you'll get all audits to your alert channel; nothing is dropped or throttled.

Finally, if audits are being throttled, it's a symptom of a larger issue. You should tune your runtime models.
Prometheus

Prometheus is a monitoring platform that collects metrics from targets by scraping their published endpoints. Prisma Cloud can be configured to be a Prometheus target.

You can use Prometheus to monitor time series data across your environment and show high-level, dashboard-like, stats to visualize trends and changes. Prisma Cloud’s instrumentation lets you track metrics such as the total number of connected Defenders and the total number of container images in your environment being protected by Defender.

Metrics

Metrics are a core Prometheus concept. Instrumented systems expose metrics. Prometheus stores the metrics in its time-series database, and makes them easily available to query to understand how systems behave over time.

Prisma Cloud has two types of metrics:

- Counters: Single monotonically increasing values. A counter’s value can only increase or be reset to zero.
- Gauges: Single numerical values that can arbitrarily go up or down.

**Prisma Cloud metrics**

All Prisma Cloud metrics are listed in the following table. Vulnerability and compliance metrics are updated every 24 hours. The rest of the metrics are updated every 10 minutes.

Note that *_vulnerabilities and *_compliance metrics report how many entities (images, containers, hosts, etc) are at risk by the highest severity issue that impacts them. In other words, images_critical_vulnerabilities is not a total count of critical vulnerabilities in the images in your environment. Rather, it is a total count of images where the highest severity CVE is critical. For a thorough explanation of how this type of metric is used, see Vulnerability Explorer.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalDefenders</td>
<td>Gauge</td>
<td>Total number of Defenders connected to Console. Connected and disconnected Defenders can be reviewed in Console under Manage &gt; Defenders &gt; Manage.</td>
</tr>
<tr>
<td>activeDefenders</td>
<td>Gauge</td>
<td>Total number of all Defenders for which a license is allocated, regardless of whether it is currently connected to Console or not.</td>
</tr>
<tr>
<td>images_critical_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of containers impacted by critical vulnerabilities.</td>
</tr>
<tr>
<td>images_high_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of containers impacted by high vulnerabilities.</td>
</tr>
<tr>
<td>images_medium_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of containers impacted by medium vulnerabilities.</td>
</tr>
<tr>
<td>images_low_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of containers impacted by low vulnerabilities.</td>
</tr>
<tr>
<td>Metric</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hosts_critical_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of hosts impacted by critical vulnerabilities.</td>
</tr>
<tr>
<td>hosts_high_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of hosts impacted by high vulnerabilities.</td>
</tr>
<tr>
<td>hosts_medium_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of hosts impacted by medium vulnerabilities.</td>
</tr>
<tr>
<td>hosts_low_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of hosts impacted by low vulnerabilities.</td>
</tr>
<tr>
<td>serverless_critical_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of serverless functions impacted by critical vulnerabilities.</td>
</tr>
<tr>
<td>serverless_high_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of serverless functions impacted by high vulnerabilities.</td>
</tr>
<tr>
<td>serverless_medium_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of serverless functions impacted by medium vulnerabilities.</td>
</tr>
<tr>
<td>serverless_low_vulnerabilities</td>
<td>Gauge</td>
<td>Total number of serverless functions impacted by low vulnerabilities.</td>
</tr>
<tr>
<td>images_critical_compliance</td>
<td>Gauge</td>
<td>Total number of images impacted by critical compliance issues.</td>
</tr>
<tr>
<td>images_high_compliance</td>
<td>Gauge</td>
<td>Total number of images impacted by high compliance issues.</td>
</tr>
<tr>
<td>images_medium_compliance</td>
<td>Gauge</td>
<td>Total number of images impacted by medium compliance issues.</td>
</tr>
<tr>
<td>images_low_compliance</td>
<td>Gauge</td>
<td>Total number of images impacted by low compliance issues.</td>
</tr>
<tr>
<td>containers_critical_compliance</td>
<td>Gauge</td>
<td>Total number of containers impacted by critical compliance issues.</td>
</tr>
<tr>
<td>containers_high_compliance</td>
<td>Gauge</td>
<td>Total number of containers impacted by high compliance issues.</td>
</tr>
<tr>
<td>containers_medium_compliance</td>
<td>Gauge</td>
<td>Total number of containers impacted by medium compliance issues.</td>
</tr>
<tr>
<td>containers_low_compliance</td>
<td>Gauge</td>
<td>Total number of containers impacted by low compliance issues.</td>
</tr>
<tr>
<td>hosts_critical_compliance</td>
<td>Gauge</td>
<td>Total number of hosts impacted by critical compliance issues.</td>
</tr>
<tr>
<td>hosts_high_compliance</td>
<td>Gauge</td>
<td>Total number of hosts impacted by high compliance issues.</td>
</tr>
<tr>
<td>hosts_medium_compliance</td>
<td>Gauge</td>
<td>Total number of hosts impacted by medium compliance issues.</td>
</tr>
</tbody>
</table>
### Integrating Prisma Cloud with Prometheus

The Prometheus server scrapes endpoints at configurable time intervals. Prisma Cloud refreshes vulnerability and compliance data every 24 hours. All other data is refreshed every 10 minutes. Regardless of the value you set for the Prometheus scrape interval, new Prisma Cloud data is only available at our refresh rates.

This procedure shows you how to enable Prisma Cloud’s Prometheus integration and spin up a Prometheus server running in a container. If you already have a Prometheus server in your environment, all you need is the Prisma Cloud scrape configuration.

**STEP 1 | Enable Prisma Cloud’s Prometheus instrumentation.**
1. Log into Prisma Cloud Console.
2. Go to System > Logging.
3. Set Prometheus instrumentation to Enabled.

**STEP 2 | Prepare a scrape configuration file for the Prometheus server.**
1. Create a new file named `prometheus.yml`, and open it for editing.
2. Enter the following configuration, where:
- CONSOLE_ADDRESS is the DNS name or IP address for Prisma Cloud Console.
- USER is a Prisma Cloud user, with the minimum role of Auditor.
- PASS is the user’s password.

```yaml
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.

# Prisma Cloud scrape configuration.
scape_configs:
- job_name: 'twistlock'
  static_configs:
    - targets: ['CONSOLE_ADDRESS:8083']
  metrics_path: /api/v1/metrics
  basic_auth:
    username: 'USER'
    password: 'PASS'
```

**STEP 3** | Start the Prometheus server with the scrape configuration file.

```
$ docker run \
--rm \
--network=host \
-p 9090:9090 \
-v /PATH_TO_YML/prometheus.yml:/etc/prometheus/prometheus.yml \
prom/prometheus
```

**STEP 4** | Validate that the Prisma Cloud integration is properly set up In a new browser window, go to http://<PROMETHEUS_HOST>:9090/targets.

---

For testing, restart Console to get results immediately instead of waiting for the first 10 minute window to elapse.

**Using Prometheus with Projects**

If you want to use Prometheus with Projects, modify the scrape configuration file with an additional job for each Twistlock Console.

If you are using tenant projects, enable Prometheus instrumentation in both the Central and Supervisor Consoles.
The following listing shows an example configuration that scrapes three Consoles:

- Central Console.
- A Supervisor Console for a scale project.
- A Supervisor Console for a tenant project.

```yaml
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.

# Twistlock scrape configuration.
scape_configs:
- job_name: 'Central Console'
  static_configs:
    - targets: [CONSOLE_ADDRESS:8083]
      metrics_path: /api/v1/metrics
      basic_auth:
        username: 'USER01'
        password: 'PASS01'

- job_name: 'Tenant Console'
  static_configs:
    - targets: [CONSOLE_ADDRESS:8083]
      metrics_path: /api/v1/metrics
      scheme: http
      params:
        project: [TENANT_PROJECT_NAME]
      basic_auth:
        username: 'USER02'
        password: 'PASS02'

- job_name: 'Scale Console'
  static_configs:
    - targets: [CONSOLE_ADDRESS:8083]
      metrics_path: /api/v1/metrics
      scheme: http
      params:
        project: [SCALE_PROJECT_NAME]
      basic_auth:
        username: 'USER01'
        password: 'PASS01'
```

Where:
- **CONSOLE_ADDRESS** — DNS name or IP address for your Twistlock Central Console
- **USER01** — Twistlock user with access to the Central Console and Scale project
- **PASS01** — USER01’s password
- **USER02** — Twistlock user with access to the Tenant project
- **PASS02** — USER02’s password
- **TENANT_PROJECT_NAME** — name of the tenant project
- **SCALE_PROJECT_NAME** — name of the scale project

The value in `job_name` does not need to match anything else. You can set it to anything.

Create a simple graph

Create a graph that shows the number of deployed Defenders.
STEP 1 | Go to http://<PROMETHEUS_HOST>:9090/graph

STEP 2 | Click Add Graph.

STEP 3 | In the drop-down list, select twistlock_total_defenders.

STEP 4 | Click Execute. In the Console tab, you will see the value for total number of Defenders connected to Console.

STEP 5 | Open the Graph tab to see a visual representation of how the number of Defenders has changed over time.
Kubernetes auditing

The Kubernetes auditing system records the activities of users, administrators, and other components, that have affected the cluster. Prisma Cloud can ingest, analyze, and alert on security-relevant events. Write custom rules or leverage Prisma Cloud Labs prewritten rules to assess the incoming audit stream and surface suspicious activity.

To get started, configure your cluster to forward audits to Prisma Cloud. Prisma Cloud supports both Google Kubernetes Engine (GKE) and self-managed clusters.

Rule library

Custom rules are stored in a central library, where they can be reused. Besides your own rules, Prisma Cloud Labs also distributes rules via the Intelligence Stream. These rules are shipped in a disabled state by default. You can review, and optionally apply them at any time.

Your Kubernetes audit policy is defined in Defend > Access > Kubernetes, and formulated from the rules in your library. There are four types of rules, but the only one relevant to the audit policy is the kubernetes-audit type. Custom rules are written and managed in Console under Defend > Runtime > Custom Rules with an online editor. The compiler checks for syntax errors when you save a rule.

Expression grammar

Expressions let you examine contents of a Kubernetes audit. Expressions have the following grammar:

expression: term (op term | in )*

- term --
  integer | string | keyword | event | (' expression ') | unaryOp term
- in --
  '(' integer | string (',' integer | string)*)?
- op --
  and | or | > | < | >= | # | = | !=
- unaryOp --
  not
- keyword --
 startswith | contains
- string --
  Strings must be enclosed in double quotes
- integer --
  int
- event --
  process, file system, or network

Kubernetes audit events

When Prisma Cloud receives an audit, it is assessed against your policy. Like all policies in Prisma Cloud, rule order is important. Rules are processed top to bottom, and processing stops at the first match. When a rule matches, an alert is raised.
Write rules to surface audits of interest. Rules are written with the jpath function. The jpath function extracts fields from JSON objects, which is the format of a Kubernetes audit. The extracted string can then be compared against strings of interest. The primary operators for jpath expressions are ' = ', ' in ', and ' contains '. For non-trivial examples, look at the Prisma Cloud Lab rules.

The argument to jpath is a single string. The right side of the expression must also be a string. A basic rule with a single jpath expressions has the following form:

\[
\text{jpath("path.in.json.object")} = \text{"something"}
\]

Let's look at some examples using the following JSON object as our example audit.

```json
{
    "user":{
        "uid":"1234",
        "username":"some-user-name",
        "groups":[
            "group1",
            "group2"
        ]
    },
    "stage":"ResponseComplete"
}
```

To examine user's UID, use the following syntax. This expression evaluates to true.

\[
\text{jpath("user.uid")} = \text{"1234"}
\]

To examine username, use the following syntax:

\[
\text{jpath("user.username")} = \text{"some-user-name"}
\]

To examine the stage field, use the following syntax:

\[
\text{jpath("stage")} = \text{"ResponseComplete"}
\]

To examine the groups list field, use the following syntax:

\[
\text{jpath("user.groups")} \text{ contains } \text{"group1"}
\]

Or alternatively:

\[
\text{jpath("user.groups")} \text{ in } \{\text{"group1"}, \text{"group2"}\}
\]

### Integrating with self-managed clusters

Prisma Cloud supports any self-managed cluster based on Kubernetes version 1.13, or later. You can deploy clusters with any number of tools, including kubeadm.

**Prerequisites:** You've already deployed a Kubernetes cluster.

#### Configure the API server

Configure the API server.

**STEP 1** Open `/etc/kubernetes/manifests/kube-apiserver.yaml` for editing.
STEP 2 | Add the following flags. Changes to `kube-apiserver.yaml` trigger the API server to restart.

```
--audit-dynamic-configuration
--feature-gates=DynamicAuditing=true
--runtime-config=auditregistration.k8s.io/v1alpha1=true
```

STEP 3 | Verify the server has restarted with the right flags.

```
$ ps -ef | grep kube-apiserver
```

Configure OpenShift 4.2+

OpenShift 4.2 uses Kubernetes 1.14 dynamic audit backend to configure webhook backends through an AuditSink API object.

STEP 1 | Authenticate to the OpenShift cluster as cluster-admin using the OC cli.

STEP 2 | Issue the command to patch the API server to allow the creation of dynamic audit backends.

```
$ oc patch kubeapiserver cluster --type=merge -p '{"spec":
{"unsupportedConfigOverrides":{"apiServerArguments":{"audit-dynamic-
configuration":["true"],"feature-gates":["DynamicAuditing=true"],"runtime-
config":["auditregistration.k8s.io/v1alpha1=true"]}}}'}
```

Configure your cluster to forward audits to Prisma Cloud

Configure your cluster to forward audits to Prisma Cloud.

STEP 1 | Open Console.

STEP 2 | Go to Defend > Access > Kubernetes.

STEP 3 | Set Kubernetes auditing to Enabled.

STEP 4 | Click Go to settings.

1. Set Deployment type to Default.
2. Copy the webhook URL. This where your cluster will send audits.

STEP 5 | Configure the webhook in the cluster with an AuditSink object.

To route audits over HTTP: Create a file named `auditsink.yaml`. Paste the following listing into it. Replace WEBHOOK-URL with the URL you copied from Prisma Cloud Console.

```
apiVersion: auditregistration.k8s.io/v1alpha1
kind: AuditSink
metadata:
  name: twistlock-sink
spec:
  policy:
    level: Request
    stages:
    - ResponseComplete
  webhook:
    throttle:
      qps: 10
      burst: 15
```
To route audits over HTTPS: Create a file named `auditsink.yaml`. Paste the following listing into it. Replace WEBHOOK-URL with the URL you copied from Prisma Cloud Console. Replace CA-BUNDLE with a PEM-encoded CA bundle, which the cluster can use to validate Prisma Cloud's certificate.

```yaml
apiVersion: auditregistration.k8s.io/v1alpha1
kind: AuditSink
metadata:
  name: twistlock-sink
spec:
  policy:
    level: Request
    stages:
      - ResponseComplete
  webhook:
    throttle:
      qps: 10
      burst: 15
  clientConfig:
    url: "WEBHOOK-URL"
    caBundle: CA-BUNDLE
```

**STEP 6** | Create the AuditSink object in your cluster.

$ kubectl apply -f auditsink.yaml

Your cluster now forwards audits to Prisma Cloud Console.

Integrating with GKE

On GKE, the master node isn’t accessible, so you cannot directly configure the API server with the `--audit-dynamic-configuration` flag to push audits to Prisma Cloud. Instead, Prisma Cloud retrieves audits from Stackdriver, polling it every 10 minutes for new data.

Note that there can be some delay between the time an event occurs in the cluster and when it appears in Stackdriver. Due to Twistlock’s polling mechanism, there’s another delay between the time an audit arrives in Stackdriver and it appears in Prisma Cloud.

For testing purposes, you might not want to wait for the 10 minute polling period to see audits in Prisma Cloud. After setting up the integration in Prisma Cloud by providing your GCP credentials, you can force Prisma Cloud to immediately poll Stackdriver by disabling then re-enabling the Kubernetes audit feature in **Defend > Access > Kubernetes**.

Prisma Cloud supports GKE clusters version 1.11.6-gke.3, or later.

**Prerequisites:** You’ve created a service account with one of the following authorization scopes:

- https://www.googleapis.com/auth/logging.read
- https://www.googleapis.com/auth/logging.admin
- https://www.googleapis.com/auth/cloud-platform.read-only
- https://www.googleapis.com/auth/cloud-platform

**STEP 1** | Open Console.
STEP 2 | Go to **Defend > Access > Kubernetes**.

STEP 3 | Set **Kubernetes auditing** to **Enabled**.

STEP 4 | Click **Go to settings**.

  1. Set **Deployment type** to **GKE**.
  2. Select a GCP credential with the minimum authorization scope described in the prerequisites. If there are no accounts to select, add one to the **credentials store**.
  3. (Optional) Set an advanced filter to reduce the amount of data transferred from Stackdriver. If your project has more than one cluster, consider filtering my cluster name with `resource.labels.cluster_name="YOUR_CLUSTER_NAME"`. Do not use the `resource.type` or `timestamp` filters because Prisma Cloud uses them internally.
  4. Click **Save**.

**CA bundle**

If you're sending audit data to Prisma Cloud's webhook over HTTPS, you must specify a CA bundle in the AuditSink object.

If you've customized Console's certificate, you can get a copy from **Manage > Authentication > System-certificates > TLS certificate for Console**. Paste the certificate into a file named `server-cert.pem`, then run the following command:

```bash
$ openssl base64 -in server-cert.pem -out base64-output -A
```

In the AuditSingle object, set the value of caBundle to the contents of the base64-output file.

**Testing your setup**

Write a new rule, or select a prewritten rule from the inventory, and add it to your audit policy. This setup installs a rule that fires when privileged pods are created in the cluster.

STEP 1 | Open Console, and go to **Defend > Access > Kubernetes**.

STEP 2 | Add a Prisma Cloud Labs prewritten rule.

  1. Click **Select rules**.
  2. If you're integrated with a managed cluster, select **Prisma Cloud Labs - Privileged pod creation**. If you're integrated with GKE, select **Prisma Cloud Labs - GKE - privileged pod creation**.

  > *There are separate rules for standard Kubernetes and GKE because the structure of the audits are different. Therefore, the logic for parsing the audit JSON is different.*

  3. Click **Save**.

STEP 3 | Create a pod deployment file named `priv-pod.yaml`, and enter the following contents.

```yaml
apiVersion: v1
class: Pod
metadata:
  name: nginx
labels:
  app: nginx
spec:
  containers:
  - name: nginx
```
STEP 4 | Create the privileged pod.

```bash
kubectl apply -f priv-pod.yaml
```

STEP 5 | Verify an audit was created.

Go to Monitor > Events, and select the Kubernetes Audits filter.

If you’re integrated with GKE, and you don’t see the audit, force Prisma Cloud to poll Stackdriver immediately by disabling then re-enabling the feature.
Prisma Cloud ships a command-line configuration and control tool called twistcli. It lets you deploy Prisma Cloud components, run scans, and more. It is supported on Linux, macOS, and Windows.

- twistcli
- Scan images with twistcli
- Install Console with twistcli
- Update the Intelligence Stream in offline environments
twistcli

Prisma Cloud ships a command-line configuration and control tool known as *twistcli*. It is supported on Linux, macOS, and Windows.

When users from a tenant project run *twistcli*, they must set the --project option to specify the proper context for the command.

Installing twistcli

The *twistcli* tool is delivered with every Prisma Cloud release. It is statically compiled, so it does not have any external dependencies, and it can run on any Linux host. No special installation is required. To run it, simply copy it to a host, and give it executable permissions.

The *twistcli* tool is available from a number of places:

- It's included with the release tarball.
- It can be downloaded from the Console UI. Go to Manage > System > Downloads.
- It can be downloaded from the API, which is typical use case for automated workflows. For more information, see the `/api/v1/util` endpoint.

The requirements for running *twistcli* are:

- *twistcli* must be able to connect to Console over the network from the host where it runs.
- For image scanning, Docker Engine must be installed on the executing machine.

Connectivity to Console

Most *twistcli* functions require connectivity to Console. All example commands specify a variable called COMPUTE_CONSOLE, which represents the address for your Console.

The address for your Console depends on how you installed it.

For Onebox installs, where you install Console on a stand-alone host, the value for COMPUTE_CONSOLE is the IP address or DNS name of the host. HTTPS access to Console is servered on port 8083, so the full address would be:

https://<IPADDR>:8083

For the default Kubernetes installation procedure, the Console service is exposed by a LoadBalancer, and so the address for COMPUTE_CONSOLE is

https://<LOAD_BALANCER>:8083

Functions

The *twistcli* tool supports the following functions:

- **console** — Installs and uninstalls Console into a cluster. Kubernetes, OpenShift, and Docker Swarm are supported. You can also export Kubernetes or OpenShift deployment files in YAML format.
- **defender** — Installs and uninstalls Defender into a cluster. Kubernetes, OpenShift, and Docker Swarm are supported. Defender is installed as either a daemon set (Kubernetes, OpenShift) or global service (Docker Swarm), which means one Defender is always automatically deployed to each node in the cluster. You can also export a Kubernetes or OpenShift deployment file in YAML format.
- **hosts** — Scans hosts for vulnerabilities and compliance issues.
- **images** — Scans container images for vulnerabilities and compliance issues. Because it runs from the command line, you can easily integrate Prisma Cloud’s scanning capabilities into your CI/CD pipeline.
- **intelligence**— Retrieves the latest threat data from the Prisma Cloud Intelligence Stream, and push those updates to a Prisma Cloud installation running in an air-gapped environment.
- **pcf**— Scan Pivotal Cloud Foundry droplets.
- **app-embedded**— Embed the App Embedded Defender into a Dockerfile.
- **restore**— Restore Console to the state stored in the specified backup file. An automated backup system (enabled by default) creates and maintains daily, weekly, and monthly backups. Additional backups can be made at any point in time from the Console UI.
- **serverless**— Scans serverless functions for vulnerabilities.
- **iac**— Scan Infrastructure-as-Code (IaC) templates for potential issues and misconfigurations. Learn more about Prisma Cloud IaC scanning capability.
- **support**— Streamlines the process of collecting and sending debug information to Prisma Cloud’s support team. Collects log data from a node and uploads it to Prisma Cloud’s support area.

### Capabilities

The **twistcli** tool offers feature parity across all supported operating systems, with a few exceptions. The following table highlights where functions are disabled, or work differently, on a given platform.

<table>
<thead>
<tr>
<th>Command</th>
<th>Subcommand</th>
<th>Linux</th>
<th>macOS</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>console</strong></td>
<td>export</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>install</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>uninstall</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>defender</strong></td>
<td>export</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>install</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>uninstall</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>hosts</strong></td>
<td>scan</td>
<td>Yes</td>
<td>No(^1)</td>
<td>No</td>
</tr>
<tr>
<td><strong>images</strong></td>
<td>scan</td>
<td>Yes</td>
<td>Yes(^2)</td>
<td>Yes(^3)</td>
</tr>
<tr>
<td><strong>intelligence</strong></td>
<td>upload</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>download</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>pcf</strong></td>
<td>scan</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>app-embedded</strong></td>
<td>embed</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>restore</strong></td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>serverless</strong></td>
<td>scan</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>support</strong></td>
<td>dump</td>
<td>Yes</td>
<td>No(^4)</td>
<td>No(^4)</td>
</tr>
</tbody>
</table>
Twistcli can scan Windows images on Windows Server 2016 and Windows Server 2019 hosts. To scan Linux images on Windows, install Docker Machine on Windows with the Microsoft Hyper-V driver. Twistcli does not support scanning Linux images on Windows hosts with Docker for Windows.

The `support dump` function collects Console’s logs when Console malfunctions. Copy `twistcli` to host where Console runs, then execute `twistcli support dump`. Defender logs can be retrieved directly from the Console UI under Manage > Defenders > Manage.

For a comprehensive list of supported options for each subcommand, run:

```
$ twistcli <COMMAND> --help
```

### Install support

Support for installing Console and Defender via `twistcli` is supported on several cluster types. The following table highlights the available support:

<table>
<thead>
<tr>
<th>twistcli</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>export</td>
<td>No</td>
</tr>
<tr>
<td>install</td>
<td>No</td>
</tr>
<tr>
<td>uninstall</td>
<td>No</td>
</tr>
</tbody>
</table>

**console**

<table>
<thead>
<tr>
<th>&gt;Command</th>
<th>&gt;Subcommand</th>
<th>&gt;Stand-alone</th>
</tr>
</thead>
<tbody>
<tr>
<td>export</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>install</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>uninstall</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**defender**

<table>
<thead>
<tr>
<th>&gt;Command</th>
<th>&gt;Subcommand</th>
<th>&gt;Stand-alone</th>
</tr>
</thead>
<tbody>
<tr>
<td>export</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>install</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>uninstall</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1 Stand-alone refers to installing an instance of Console or Defender onto a single host that isn’t part of a cluster. For stand-alone installations of Console, use the `twistlock.sh` script to install Onebox. For stand-alone installations of Defender, log into Console, go to Manage > Defenders > Deploy, and generate an install command.

The `twistcli console install` command for Kubernetes and OpenShift combines two steps into a single command to simplify how Console is deployed. This command internally generates a YAML configuration file and then creates Console’s resources with `kubectl create` in a single shot. This command is only supported on Linux. Use it when you don’t need a copy of the YAML configuration file. Otherwise, use `twistcli console export`.

---

1 Prisma Cloud doesn’t support deployment to macOS hosts, so there is no support for scanning macOS hosts.

2 Scans Linux images on macOS hosts. Docker for Mac must be installed.

3 Prisma Cloud doesn’t support deployment to macOS hosts, so there is no support for scanning macOS hosts.

4 The `support dump` function collects Console’s logs when Console malfunctions. Copy `twistcli` to host where Console runs, then execute `twistcli support dump`. Defender logs can be retrieved directly from the Console UI under Manage > Defenders > Manage.

For a comprehensive list of supported options for each subcommand, run:

```
$ twistcli <COMMAND> --help
```
Scan images with twistcli

Prisma Cloud ships a command-line scanner for scanning container images and serverless functions. It is supported on Linux, macOS, and Windows.

Command reference

The `twistcli` command has several subcommands. Use the `twistcli images scan` subcommand to invoke the scanner.

Projects

When users from a tenant project run `twistcli`, they must set the `--project` option to specify the proper context for the command.

```
twistcli images scan --project "<project_name>"
```

Command

```
twistcli images scan -- Scan an image for vulnerabilities and compliance issues. The image must reside on the system where twistcli runs. If not, retrieve the image with docker pull before scanning it. Twistcli does not pull images for you.
```

Synopsis

```
twistcli images scan [OPTIONS] [IMAGE]
```

Description

The `twistcli images scan` function collects information about the packages and binaries in the container image, and then sends it to Console for analysis.

Data collected by `twistcli` includes:

- Packages in the image.
- Files installed by each package.
- Hashes for files in the image.

After Console analyzes the image for vulnerabilities, `twistcli`:

- Outputs a summary report.
- Exits with a pass or fail return value.

Scan results can be retrieved in JSON format from the Console using API calls.

To specify an image to scan, use either the image ID, or repository name and tag. The image should be present on the system, having either been built or pulled there. If a repository is specified without a tag, `twistcli` looks for an image tagged `latest`.

**When invoking twistcli, the last parameter should be the image to scan. If you list options after the image, they will be ignored.**

Options

- `--address URL --`
Complete URL for Console, including the protocol and port. Only the HTTPS protocol is supported. By default, Console listens to HTTPS on port 8083, although your administrator can configure Console to listen on a different port. Defaults to https://127.0.0.1:8083.

Example: --address https://console.example.com:8083

- **-u, --user USERNAME**
  Username to access Console. If not provided, the `TWISTLOCK_USER` environment variable will be used if defined, or “admin” is used as the default.

- **-p, --password PASSWORD**
  Password for the user specified with -u, --user. If not specified on the command-line, the `TWISTLOCK_PASSWORD` environment variable will be used if defined, or otherwise will prompt for the user’s password before the scan runs.

- **--project PROJECT NAME**
  Interface with a specific supervisor Console to retrieve policy and publish results.
  Example: --project "Tenant Console"

- **--output-file FILENAME**
  Write the results of the scan to a file in JSON format.
  Example: --output-file scan-results.json

- **--details**
  Show all vulnerability details.

- **--containerized**
  Run the scan from inside the container.

- **--custom-labels**
  Include the image custom labels in the results.

- **--docker-address DOCKER_CLIENT_ADDRESS**
  Docker daemon listening address (default: unix:///var/run/docker.sock). Can be specified with the `DOCKER_CLIENT_ADDRESS` environment variable.

- **--docker-tlsca PATH**
  Path to Docker client CA certificate.

- **--docker-tlscert PATH**
  Path to Docker client Client certificate.

- **--docker-tlskey PATH**
  Path to Docker client Client private key.

- **--tlsca PATH**
  Path to Prisma Cloud CA certificate file. If no CA certificate is specified, the connection to Console is insecure.

- **--podman-path PATH**
  Forces twistcli to use Podman. To use the default installation path, set as podman. Otherwise, provide the appropriate path.

- **--include-js-dependencies**
  Evaluates packages listed only in manifests.
• **--token** TOKen

  Token to use for Prisma Cloud Console authentication. Tokens can be retrieved from the API endpoint `api/v1/authenticate` or from the Manage > Authenticate > User Certificates page in Console.

**RETURN VALUE**

The exit code is 0 if `twistcli images scan` finds no vulnerabilities or compliance issues. Otherwise, the exit code is 1.

The criteria for passing or failing a scan is determined by the CI vulnerability and compliance policies set in Console. The default CI vulnerability policy alerts on all CVEs detected. The default CI compliance policy alerts on all critical and high compliance issues.

There are two reasons why `twistcli images scan` might return an exit code of 1.

- The scan failed because the scanner found issues that violate your CI policy.
- Twistcli failed to run due to an error.

Although the return value is ambiguous — you cannot determine the exact reason for the failure by just examining the return value — this setup supports automation. From an automation process perspective, you expect that the entire flow will work. If you scan an image, with or without a threshold, either it works or it does not work. If it fails, for whatever reason, you want to fail everything because there is a problem.

**Scan results**

To view scan reports in Console, go to Monitor > Vulnerabilities > Images > CI or Monitor > Compliance > Images > CI.

You can also retrieve scan reports in JSON format using the Prisma Cloud API, see the API section.

**Output**

The twistcli tool can output scan results to several places:

- stdout.
- File. Scan results are saved in JSON format.
- Console. Scan results can be viewed under Monitor > Vulnerabilities > Images > CI.

By passing certain flags, you can adjust how the twistcli scan output looks and where it goes. By default, twistcli writes scan results to stdout and sends the results to Console.

To write scan results to stdout in tabular format, pass the `--details` flag to twistcli. This does not affect where the results are sent.

To write scan results to a file in JSON format, pass the `--output-file` flag to twistcli. If you specify an output file, then results cannot be sent to Console.

**API**

You can retrieve scan reports in JSON format using the Prisma Cloud Compute API. The API returns comprehensive information for each scan report, including the full list of packages, files, and vulnerabilities.

The following example `curl` command calls the API with Basic authentication. You’ll need to apply some filtering with tools like `jq` to extract specific items from the response. For more information on accessing the API, see Accessing the API.

```
$ curl
```
If you are using assigned collections, then specify the collection in a query parameter:

```
$ curl \
  -u <COMPUTE_CONSOLE_USER> \ 
  -o scan_results.json \ 
  'https://<COMPUTE_CONSOLE>/api/v1/scans?type=ciImage&collections=<COLLECTION_NAME>'
```

### Dockerless scan

By default, twistcli is run from outside the container image.

### Podman Twistcli scans

Twistcli can run scans on Podman hosts. Use `--podman-path PATH` to specify the path to podman and force the twistcli scanner to use podman. For additional information, see the Podman section.

### Running from inside of the container

In some cases, you might need to copy twistcli to the container’s file system, and then run the scanner from inside the container.

One reason you might want to run the scanner this way is when your build platform doesn't give you access to the Docker socket. CodeFresh is an example of such a platform.

There are some shortcomings with scanning from inside a container, so you should only use this approach when no other approach is viable. The shortcomings are:

- Automating the scan in your continuous integration pipeline is more difficult.
- Image metadata, such as registry, repository, and tag aren't available in the scan report. When twistcli is run from outside the container, this information is retrieved from the Docker API.
- The image ID isn't available in the scan report because it cannot be determined when the scan is run from inside a container.
- The scan report won't show a layer-by-layer analysis of the image.

### Usage

When running the scanner from inside a container, you need to properly orient it by passing it the `--containerized` flag. There are a couple of ways to run twistcli with the `--containerized` flag: build-time and run-time.

For security reasons, Prisma Cloud recommends that you create a user with the CI User role for running scans.

#### Build-time invocation

After building an image, run it. Mount the host directory that holds the twistcli binary, pass the Prisma Cloud Console user credentials to the container with environment variables, then run the scanner inside the container. The `<REPORT_ID>` is a user defined string that uniquely identifies the scan report in the Console UI.

```
$ docker run \
  -v /PATH/TO/TWISTCLIDIR:/tools \ 
  -e TW_USER=<COMPUTE_CONSOLE_USER> \ 
  -e TW_PASS=<COMPUTE_CONSOLE_PASSWD> \ 
  -e TW_CONSOLE=<COMPUTE_CONSOLE> \
```
Rather than username and password, twistcli can also authenticate to Console with a token. Your API token can be found in Console under Manage > Authentication > User Certificates > API token. For security reasons, API tokens expire.

$ docker run
 -v /PATH/TO/TWISTCLI_DIR:/tools
 -e TW_TOKEN=<API_TOKEN>
 -e TW_CONSOLE=<COMPUTE_CONSOLE>
 --entrypoint="" \
 /tools/twistcli images scan \
 --containerized \
 --details \
 --address $TW_CONSOLE \
 --token $TW_TOKEN \
 <REPORT_ID>

**Run-time invocation**

If you have access to the orchestrator, you can exec into the running container to run the twistcli scanner. Alternatively, you could SSH to the container. Once you have a shell on the running container, invoke the scanner:

$ ./twistcli images scan \
 --address <COMPUTE_CONSOLE> \
 --user <COMPUTE_CONSOLE_USER> \
 --password <COMPUTE_CONSOLE_PASSWD> \
 --containerized \
 <REPORT_ID>

To invoke the scanner with an API token:

$ ./twistcli images scan \
 --address <COMPUTE_CONSOLE> \
 --token <API_TOKEN> \
 --containerized \
 <REPORT_ID>

**Simple scan**

Scan an image with twistcli and print the summary report to stdout.

Scan an image named *myimage:latest*.

$ twistcli images scan \
 --address <COMPUTE_CONSOLE> \
 --user <COMPUTE_CONSOLE_USER> \
 --password <COMPUTE_CONSOLE_PASSWD> \
 myimage:latest
You can have twistcli generate a detailed report for each scan. The following procedure shows you how to scan an image with twistcli, and then retrieve the results from Console.

**STEP 1** | Scan an image named `myimage:latest`.

```
$ twistcli images scan \
   --address <COMPUTE_CONSOLE> \
   --user <COMPUTE_CONSOLE_USER> \
   --password <COMPUTE_CONSOLE_PASSWD> \
   --details \
   myimage:latest
```

Sample command output (results have been truncated):

```
myimage:latest sha256:2073e0bcb60ee98548d313ead5eacbfe16d9054f8

for image myimage:latest: total - 35, critical - 0, high - 2, medium - 1, low - 33, medium check results: PASS

myimage:latest: total - 1, critical - 0, high - 1, medium - 1, low - 0, medium check results: PASS
```
<table>
<thead>
<tr>
<th>VERSION</th>
<th>STATUS</th>
<th>PUBLISHED</th>
<th>DI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>open</td>
<td>&gt; 9 months</td>
<td>&gt;</td>
</tr>
<tr>
<td></td>
<td>fixed in 2.0.5-1+deb10u1</td>
<td>&gt; 4 months</td>
<td>&gt;</td>
</tr>
<tr>
<td>deb10u2</td>
<td>fixed in 1.44.5-1+deb10u3</td>
<td>47 days</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>open</td>
<td>&gt; 3 months</td>
<td>&gt;</td>
</tr>
<tr>
<td>191117-2~deb10u1</td>
<td>open</td>
<td>&gt; 2 years</td>
<td>&gt;</td>
</tr>
<tr>
<td>10u2</td>
<td>open</td>
<td>&gt; 10 months</td>
<td>&gt;</td>
</tr>
<tr>
<td>10u2</td>
<td>open</td>
<td>&gt; 10 months</td>
<td>&gt;</td>
</tr>
</tbody>
</table>
STEP 2 | This outputs a tabular representation of your scan results to stdout. If you need to retrieve the results of your scan in JSON format, this can be done using the API. For more information on the API, refer to Accessing the API.

1. Call the API with authentication (demonstrated here using Basic authentication) to fetch the results of the scan.

   ```bash
   $ curl \
   -o scan_results.json \
   -H 'Authorization: Basic YXBpOmFwaQ==' \
   'https://<COMPUTE_CONSOLE>/api/v1/scans?
   search=myimage&limit=1&reverse=true&type=ciImage'
   ```

2. Format the scan results into human-readable format.

   ```bash
   $ python -m json.tool scan_results.json > scan_results_pp.json
   ```

3. Inspect the results.

   Open `scan_results_pp.json` to view the results. Vulnerability information can be found in the `vulnerabilities` array, and compliance results can be found in the `complianceIssues` array.

   ```json
   [
   {
   "entityInfo": {
   "_id": "",
   "type": "ciImage",
   ...
   "complianceIssues": [
   {
   "text": "",
   "id": 41,
   "severity": "high",
   "cvss": 0,
   "status": "",
   "cve": "",
   "cause": "",
   "description": "It is a good practice to run the container as a non-root user, if possible. Though user namespace mapping is now available, if a user is already defined in the container image, the container is run as that user by default and specific user namespace remapping is not required",
   "title": "(CIS_Docker_CE_v1.1.0-4.1) Image should be created with a non-root user",
   "vecStr": "",
   "exploit": "",
   "riskFactors": null,
   "link": "",
   "type": "image",
   "packageName": "",
   "packageVersion": "",
   "layerTime": 0,
   "templates": []
   },
   "twistlock": false,
   "published": 0,
   "discovered": "0001-01-01T00:00:00Z"
   }
   ...
   "vulnerabilities": [
   {
   "text": "",
   ```
Scan images built with Jenkins in an OpenShift environment

If you are building and deploying images on OpenShift Container Platform (OCP), and you are utilizing their Jenkins infrastructure, then invoke a scan with the `twistcli hosts scan` command, not the `twistcli images scan` command.

You can scan images generated by Jenkins with the OpenShift plugin by invoking twistcli from a build hook. Build hooks let you inject custom logic into the build process. They run your commands inside a temporary container instantiated from build output image. Build hooks are called when the last layer of the image has been committed, but before the image is pushed to a registry. An non-zero exit code fails the build. A zero exit code passes the build, and allows it to proceed to the next step.

To call twistcli from a build hook:
**STEP 1** | Download twistcli into your build environment. Depending on your build strategy, one option is to download it as an external artifact using a **save-artifacts S2I** script.

**STEP 2** | In your **BuildConfig**, call twistcli as a **script** from the **postCommit** hook.

```bash
$ twistcli hosts scan \
   --address <COMPUTE_CONSOLE> \
   --user <COMPUTE_CONSOLE_USER> \
   --password <COMPUTE_CONSOLE_PASSWD> \
   --skip-docker \
   --include-3rd-party
```

Where the **--skip-docker** option skips all Docker compliance checks such as the Docker daemon configuration and the **--include-3rd-party** option scans application-specific files such as JARs.

### Scan images when the Docker socket isn’t in the default location

The twistcli scanner uses the Docker API, so it must be able to access the socket where the Docker daemon listens. If your Docker socket isn’t in the default location, use the **--docker-address** option to tell twistcli where to find it:

- **--docker-address PATH**
  Path to the Docker socket. By default, twistcli looks for the Docker socket **unix:///var/run/docker.sock**.

```bash
$ ./twistcli images scan  \
   --address <COMPUTE_CONSOLE> \
   --user <COMPUTE_CONSOLE_USER> \
   --password <COMPUTE_CONSOLE_PASSWD> \
   --docker-address unix:///<PATH/TO>/docker.sock \
   <IMAGE_NAME>
```

### Scan Podman/CRI images

Podman is a daemon-less container engine for developing, managing, and running OCI containers on Linux. The twistcli tool can use the preinstalled Podman binary to scan CRI images.

- **--podman-path PATH**
  Forces twistcli to use Podman. To call podman from its default install path, specify **podman**. Otherwise, specify an explicit path.

```bash
$ ./twistcli images scan  \
   --address <COMPUTE_CONSOLE> \
   --user <COMPUTE_CONSOLE_USER> \
   --password <COMPUTE_CONSOLE_PASSWD> \
   --podman-path podman  \
   <IMAGE_NAME>
```

### CI/CD Automation

Twistcli images scan can be used to shift-left security scans inside of your build pipeline. Plugins are available for Jenkins and other CI/CD tools, but twistcli can also be used from a CI pipeline in order to initiate vulnerability and compliance scans on images.

The exit status code can be verified inside of your pipeline to determine pass and fail status of the image scan. A zero exit code signals the scan passes, and any non-zero exit code signals a failure.
In order to automate the download and version sync of twistcli, reference the sample Jenkins code below:

```java
stage('Check twistcli version') {
    def TCLI_VERSION = sh(script: './twistcli | grep -A1 VERSION | sed 1d', returnStdout:true).trim()
    println "TCLI_VERSION = $TCLI_VERSION"
    println "CONSOLE_VERSION = $CONSOLE_VERSION"
    if ('$TCLI_VERSION' != '$CONSOLE_VERSION') {
        println "downloading twistcli"
        sh 'curl -k -u $TL_USER:$TL_PASS --output ./twistcli https://$TL_CONSOLE/api/v1/util/twistcli'
        sh 'sudo chmod a+x ./twistcli'
    }
}

stage('Scan with Twistcli') {
    sh './twistcli images scan --address https://$TL_CONSOLE -u $TL_USER -p $TL_PASS --details $IMAGE'
}
```
Install Console with twistcli

When twistcli installs Console into a Kubernetes or OpenShift cluster, it executes a series of steps. To help you troubleshoot issues when twistcli fails, the steps in the install flow are described here:

When you run `twistcli console install`, it:

1. Loads the Console image on localhost, and tags it with the registry address.
2. Deletes the old Console replication controller, if it exists, and waits for Console deletion.
3. Deletes the config map, if it exists.
4. Creates Prisma Cloud namespace, if it does not exist.
5. If the service does not exist, twistcli resolves the service template to a file and creates a new service.
6. If persistent volume claim (PVC) does not exist, twistcli resolves the PVC template to a file and creates a new PVC.
7. Waits to the PVC to bind to a persistent volume resource. twistcli expects that the persistent volume has already been created by the user. Note that the PVC is not deleted and recreated because once the PVC is be deleted, it cannot bind again to the persistent volume without recreating the persistent volume.
8. Retrieves the service IPs (Cluster IPs, and adds them to the SAN.
9. Creates a config map.
10. Resolves Console template to a file, and creates a Console replication controller.
11. Deletes the working directory.
Update the Intelligence Stream in offline environments

Prisma Cloud provides a method to update Console’s vulnerability and threat data even if it runs in an offline environment.

The Prisma Cloud Intelligence Stream (IS) is a real-time feed that contains vulnerability data and threat intelligence from commercial providers, Prisma Cloud Labs, and the open source community.

When you install Prisma Cloud, Console is automatically configured to connect to intelligence.twistlock.com to download updates. The IS is updated several times per day. Each time the IS is updated, Console immediately downloads the new data.

If you run Prisma Cloud in an offline environment, where Console does not have access to the Internet to download updates from the IS, then you can manually download and install IS updates.

Downloading updates from the IS

Before starting, ensure the Internet-connected host to where you will initially download the updates can access the Intelligence Stream. The most reliable way to test connectivity is to ping the Intelligence Stream. This following curl command verifies that name resolution and any intermediary HTTP proxies are functioning properly.

```
$ curl -k \
--silent \
--output /dev/null \
--write-out "%{http_code}\n" \
https://intelligence.twistlock.com/api/v1/_ping
```

If you’ve got connectivity, you’ll get back a 200 (Successful) response code.

| 200 |

STEP 1 | Open Console.

STEP 2 | Go to Manage > System > Intelligence.

STEP 3 | Copy the access token.

STEP 4 | Download the data in the Intelligence Stream.

1. Open a shell window
2. Go to the directory where you unpacked the Prisma Cloud release tarball.
3. Run the following command, using the access token you downloaded.

```
$ ./linux/twistcli intelligence download --token <TWISTLOCK-ACCESS-TOKEN>
```

The script downloads all the feeds and generates an archive named `twistlock_feed_<random_string>.tar.gz`
Uploading IS updates to Console

Use the `twistcli` tool to upload the intelligence stream archive to your Prisma Cloud Console.

**STEP 1 |** Connect to the host machine running Console.

**STEP 2 |** Open a shell window.

**STEP 3 |** Go to the directory where you unpacked the Prisma Cloud release tarball.

**STEP 4 |** Run the following command:

```
$ ./linux/twistcli intelligence upload \
   --address <COMPUTE_CONSOLE> \
   --user <USER> \
   --password <PASSWORD> \
   --tlscacert <PATH-TO-CERT> \n   <TWISTLOCK-FEED-ARCHIVE>
```

Where:

- **<PATH-TO-CERT>** --
  (Optional) Path to to Prisma Cloud’s CA certificate file. With the CA cert, a secure connection is used to upload the intelligence data to Console. For example, `/var/lib/twistlock/certificates/console-cert.pem`.

- **<TWISTLOCK-FEED-ARCHIVE>** --
  Archive generated in Downloading updates from the IS. For example, `twistlock_feed_1524655717.tar.gz`.

**IMPORTANT:** Sometimes after Console is restarted, you may see this error on the login page: “failed to query license”. This is by design, and it is not a bug. It happens because a Console restart triggers a user auth token renewal. For more information, see Long-lived tokens.
Deployment patterns

As you prepare to deploy Prisma Cloud, consider how to tailor it fit into your environment. Prisma Cloud supports multitenancy, which gives you a way to manage all your deployments from a single interface, and control which data each team can see.

> Projects
> Best practices for DNS and certificate management
> Caps
Projects

Some deployments are very large, running on more than 10,000 hosts. Other deployments must be compartmentalized, for regulatory or operational reasons. Projects solve two problems: Scale and multi-tenancy.

Projects let you deploy a single master Console, with a single URL, that can scale out to support an infinite number of container hosts. You can set up an environment that shares the same rules and configurations as the master Console, or deploy separate compartmentalized environments which operate independently with their own rules and configurations.

For example, you might have https://console.customer.com as the single URL for accessing the Console UI and API. Then you could deploy a Console to each of your regional data centers to support a scaled-out production environment, or segregated instances of Console for each business unit, or both, and manage all of them from a single Central Console.

Role-based access control (RBAC) rules manage who can access which project. When users log onto Prisma Cloud Central Console, they are shown a list of projects to which they have access and can switch between them.

Terminology

The following terms are used throughout this article:

- **Central Console** --
  Also known as the master Console or just master. This is the interface from which administrators manage (create, access, and delete) their projects.

- **Supervisor** --
  Secondary, slave Console responsible for the operation of a project. Supervisor Consoles are headless. Their UI and API are not directly accessible. Instead, users interact with a project from Central Console's UI and API.

- **Project** --
  A deployment unit that consists of a Supervisor Console and up to 5,000 Defenders. There are two types of projects: scale projects and tenant projects.

- **Scale project** --
  The Supervisor Console inherits all rules and settings from the Central Console. Stack multiple scale projects together to deploy Prisma Cloud to large environments with a large number of hosts. Each scale project supports 5,000 Defenders. Two scale projects can support an environment with 10,000 hosts, three scale projects supports 15,000 hosts, and so on.

- **Tenant project** --
  Tenant projects maintain all their own rules and settings, separate from Central Console and any other Supervisor Consoles.

When to use projects

Carefully assess whether you need projects. Provisioning projects when they are not required will needlessly complicate the operation and administration of your environment.

1. Does your container environment have more than 5,000 hosts?

If yes, then provision a scale project, where each scale project can handle a maximum of 5,000 hosts (Defenders). Add a scale project for every 5,000 hosts in your environments. Stacked scale projects work together as a single, cohesive environment with shared rules and settings.
If your environment has fewer than 5,000 hosts, then you do not need to provision any scale projects. A single Console will be sufficient for your needs. You can always migrate to a project structure if your environment does grow past 5,000 hosts. For more information, see Migration strategies.

2. Do you have multiple segregated environments, where each environment must be configured with its own rules and policies?

If yes, then deploy a tenant project for each environment.

3. Are you upgrading from Prisma Cloud 2.3?

If yes, then your existing deployment will continue to work exactly as it did before.

Migrating to projects is not required. Projects are an optional feature that is disabled by default. There is no need to migrate to projects unless you have a specific need for the functionality it offers.

If you are using Prisma Cloud 2.3, and you've deployed multiple Consoles, you can easily adopt projects to fold your stand-alone Consoles under the management of a single Central Console. First upgrade all your Consoles to Twistock 2.4, then see Migration strategies.

4. If you choose not to use projects now, can you migrate to projects at a later time?

Yes. Even if you choose not to use projects now, you're not locked into that decision. You can always migrate to projects at a later time.

Architecture

Projects federate the UI and API for multiple Consoles.

For example, if you have three separate instances of Consoles for development, test, and production environments, projects let you manage all of them from a single Central Console. With projects, one Console is designated as the master and all others are designated as supervisors. Thereafter, all UI and API requests for a project are proxied through the master and routed to the relevant supervisor. Supervisors do not serve a UI or API.

A single Central Console can support both scale and tenant projects simultaneously.
Connectivity

By default, the master and its supervisor Consoles communicate over port 8083. You can configure a different port by setting MANAGEMENT_PORT_HTTPS in twistlock.cfg at install time. All Consoles must
use the same value for MANAGEMENT_PORT_HTTPS. Communication between the master and supervisor Consoles must be direct, and cannot be routed through a proxy.

Defenders communicate with their respective supervisor Consoles. Project Defenders never communicate directly with the Central Console.

Prisma Cloud CA signed certs are used for establishing the Central Console to Supervisor Console communication link. Since no user interacts with the Supervisor Console directly, the link is an internal architectural detail, and we use our own CA. This setup reduces the risk of outages due to expired certs.

> **When configuring Central and Supervisor Consoles, you must configure the Supervisor Console to include the Subject Alternative Name (SAN) for the Central Console.**

> **When configuring access to the Consoles via Ingress Network Routes in Kubernetes, you must add the Central Console to the Supervisor Console Ingress configuration.**

Central Console can have its own set of Defenders. In this case, these Defenders do communicate directly with Central Console. However, no project Defenders ever communicate directly with Central Console.

**Multi-tenancy**

Tenant projects are like silos. They each have their own rules and settings that are created and maintained separately from all other projects.

**Infinite scale**

Each project can support a maximum of 5,000 Defenders. Scale projects can be stacked to support very large environments (greater than 5,000 hosts). For example, if your your container environment runs 10,000 hosts, then you would deploy 2 scale projects.

Data for each project is maintained with each supervisor. Data includes such things as audit event records and image scan reports. This division of data is what helps enable infinite horizontal scale. Even though two scale projects might share the same rules and settings, they will have their own audits, container scan reports, and so on.

**Inherited rules and settings for scale projects**

Scale projects inherit all rules and almost all configurations from the Central Console.

Rules and settings are distributed to supervisor Consoles by way of their REST APIs. All scale projects are updated in parallel. The Central Console queries the existing policies and settings in the supervisor Consoles. If the local policies and settings are different than the remote policies and settings, they are updated using the supervisor's REST API.

Rules and configurations are distributed to supervisor Consoles when:

- A supervisor Console becomes available (on connect or reconnect).
- A rule or setting changes in the Central Console.

For scale projects, requests for the settings pages and policies pages will return data from the master project.

Scale projects inherit almost all rules from Central Console:

- Defend > Firewalls > WAAS
- Defend > Runtime > [Container Policy, Host Policy]
- Defend > Vulnerabilities > Policy
- **Defend > Compliance > [Policy, Trusted Images]**

Scale projects inherit the following settings from Central Console:
- **Manage > Collections**
- **Manage > System > Scan** (Scan settings)
- **Manage > System > Proxy** (Proxy settings)
- **Manage > System > Logging**
- **Manage > System > Custom Feeds**
- **Manage > System > Forensics**
- **Manage > System > Intelligence**

The settings under **Defend > Vulnerabilities > [Registry, Serverless]** and **Manage > Alerts** are not shared between scale projects.

**Access control**

When users log into Prisma Cloud Console, they are presented with a list of projects to which they have access, and they can choose the project they want to work in. Access to projects is controlled by role-based access control rules.

You can grant access to specific projects for any 'local' users created in Console under **Manage > Authentication > Users**. If you have integrated Console with an OpenLDAP, Active Directory, or SAML provider, you can grant access to projects by group. Users and groups can be granted access to multiple projects.

A user's role is applied globally across all projects. That is, a user will have the same role for each project for which he has been granted access.

Rules and settings in scale projects can only be modified by users with access to Central Console. If you are granted access to a scale project, but not Central Console, you'll get an error when you try to create a new rule or change a setting. New rules and settings must be made in Central Console.

> *Project access control rules at the user level take precedence over access control granted at the group level. For example, if a 'local' user has been granted access to project1, but also belongs to group1, which has been granted access to project2, he will only have permissions to access project1.*

**Secrets**

Prisma Cloud fully supports secrets management for tenant projects. Secrets management can be independently configured and managed for each tenant project.

For scale projects, secrets management is supported in Central Console only. Secrets are not propagated from Central Console to any connected scale projects.

**Limitations**

Moving Defenders between projects is not supported. To “move” a Defender, decommission it one project and install it in another.

**Provisioning flow**

Let's look at how projects are provisioned.
**Step 1:** Install Console using any installation method. For example, you could install Console (onebox) with the `twistlock.sh` script or as a service in a Kubernetes cluster. When Console is installed, it runs in master mode by default.

Master mode

---

**Step 2:** Install a second Console on a different host. By default, it also runs in master mode.

Master mode

---

**Step 3:** In the UI for Console 1, provision a new project. Specify the URL to Console 2 and the project type (tenant or scale). The provisioning process automatically changes the operating mode for Console 2 to supervisor. The UI and API for Console 2 are now no longer directly accessible.
Step 4: The only difference between a master Console and a supervisor Console is whether its UI and API can be accessed directly, or whether it is proxied through the master. Assume you provisioned a tenant project in Step 3. To view your tenant project (managed by Console 2), open Console 1 and select the project. All your rules and settings for your project are loaded and displayed in Console 1.
You can release a supervisor, and return it to its original state by deleting the project. The supervisor Console reverts back to master mode.

**Migration strategies**

If you have already deployed one or more stand-alone Consoles, and you want to adopt a project-based structure, then the migration is easy. Simply designate one Console as master, then designate each remaining Console as a supervisor by provisioning projects for them.

Adding an existing Console to a project is not a destructive operation. All data is preserved, and the process can be reversed. The only thing that changes is the way you access Console when it is retooled as a supervisor. Supervisor Consoles cannot be accessed directly. They can only be accessed through the master Console, by selecting the project from **Selected project** drop down list.

For example, assume you’ve deployed three separate stand-alone Consoles: one for your production environment, one for your test environment, and one for your development environment.
When migrating to projects, you have the following options:

**Option 1:** Promote one Console to master, and designate the others as supervisors. In this example, you pick the prod Console to be master, then create tenant projects for the test and development Consoles. By default, Consoles run in master mode when they are installed, so you don’t need to do anything to “promote” prod to master. To relegate test and dev to supervisor, provision a project for each one.

**Option 2:** Install a new Console on a dedicated host and designate it as master. Provision a tenant project for each of the prod, test, and dev Consoles.
Accessing the API

All API requests should be routed to Central Console only. Central Console checks if the client has the correct permissions to access the given project, and then:

- For tenant projects, Central Console redirects the request to right supervisor, and then returns to supervisor’s response to the client.
- For scale projects, Central Console responds directly to requests for rules and settings. For other data, such as audits and scan reports, you must specify the project in the API request.

For API requests that create, modify, or delete data, Central Console responds to the client with a success return code, and then updates the supervisor asynchronously.

To target an API request to a specific project, append the `project=` query parameter to your request. For example, to get a list of Defenders deployed in the prod project:

```
GET http://<CENTRAL-CONSOLE>:8083/api/v1/defenders?project=prod
```

Central Console reroutes the request to the appropriate supervisor. Not all requests need to be rerouted. For example, the endpoints for getting a list of users, groups, or projects are all handled by Central Console directly. Some endpoints require no special permissions to access them, such as getting a list projects to which a user has been granted access.

Provisioning a project

Provision new projects from the Central Console UI.

> Communication between the master and supervisor Consoles must be direct, and cannot be routed through a proxy.

**STEP 1** Install a Console on a host in your environment using any install procedure.
There is no need to create an admin user or enter your license. Those details will be handled for you in the provisioning phase of this procedure.

**STEP 2** | Register the newly installed Console with the Central Console and create a project.

**STEP 3** | Go to Manage > Projects > Manage

**STEP 4** | Set Use Projects to On.

**STEP 5** | Click on the Provision tab.

**STEP 6** | Under Select Project type, choose Tenant or Scale.

**STEP 7** | In Project name, give your project a name.

**STEP 8** | In Supervisor address, enter the URL for accessing Console Include both the protocol (https://) and port.

**STEP 9** | For a fresh Console install, there is no need to enter any credentials. They will be created for you automatically.

If you are migrating an existing Console to a project, specify the admin credentials.

### Decommissioning a project

Decommissioning a project simply reverts the supervisor Console back to a stand-alone master Console. The link between Central Console and the former supervisor Console is severed. All project data (rules, audits, scan reports) is left intact.

When a project is created, the Console is configured with an admin user. When you delete the project, the admin credentials are shown to you so that you can continue to access and administer it. The credentials are shown only one time, so copy them and set them aside in a safe place.

**STEP 1** | Open Central Console.

**STEP 2** | Go to Manage > Projects > Manage.

**STEP 3** | In the Provisioned Projects table, click delete on the project you want to delete.

### Decommissioning disconnected projects

Central Console lets you delete projects, even if the supervisor Console is disconnected. The project is deleted from the master’s database, but it leaves the supervisor Console in the wrong state.

When you delete a disconnected project, Prisma Cloud tells you that the supervisor cannot be reached. To manually revert the supervisor Console back to a stand-alone master Console, call the supervisor’s REST API to change its settings.

**STEP 1** | Decide how you want to access the supervisor’s REST API. You can use basic auth or an auth token.

**STEP 2** | Update the supervisor’s project settings. The following example command uses basic auth. Only admin users are permitted to change project settings.

```
$ curl -k \
```
Deploying Defender DaemonSets for Projects (Console UI)

When creating a DaemonSet for a project, you can use the Console UI, twistcli, or Prisma Cloud API.

**STEP 1** | In Console, use the drop-down menu at the top right of the UI to select the project where you want to deploy your DaemonSet.

**STEP 2** | Go to Manage > Defenders > Deploy Daemon Set.

**STEP 3** | Configure the deployment parameters, then copy and run the resulting install script.

Deploying Defender DaemonSets for Projects (twistcli)

Create a DaemonSet deployment file with twistcli. Specify both the project name and the DNS name or IP address of the Supervisor Console to which the DaemonSet Defenders will connect. The DNS name or IP address must be a Subject Alternative Name in the Supervisor Console’s certificate.

```
$ <PLATFORM>/twistcli defender export kubernetes \
   --address https://<CENTRAL-CONSOLE>:8083 \
   --project <PROJECT-NAME> \
   --user <USER> \
   --cluster-address <SUPERVISOR-CONSOLE-SAN>
```

Deploying Defender DaemonSets for Projects (Prisma Cloud API)

A DaemonSet deployment file can also be created with the API. Specify both the project name and the DNS name or IP address of the Supervisor Console to which the DaemonSet Defenders will connect. The DNS name or IP address must be a Subject Alternative Name in the Supervisor Console’s certificate.

```
$ curl -k \
   -u <USER> \
   -X GET \
```

Limitations

A few features aren’t supported when Projects is enabled. Many of these will be fully enabled for Projects in an upcoming release.

**Secrets**

Prisma Cloud fully supports secrets management for tenant projects. Secrets management can be independently configured and managed for each tenant project.

For scale projects, secrets management is supported in Central Console only. Secrets are not propagated from Central Console to any connected scale projects.
Type-ahead support during rule creation

Auto-complete for resource targeting during rule creation is supported in tenant projects only. It’s not supported in scale projects. Only resources connected to the Central Console are supported with type-ahead functionality. The resources under scale projects are not visible for type-ahead.
Best practices for DNS and certificate management

As with most cloud-native software, Prisma Cloud relies on core infrastructure services, such as x509 cryptography and DNS name resolution. Defenders use these services to find and securely connect back to Console, and administrators use them to connect to Console and the API endpoints. When Console’s name can’t be resolved, or its certificate doesn’t include the name that Defenders use to connect to it, setup might fail and/or Defenders might not be able to successfully connect to Console.

In relatively simple environments, such as an on-premises environment with a flat network, Prisma Cloud can automatically discover and configure the appropriate network configuration during setup. However, in more complex environments, auto-discovery is difficult and administrators typically have to manually configure the appropriate settings.

Consider a deployment where Console exists in one cloud service, but protects hosts distributed across other cloud services in different regions. In this model, Console’s hostname is probably not resolvable by remote Defenders. And since Defenders probably do not connect directly to Console, but through some reverse NAT or a load balancer, the details of the underlying connectivity are probably obscured.

Map out your topology

Mapping out your topology is a fairly obvious step that is often overlooked, but it is the single best way to avoid connectivity problems.

First, document Console’s local hostname and IP. Try to determine whether this name is the actual name that Defenders will use to connect, or if there is another entity in between, such as a load balancer or reverse NAT service.

Then, map out all the potential connection paths from Defenders to Console. For example, there might be some Defenders deployed in the same cloud service as Console. They can connect to Console directly. Other Defenders might connect from another routed network or over the Internet using different names.

Documenting all of these paths and names at the beginning of the planning process saves significant time later when you’re troubleshooting. Use the following sample worksheet as a starting point:

| Console IP address:       |
| Console local host name:  |
| Console management port:  |
| Console / Defender communication port: |

| Load balancer / NAT IP address | Load balancer / NAT name: |
| Load balancer / NAT management port: |
| Load balancer / NAT Console / Defender communication port: |

| Defender to Console connection paths: |
| Direct? |
| From other cloud services in same deployment? |
| Over the Internet? |

Because naming is so critical to connectivity, you should use durable, Prisma Cloud-specific names for accessing Console. For example, although the default host name might be ip-10-1-27-12, it would be a poor choice because it’s tied to a specific hostname, which could change if you redeploy Console to a new host.

Instead, create a CNAME with a short TTL to reference this hostname, and use the CNAME for all name resolution. This way, if your hostname changes in the future, you simply need to remap the CNAME to
the new hostname. Using CNAMEs is preferable to directly mapping an A record because many cloud services automate DNS resolution within their fabric and offer limited options for overriding this behavior. In a complex, multi-network environment, the CNAME can be used to reference Console both from the local network and from other networks, including the Internet, through simple and well established DNS configurations.

Consider the following example scenario:

- Console runs in cloud network 1, with an IP of 10.1.27.12, and local hostname of ip-10-1-27-12.
- This IP can be accessed over the Internet through a load balancer.
- The load balancer’s IP is 100.4.1.8, with a name of lb1.cloudprovider.com.
- Some Defenders also run in cloud network 1.
- Other Defenders run in a data center in another region, and connect to Console over the Internet.

In this scenario, a good approach would be to create a CNAME, such as console.customer.com. Internet facing DNS servers would answer queries for Console with lb1.cloudprovider.com. Internal facing DNS servers would answer queries for Console with ip-10-1-27-12.

Implement the topology

After your naming scheme has been planned, the final step is implementing the names in Prisma Cloud.

When you deploy a Defender, you must specify how it connects to Console, with either an IP address or, preferably, a DNS name. The Prisma Cloud dashboard lets you specify these names, and provides some preconfigured names, in the Subject Alternative Names table on the Manage > Defenders > Deploy page. Any name in the table is added to Console’s certificate and becomes available as a configuration parameter in the Defender deployment pages.

Using our example scenario described in the previous section, the Subject Alternative Name table should contain the CNAME we chose (console.customer.com). If you have multiple names that you want to use to address Console, add them to the Subject Alternative Name table. For example, if Defenders in the same cloud network should access Console using cs1-console, you should have the following entries:

- console.customer.com
- cs1-console

After Prisma Cloud is set up with these values, you will see them in the drop down menu in all of the Defender deployment pages as a configuration parameter. When you set up a new Defender, select how it should connect to Console from the same list of names in the Subject Alternative Names table.
When you're installing Defender, always ensure that the name you select from the drop down list can be resolved from the host where Defender will run. Using our example scenario, this means that you would select cs1-console for 'local' hosts that run in the same cloud service as the Console, and that you would select console.customer.com for 'remote' hosts. If the name you select cannot be resolved from the host where you install Defender, Defender set up will fail.

### Updating the list of resolvable names for Console

Define additional names Defenders can use to connect to Console. After adding a name to the Subject Alternative Name table, the name is added to Console's certificate and it is available in the drop down list in the Defender deployment pages.

> The values for CONSOLE_CN and DEFENDER_CN in twistlock.cfg should never be modified unless you are directed to do so by Prisma Cloud Support. These values are needed to work around distribution-specific abnormalities in the hostname command, which we use to create certificates during set up. Your custom names should always go in the Subject Alternative Name table, and never be hard-coded into CONSOLE_CN or DEFENDER_CN.

**STEP 1** | In Console, go to Manage > Defenders > Deploy.

**STEP 2** | In the Subject Alternative Name table, click Add row.

**STEP 3** | Specify an IP address or fully qualified domain name.

**STEP 4** | Redeploy any Defenders that require the new name to connect to Console.

  If the old names are still accessible, this step can be skipped.
Prisma Cloud caps the size of some data collections to prevent misconfigured or noisy systems from consuming all available disk space and compromising the availability of the Console service.

The following limits are currently enforced in Console’s database. If you must retain all audits, consider configuring Console to send audits to syslog, and then forward audits to a log management system for long term storage.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Cap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jenkins plugin and twistcli scan reports</td>
<td>1000 scan reports or 100 MB (whichever limit is hit first)</td>
</tr>
<tr>
<td>Access audits</td>
<td>100K audits or 50 MB (whichever limit is hit first)</td>
</tr>
<tr>
<td>Kubernetes audits</td>
<td>100K audits or 50 MB (whichever limit is hit first)</td>
</tr>
<tr>
<td>Admission audits</td>
<td>100K audits or 50 MB (whichever limit is hit first)</td>
</tr>
<tr>
<td>Log inspection events</td>
<td>100K audits or 50 MB (whichever limit is hit first)</td>
</tr>
<tr>
<td>File integrity events</td>
<td>100K audits or 50 MB (whichever limit is hit first)</td>
</tr>
<tr>
<td>Host activities</td>
<td>100K audits or 50 MB (whichever limit is hit first)</td>
</tr>
<tr>
<td>Host history</td>
<td>100K audits or 50 MB (whichever limit is hit first)</td>
</tr>
</tbody>
</table>
The Prisma Cloud API enables automation and integration with other systems in your environment. Every control in the Console UI is exposed through the RESTful API. You can use the API for setup, configuration, monitoring, alerting, and data extraction.

The API reference can be found here.

- Access the API
- Set up a fresh Console
- Automate Defender install
- Manage compliance policies with the API
Access the API

The API reference can be found [here](#).

The Prisma Cloud API is exposed on the host that runs Console on port 8083 (HTTPS). The port is specified at install time in `twistlock.cfg`.

All example commands specify a variable called `COMPUTE_CONSOLE`, which represents the address for your Console. The address for your Console depends on how you installed it.

For Onebox installs, where you install Console on a stand-alone host, the value for `COMPUTE_CONSOLE` is the IP address or DNS name of the host. HTTPS access to Console is servered on port 8083, so the full address would be:

```
COMPUTE_CONSOLE = https://<IPADDR>:8083
```

For the default Kubernetes installation procedure, the Console service is exposed by a LoadBalancer, and so the address for `COMPUTE_CONSOLE` is

```
COMPUTE_CONSOLE = https://<LOAD_BALANCER>:8083
```

Access to the API requires authentication. You can either:

- Retrieve a token, then pass the token in the Authorization field of all subsequent requests.
- Use Basic HTTP authentication for each request.

The default install of Prisma Cloud Compute Edition uses self-signed certificates. By default, `curl` validates the server’s certificate. Because the certificate for the CA that signed the server’s cert isn’t in your CA store, `curl` can’t validate the server’s cert.

You’ve got two options:

- Pass the `--insecure` flag to `curl`. With this flag, validation that the server is who it claims to be is bypassed. The connection is still encrypted.
- Configure Prisma Cloud Compute to use your own custom certs.

Accessing the API using Basic authentication

The basic token is a Base64 encoded string of type username:password.

**STEP 1** | Generate the Base64 encoding of your username and password. Assume your username is api, and your password is api.

```
$ echo -n "api:api" | openssl base64
YXBpOmFwaQ==
```

**STEP 2** | To access any other endpoint, set the Authorization field of your HTTP header to Basic and add the encoded string. For example, to get all your runtime container policies:

```
$ curl --insecure \
-H 'Authorization: Basic YWRtaW46YWRtaW4=' \
https://<COMPUTE_CONSOLE>:8083/api/v1/policies/runtime/container
```

The `curl` command can handle basic auth for you with the `--user` option.
Accessing the API using token authentication

To access the API using a token:

**STEP 1 |** Retrieve a token from the api/v1/authenticate endpoint with your user credentials. Tokens are valid for 24 hours. You can also retrieve tokens using client certificates.

```
$ curl \
   -H "Content-Type: application/json" \ 
   -d '{"username":"admin", "password":"admin"}' \ 
   https://<COMPUTE_CONSOLE>:8083/api/v1/authenticate 
{
   "token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."
}
```

If you integrated Prisma Cloud Console with Active Directory, and you’re using the sAMAccountName user identifier, escape the backslash in the DOMAIN\sAMAccountName username value. For example:

```
$ curl \
   -H "Content-Type: application/json" \ 
   -d '{"username":"DOMAIN\admin", "password":"admin"}' \ 
   https://<COMPUTE_CONSOLE>:8083/api/v1/authenticate 
{
   "token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."
}
```

**STEP 2 |** Call the Prisma Cloud API, submitting the token in the Authorization field in the HTTP header of your request. For example, test connection to the API using the /api/v1/policies endpoint:

```
$ curl --insecure \
   -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..." \
   https://<COMPUTE_CONSOLE>:8083/api/v1/policies/runtime/container
```

Accessing the API using a client certificate

You can retrieve a token using client certificates issued by your public key infrastructure.

**Prerequisites:**

- You have configured Prisma Cloud Console with your server certificate. Go to Manage > Authentication > Certificates > TLS certificate for Console, and upload your certificate (cat the cert and private key into a single file).

**STEP 1 |** Install your client certificate on your local machine.

**STEP 2 |** Request a token using your client certificate.

```
$ curl --insecure \
   -X POST \ 
   --cert cert.pem \ 
   https://<COMPUTE_CONSOLE>:8083/api/v1/authenticate-client 
{
   "token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."
}
```
STEP 3 | Call the Prisma Cloud API, submitting the token in the Authorization field in the HTTP header of your request. For example, to get all policies:

```
$ curl --insecure \
-H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..." \
https://<COMPUTE_CONSOLE>:8083/api/v1/policies/runtime/container
```
Set up a fresh Console

After first installing Prisma Cloud Console, you must create an initial admin user and set up your license. The Prisma Cloud API provides endpoints to complete the set up of a fresh Console install.

Creating your first admin user

After Console is first installed, you must create the first admin user. To do this, use the `/api/v1/signup` endpoint.

The following example curl command creates the initial admin user named butterbean.

```
$ curl -k 
-H 'Content-Type: application/json' 
-X POST 
-d '{"username": "butterbean", "password": "<PASSWORD>"}' 
https://<CONSOLE>:8083/api/v1/signup
```

The signup process can only be executed once, whether from the Console UI or the API. Calling this endpoint after the initial sign up has been completed results in a 400 error response.

Setting up your license

Console isn’t functional until you provide your license key. The Prisma Cloud API provides an endpoint for setting up your license.

In this procedure, you access the Prisma Cloud API using an auth token. You could also access the API using basic authentication. For more information, see Access the API.

Prisma Cloud provides a single license that protects a specific number of nodes. The number of nodes covered depends on your subscription. You can use the same license to install multiple instances of Console. There is need to get a new license when building out new environments with Prisma Cloud.

For example, if you have licensed 100 nodes and you have deployed to 10 separate tenants, each with its own Console, use the same license key for each instance of Console.

STEP 1 | Get an auth token.

```
$ curl -H "Content-Type: application/json" 
-d '{"username":"admin", "password":"admin"}' 
https://localhost:8083/api/v1/authenticate
```

STEP 2 | Set environment variables for your auth token and license key.

```
$ echo $LICENSE_KEY
{"key":"your license key here"}

$ echo $TOKEN
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...
```
STEP 3 | Execute the command referencing these vars to set the license.

```
$ curl -H "Authorization: Bearer $TOKEN" \\
-H "Content-Type: application/json" \\
-d $LICENSE_KEY \\
https://localhost:8083/api/v1/settings/license
```

The result should be 200 OK with an empty body "[]".
Automate Defender install

You can automate the deployment of Defenders using the Prisma Cloud API.

To steps to deploy Defender using the Prisma Cloud API are:

1. Authenticate with Console.
2. Retrieve the Defender install script from Console.
3. Execute the install script on the host where you want Defender to run.
4. Validate that Defender has been installed and it is running.

You should create a service account for the system that will install Defender. The service account should have the role of Defender Manager, or higher, and you should use a long, random string for the password.

This procedure shows you how to access the API using an auth token, which is retrieved from the `/api/v1/authenticate` endpoint. By default, tokens expire after 30 minutes, but you can configure Console to issue long-lived tokens. The maximum validity period that can be configured for long-lived tokens is 71580 minutes (49.7 days). If these validity periods are too short, then access the Prisma Cloud API using basic authentication.

When you’re installing Defender into a Kubernetes, OpenShift, or Docker Swarm cluster, use the deployment method documented in each respective install guide as a reference. Prisma Cloud supports these setups using orchestrator-native constructs, such as DaemonSet (for Kubernetes and OpenShift) and global services (for Docker Swarm), to automate Defender deployment. Deployment files can be generated from the Console web interface, the `twistcli` utility, and the Prisma Cloud API.

Install a stand-alone Defender

Single Container Defenders are installed on hosts that are not part of a cluster.

**Prerequisites:**

- Console has been installed and deployed on a host in your environment.
- Console is reachable over the network from the host where you want to install Defender.
- You have created a service account with the role of Defender Manager, or higher.

**STEP 1 |** Validate that the node where you will install Defender can reach Console over the network.

```bash
$ curl -k https://<COMPUTE_CONSOLE>:8083/api/v1/_ping
```

**STEP 2 |** Retrieve an auth token from Console.

```bash
$ curl -H "Content-Type: application/json" \
-d '{"username":"<USERNAME>", "password":"<PASSWORD>"}' \
https://<COMPUTE_CONSOLE>:8083/api/v1/authenticate
```

Where:

- `-H` -- Extra header in the HTTP request, specifying that the message content is JSON.
- `-d` -- Data to be sent in the POST request.
- `<USERNAME>` --
User with the role of Defender Manager or higher.

- `<PASSWORD>` --
  User’s password.

**STEP 3** Validate that you are authorized to access the Prisma Cloud API. Use the token you just retrieved to get a list of deployed Defenders.

```
$ curl \
  -H "authorization: Bearer <TOKEN>" \
  https://<COMPUTE_CONSOLE>:8083/api/v1/defenders
```

**STEP 4** Download and run the Defender install script.

```
$ curl \
  -H "authorization: Bearer <TOKEN>" \
  https://<COMPUTE_CONSOLE>:8083/api/v1/scripts/defender.sh \
  -o defender.sh && \
  chmod a+x defender.sh && \
  sudo ./defender.sh -c "<CONSOLE>" -d "none"
```

To see what options can be passed to the install script, go the Defender deployment page in Console at Manage > Defenders > Deploy, play with the various configuration settings, and observe the changes to the resulting curl-bash script.

**Install a stand-alone Defender (no Docker)**

Prisma Cloud Defender can be installed on hosts that do not run Docker. In this setup, Defender is installed as a service rather than a container. After downloading the Defender install script from the API, run it with the mandatory `--install-host` option.

Other options that can be passed to the install script include:

- `--install-folder` --
  Specifies the install folder. By default, it is `/opt/twistlock`.

- `--install-data-folder` --
  Specifies the data folder. By default, it is `/var/lib/twistlock`.

- `--ws-port` --
  Specifies the websocket communication port. By default, it is 8084.

**STEP 1** Retrieve an auth token from Console, where `<USER>` must have the role of Defender Manager, or higher.

```
$ curl -H "Content-Type: application/json" \
  -d '{"username":"<USERNAME>", "password":"<PASSWORD>"}' \
  https://<COMPUTE_CONSOLE>:8083/api/v1/authenticate
```

**STEP 2** Download and run the Defender install script with the `--install-host` option.

```
$ curl \
  -H "authorization: Bearer <TOKEN>" \
  https://<COMPUTE_CONSOLE>:8083/api/v1/scripts/defender.sh \
  -o defender.sh && \
  chmod a+x defender.sh && \
```
Install a Defender DaemonSet

You can deploy a Defender DaemonSet to your Kubernetes or OpenShift cluster using the Prisma Cloud API. Use the `/api/v1/defenders/daemonset.yaml` API endpoint to create a DaemonSet configuration file for Defender. This endpoint accepts a number of parameters to customize the configuration file for your environment.

Defender requires a handful of kernel capabilities to function. When AppArmor is enabled in your cluster, which is the case when nodes run Ubuntu, Defenders must run as privileged to acquire those capabilities. With AppArmor, pods are started with the default runtime profile, `runtime/default`, unless a specific profile is explicitly specified. The YAML file generated by the Prisma Cloud tools (twistcli, API) does not explicitly set a profile, but sets privileged to true. When privileged is true, Defender runs unconfined (no profile).

A deployment consists of the following steps:

- Delete any old Defender deployment configurations.
- Retrieve a daemon set configuration from the Prisma Cloud API.
- Create the Defender daemon set.

**STEP 1** (Optional) Delete any old Kubernetes/OpenShift configurations:

Before installing or upgrading, any existing Prisma Cloud DaemonSet components must be removed. They will be re-installed again right after this step. The following items should be deleted from the namespace they were created in:

- The `.twistlock` directory in the current path, which is a remnant of previous Prisma Cloud installations.
-DaemonSet: twistlock-defender-ds
-ServiceAccount: twistlock-service
-Secrets: twistlock-secrets
-Security Context Constraints (OpenShift only): twistlock-scc
-Namespace: The default namespace is twistlock, but it can be overridden by the user at install time.

**STEP 2** Retrieve a Defender DaemonSet configuration file, setting the appropriate parameters for your environment.

The following call generates the same YAML file as the default twistcli invocation for Kubernetes:

```
$ curl -k \
-u <USER> \
'https://<COMPUTE_CONSOLE>:8083/api/v1/defenders/daemonset.yaml?consoleaddr=<COMPUTE_CONSOLE>&namespace=twistlock&orchestration=kubernetes&privileged=true' \
> defender.yaml
```

The following command generates the same YAML file as the default twistcli invocation for OpenShift:

```
$ curl -k \
-u <USER> \
'https://<COMPUTE_CONSOLE>:8083/api/v1/defenders/daemonset.yaml?consoleaddr=<COMPUTE_CONSOLE>&namespace=twistlock&orchestration=openshift' \
> defender.yaml
```
STEP 3 | Create the DaemonSet.

For Kubernetes:

```
kubectl create -f defender.yaml
```

For OpenShift:

```
oc create -f defender.yaml
```

For Google Cloud deployments, you might not have access to the cluster’s master node. In this case, use `kubectl proxy`. Click the **Connect** button beside your cluster, gives you a command like this:

```
$ gcloud container clusters get-credentials aqsa-test \
   --zone us-central1-a --project twistlock
```

Then run:

```
$ kubectl proxy
```
Manage compliance policies with the API

The Prisma Cloud API can be used to automate all aspects of the product. In this article, we discuss using it to automate management of compliance policies.

For example, you may want to create different compliances for different parts of your environment, but base them all from a common baseline. Alternatively, you may have a central security or internal audit team that needs to be able to assess compliance across multiple environments. In both cases, the API helps ‘fan out’ management of compliance policies.

Creating and editing policies

Prisma Cloud uses a single API (a single object) to update all compliance policies at once. This makes it easy to keep a strict order between the policies. The process of adding, editing, or removing a policy is the same:

1. Get all policies (returns JSON).
2. Modify the JSON output according to your needs.
3. Update policies by pushing the new JSON payload.

**STEP 1** | Get all policies

```bash
$ curl -k -u admin:admin https://127.0.0.1:8083/api/v1/policies/compliance
```

**STEP 2** | Modify the JSON output, explanatory comments provided after //

```json
{
  "effect": "alert", // See 1
  "action": ["*"], // See 1
  "resource": ["container:*", "image:imagename1*", "image:imagename2*", "host:hostname*", "label:*"], // See 2
  "condition": {
    "vulnerabilities": [{ // See 3
      "id": 510, // See 4
      "block": true // See 5
    }, {
      "id": 511, // See 6
      "block": false // See 7
    }]
  },
  "principal": [], // See 8
  "group": ["*"], // See 8
  "namespace": ["*"], // See 8
  "trust_group": [], // See 8
  "last_modified": "Mon, 11 Jul 2016 05:25:58 GMT",
  "owner": "admin", // See 9
  "name": "dima-2" // See 10
}
```

Where:

1 — Obsolete.
2 — You can specify multiple resources of the same type.
3 — Compliance rules are called vulnerabilities as well.
4 — Compliance rule ID.
5—Block on compliance rule 510. If vuln is unspecified, the effect is ignore.
6—Compliance rule ID.
7—Alert on compliance rule 511.
8—Not relevant
9—User who created the rule.
10—Name of the rule.

If any policy is modified or deleted here, this will replace the existing policies in Console when uploaded. For any policy to remain unchanged / undeleted, please keep it as is in the file.

STEP 3 | Update policies, where policy_upload.txt contains the JSON payload.

```bash
$ curl -k -u admin:admin -X PUT --data-binary "@policy_upload.txt"
```

You can find a sample policy here.

Getting compliance results

Compliance results can also be retrieved via the API.

For example, to get the compliance status of containers:

```bash
$ curl -k -u admin:admin https://127.0.0.1:8083/api/v1/health/containers
```

Which returns:

```json
"compliance_vulnerabilities": [{
  "text": "",
  "id": 599,
  "severity": "high",
  "cvss": 0,
  "cve": "",
  "description": "Container is running as root",
  "type": "container",
  "package_name": "",
  "package_version": "",
  "extension": null,
  "_id": "5783349a8485601100a33f3b"
}, {
  "text": "",
  "id": 51,
  "severity": "high",
  "cvss": 0,
  "cve": "",
  "description": "Verify AppArmor Profile, if applicable (CIS 5.1)",
  "type": "container",
  "package_name": "",
  "package_version": "",
  "extension": null,
  "_id": "5783349a8485601100a33f3a"
}, ...
```
To filter results, use the id parameter:

```
$ curl -k -u admin:admin "https://127.0.0.1:8083/api/v1/health/containers?id=4cba*"
```

Or, use the name parameter (offset and limit must also be included):

```
$ curl -k -u admin:admin "https://127.0.0.1:8083/api/v1/health/containers?offset=0&limit=10&search=distracted_yona"
```

Use the "scan_time" property of the scan result to determine if the scan is completed.

```
$ curl -k -u admin:admin "https://127.0.0.1:8083/api/v1/health/containers?offset=0&limit=10&search=distracted_yona"
```

Which returns:

```
[
  {
    "_id": "577dee9a8485601100a332a5",
    "scan_time": "2016-07-11T08:09:01.541Z",
    "hostname": "dima-debian-8-demo.c.dima-rack.internal",
    "info": {
      "name": "distracted_yona",
      "id": "4cba09825cdad40d0cc68ff6426bb2573e20f266c18aa61a0c5d4fede0f5d2a",
      "image_id": "sha256:54f193f02c4cf84ed03fd98707d26c7bd1d1e19c25003eaefcc54b0c97f7d582",
      "image": "docker.io/dimastopel/testsyscalls5:latest",
      "app": "node server.js",
      "network_settings": {
        "SandboxKey": "/var/run/docker/netns/2417b45657d8",
        "EndpointID": "51c2f850148666f2ac912752f6c62b791afae3003464702c02a91509f7a182df"
      }
    }
  }
]
```

The scan request returns the time the scan started (see below). Repeat the request until the scan time of the result is greater than the one you received from scan request. We recommend checking every 10 seconds; few scans should take more than 20-30 seconds to complete.

```
$ curl -k -u admin:admin -X POST https://127.0.0.1:8083/api/v1/health/containers/scan
{"scanTime": "2016-07-11T08:12:29.228Z"}
```
This section contains guides for deploying various advanced setups.

- Configure an AWS Classic Load Balancer for ECS
- Configure Prisma Cloud to use Nginx Ingress Controller in Kubernetes
- Configure the load balancer type for AWS EKS
- Deploy Defenders outside an OpenShift Cluster
- Configure Prisma Cloud Console's listening ports
- Provision tenant projects in OpenShift
- Configure Prisma Cloud to use Istio Ingress Gateway
- Disable automatic learning
- Performing a rolling upgrade of Defenders
- Debug data
Configure an AWS Classic Load Balancer for ECS

This guide shows you how to configure a Classic Load Balancer in AWS for Prisma Cloud Console. Console serves its UI and API over HTTPS on port 8083, and Defender communicates with Console over a websocket on port 8084. You’ll set up a single load balancer to forward requests for both port 8083 and 8084 to Console, with the load balancer checking Console’s health using the /api/v1/_ping endpoint on port 8083.

**STEP 1 |** Log into the AWS Management Console.

**STEP 2 |** Go to **Services > Compute > EC2**.

**STEP 3 |** In the left menu, go to **LOAD BALANCING > Load Balancers**.

**STEP 4 |** Create a load balancer.

1. Click **Create Load Balancer**.
2. In **Classic Load Balancer**, click **Create**.
3. Give your load balancer a name, such as **pc-ecs-lb**.
4. Leave default **VPC**.
5. Create the following listener configuration:
   - **Load Balancer Protocol**: TCP
   - **Load Balancer Port**: 8083
   - **Instance Protocol**: TCP
   - **Instance Port**: 8083
6. Click **Add** to add another listener using following listener configuration:
   - **Load Balancer Protocol**: TCP
   - **Load Balancer Port**: 8084
   - **Instance Protocol**: TCP
   - **Instance Port**: 8084
7. Click **Next: Assign Security Groups**.
   - Select the **pc-security-group**
8. Click **Next Configure Security Settings**.
   - ignore the warning and click **Next: Configure Health Check**
9. Use the following health check configuration:
   - **Ping Protocol**: HTTPS
   - **Ping Port**: 8083
   - **Ping Path**: /api/v1/_ping
   - Use default settings for **Advanced Details**
10. Click **Next: Add EC2 Instances**
    - Do not select any instance
11. Click **Next: Add Tags**.
    - Under **Key**, enter **Name**.
    - Under **Value**, enter **pc-ecs-lb**.
12. Click **Review and Create**.
13. Review your settings and select Create.
14. Review created load balancer and record its DNS Name

For the complete install procedure on Amazon ECS environment, follow the steps in Install Prisma Cloud on Amazon ECS.
Configure Prisma Cloud to use Nginx Ingress Controller in Kubernetes

This guide shows you how to configure the Nginx ingress controller for Prisma Cloud Console communications. Console serves its UI and API on ports 8081 (HTTP) and 8083 (HTTPS) according to its configuration. You'll set up an ingress controller to forward requests to port 8081 on the backend Console (notice that by default the Console serves only on HTTPS port. You can configure an HTTP port in twistlock.cfg before installing the Console).

Note that the Console and Defender communications that occur over port 8084 are not included in this example. There are properties of those connections that are not well supported by the Nginx Ingress.

Prerequisites:
- You have a Kubernetes cluster and you can create LoadBalancers from YAML configuration files.
- You have enabled Ingress on your cluster.

Configuration

Your controller should point to / on your twistlock-console service.

In the sample ingress.yml below, you create an ingress definition for "console-yourconsole.example.com". Both of these endpoints should point to your twistlock-console service created by the DaemonSet definition you generated during your Kubernetes install.

```yaml
--- incomplete sample ingress definitions---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: console-ingress
  namespace: twistlock
  annotations:
    kubernetes.io/ingress.class: "nginx"
# your other ingress annotations
spec:
rules:
- host: console-yourconsole.example.com
  http:
    paths:
    - path: /
      backend:
        serviceName: twistlock-console
        servicePort: 8081
  tls:
    - secretName: console-cert
      hosts:
      - console-yourconsole.example.com
```

For the complete install procedure for Kubernetes, see Installing Prisma Cloud on Kubernetes.
Configure the load balancer type for AWS EKS

When installing Prisma Cloud on AWS EKS, the deployment creates an AWS Classic Load Balancer (ELB) by default, and Prisma Cloud Console is accessed through the ELB. The ELB is internet-facing, with a security group that serves ports 8081 and 8083 to the internet. In many cases, this is not ideal, because anyone on the internet with the load balancer's DNS name can access Console's login page.

Starting with version 1.9.0, Kubernetes supports the AWS Network Load Balancer (NLB). Unlike ELBs, NLBs forward the client's IP through to the node. You can leverage this property to restrict which IPs can access the NLB by setting `.spec.loadBalancerSourceRanges` in your deployment file. If `.spec.loadBalancerSourceRanges` is not set, Kubernetes allows traffic from 0.0.0.0/0 to the Node Security Group(s). If nodes have public IP addresses, be aware that non-NLB traffic can also reach all instances in those modified security groups.

If you utilize a mixed environment, it is sometimes necessary to route traffic from services inside the same VPC. In a split-horizon DNS environment, you would need two services to be able to route both external and internal traffic to your endpoints. This is where an internal load balancer would be useful, allowing more restrictive settings to be applied to the load balancer created by the Prisma Cloud Console deployment.

This guide shows you how to change the configuration of your load balancer. It is controlled by annotations added to your Prisma Cloud Console service deployment file.

For more information about Load Balancing in EKS, see the EKS Load Balancing user guide

Provision a Network Load Balancer

Serve Prisma Cloud Console through a Network Load Balancer.

**Prerequisites:**

- You have already created an EKS cluster.
- You have `twistlock_console.yaml` in your current working directory. This deployment file is generated with the `twistcli` tool.

**STEP 1** | Open `twistlock_console.yaml` for editing.

**STEP 2** | Add the following annotations to the Service:

```
annotations: service.beta.kubernetes.io/aws-load-balancer-type: nlb
```

**STEP 3** | (Optional) To limit which client IP's can access the Network Load Balancer, specify the following:

```
spec:
  loadBalancerSourceRanges:
  - "143.231.0.0/16"
```

The resulting Service YAML in `twistlock_console.yaml` should look like this:

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    name: console
    name: twistlock-console
```
namespace: twistlock
annotations: service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
spec:
  ports:
  - name: communication-port
    port: 8084
  - name: management-port-https
    port: 8083
  - name: mgmt-http
    port: 8081
loadBalancerSourceRanges:
  - "143.231.0.0/16"
selector:
  name: twistlock-console
  name: twistlock-console
type: LoadBalancer

---

STEP 4 | Deploy Prisma Cloud Console.

$ kubectl create -f twistlock_console.yaml

Prisma Cloud Console is served through a Network Load Balancer.

Provision an internal load balancer

Serve Console through an internal load balancer.

For the complete Kubernetes install procedure, see Installing Prisma Cloud on Kubernetes.

For internal load balancers, your Amazon EKS cluster must be configured to use at least one private subnet in your VPC. Kubernetes examines the route table for your subnets to identify whether they are public or private. Public subnets have a route directly to the internet using an internet gateway, but private subnets do not.

Prerequisites:

- You have already created an EKS cluster.
- You have twistlock_console.yaml in your current working directory. This deployment file is generated with the twistcli tool.

STEP 1 | Open twistlock_console.yaml for editing

STEP 2 | Add the following annotations to the Service.

annotations: service.beta.kubernetes.io/aws-load-balancer-internal: 0.0.0.0/0

The resulting Service YAML in twistlock_console.yaml should look like this:

---
apiVersion: v1
kind: Service
metadata:
  labels:
    name: console
    name: twistlock-console
  namespace: twistlock
STEP 3 | Deploy Prisma Cloud Console.

$ kubectl create -f twistlock_console.yaml

Prisma Cloud Console is served through an internal Load Balancer.
Deploy Defenders outside an OpenShift Cluster

This guide demonstrates how to deploy Prisma Cloud Defenders outside the OpenShift cluster where Prisma Cloud Console is running. You need to expose the Prisma Cloud-Console service's TCP ports 8083 and 8084 as external OpenShift routes. Each route will be an unique, fully qualified domain name. In this example, you deploy Prisma Cloud Defenders as a DaemonSet in a second OpenShift cluster and to a Windows Server 2016 with Containers node. Prisma Cloud API calls are made to the Prisma Cloud-Console external OSE router https://console1.apps.jonathan.lab.twistlock.com The Prisma Cloud Defenders will communicate to the Console via wss://defenders.apps.jonathan.lab.twistlock.com:443

Prerequisites:

- The Prisma Cloud Console is fully operational, Prisma Cloud OpenShift Deployment guide
- An existing OpenShift external route to the Prisma Cloud-Console's TCP port 8083 (Prisma Cloud UI and API)

OpenShift and Prisma Cloud Console configuration

All commands are run from a system that is external to the OpenShift Cluster using the oc and twistcli commands.

STEP 1 | Log into the OpenShift Cluster running the Prisma Cloud Console.

STEP 2 | Go to Prisma Cloud Project > Applications > Routes.

STEP 3 | Create New Route.

1. Name: twistlock-defender.
3. Target Port: 8084 → 8084.
4. Security:
   - TLS Termination: Passthrough.
   - Insecure Traffic: Redirect.
Create Route

Routing is a way to make your application publicly visible.

* Name

    twistlock-defender

A unique name for the route within the project.

Hostname

    defenders.apps.jonathan.lab.twistlock.com

Public hostname for the route. If not specified, a hostname is generated.

Path

    /

Path that the router watches to route traffic to the service.

* Service

    twistlock-console

Service to route to.

Target Port

    8084 → 8084 (TCP)

Target port for traffic.

Security

    Secure route

Routes can be secured using several TLS termination types for serving certificates.

TLS Termination

    Passthrough

    Learn More

Insecure Traffic

    Redirect

Policy for traffic on insecure schemes like HTTP.

Certificates

    TLS certificates for edge and re-encrypt termination. If not specified, the router's default certificate is used.
STEP 4 | Add the new route to the Prisma Cloud Console’s SubjectAlternativeName.
   1. In the Prisma Cloud Console go to Manage > Defenders > Names.
   2. Click Add SAN.
   3. Add the new route FQDN defenders.apps.jonathan.lab.twistlock.com.

Deploy Prisma Cloud Defender Daemonset in Second OpenShift Cluster

Using the twistcli tool generate the Prisma Cloud Defender defender.yaml file.

STEP 1 | Run the command:

```
$ linux/twistcli defender export openshift \
   --address https://console1.apps.jonathan.lab.twistlock.com \
   --cluster-address defenders.apps.jonathan.lab.twistlock.com \
   --namespace twistlock \
   --selinux-enabled
```

STEP 2 | Edit the resulting defender.yaml and change:

```
- name: WS_ADDRESS
  value: wss://defenders.apps.jonathan.lab.twistlock.com:8084
```
STEP 3 | **oc login** to the OpenShift Cluster you will be deploying the Prisma Cloud Defenders to.

STEP 4 | Create the Prisma Cloud Project **oc new-project twistlock**.

STEP 5 | Deploy the Tiwslck Defender daemonset **oc create -f ./defender.yaml**.

STEP 6 | The Defenders in the second OpenShift Cluster will appear in the Prisma Cloud Console's Manage > Defenders > Manage.
STEP 3 | Copy the powershell script that is generated in 1.c.

STEP 4 | Modify the following in the script:
1. Remove ":8083" from the 
   -Uri.

   ```powershell
   -Uri "https://console1.apps.jonathan.lab.twistlock.com/api/v1/scripts/defender.ps1"
   ```

2. Change the -consoleCN to the twistlock-defender FQDN and add the -wsPort 443 variable.

   ```powershell
   -consoleCN defenders.apps.jonathan.lab.twistlock.com -wsPort 443
   ```

STEP 5 | The resulting script looks similar to the following:

```powershell
```
console1.apps.jonathan.lab.twistlock.com/api/v1/scripts/defender.ps1" -Headers @{"authorization" = "Bearer <token>" } -OutFile defender.ps1; .\defender.ps1 -type dockerWindows -consoleCN defenders.apps.jonathan.lab.twistlock.com -wsPort 443 -install

**STEP 6** | On the Windows Server node, run the script in a Powershell x64 shell.

**STEP 7** | The Windows Prisma Cloud Defender will appear in Manage > Defenders > Manage.
Configure Prisma Cloud Console’s listening ports

This guide shows you how to configure Prisma Cloud to listen on different ports. Typically this type of configuration is made at the load balancer layer, but it can be done directly with Prisma Cloud.

By default Prisma Cloud listens on:

• 8083 HTTPS management port for access to Console.
• 8084 WSS port for Defender to Console communication.

For more information, see the https://cdn.twistlock.com/docs/downloads/Prisma Cloud-Reference-Architecture.pdf[Reference Architecture].

If you are setting the port below 1024 then Prisma Cloud needs permission to access this privileged port. You must also set RUN_CONSOLE_AS_ROOT=${RUN_CONSOLE_AS_ROOT:-false} to true.

Prerequisite: You have downloaded and unpacked the Prisma Cloud software.

STEP 1 | Go to the directory where you unpacked the Prisma Cloud software.

STEP 2 | Open twistlock.cfg for editing.

- MANAGEMENT_PORT_HTTP sets the HTTP access port, leaving this blank disables HTTP access.
  
  Example: MANAGEMENT_PORT_HTTP=${MANAGEMENT_PORT_HTTP-80} configures Console to listen on port 80.

- MANAGEMENT_PORT_HTTPS sets the HTTPS access port.
  
  Example: MANAGEMENT_PORT_HTTPS=443 configures Console to listen on port 443.

- COMMUNICATION_PORT sets the WSS port used for Defender to Console communication.
  
  Example: COMMUNICATION_PORT=9090 configures Console to listen on port 9090.

STEP 3 | Run twistlock.sh to install Prisma Cloud Console with your settings.

If you are setting the port below 1024 then Prisma Cloud needs permission to access this privileged port. You must also set RUN_CONSOLE_AS_ROOT=${RUN_CONSOLE_AS_ROOT:-false} to true.
Provision tenant projects in OpenShift

This guide shows you how to set up tenant projects on OpenShift clusters. If you try to provision tenant projects using the normal provisioning flow, Central Console cannot reach the host where Supervisor Console runs. Failing to follow these steps can lead to an 'Internal Server Error', even when everything seems to be set up properly.

In this example provisioning flow, the DNS names for Central Console and Supervisor Console are:

- Central Console — https://console.apps.jonathan.lab.twistlock.com
- Supervisor Console to be provisioned — https://console.39apps.jonathan.lab.twistlock.com

Prerequisites:

- Two fully operational Prisma Cloud Consoles are already deployed. For more information, see Prisma Cloud OpenShift Deployment guide
- OpenShift external routes to both Consoles' TCP port 8083 (Prisma Cloud UI and API), with the TLS termination type set to pass through, already exist.
- The to-be Central and Supervisor Consoles are already licensed and you've created initial admin users.

**STEP 1 |** Designate one Console to be Supervisor and the other to be Central.
**STEP 2** | Log into the Supervisor Console with your admin user.

**STEP 3** | Add the FQDN of the Supervisor Console to the Subject Alternative Name field of the Supervisor Console’s certificate.

1. In the Supervisor Console, go to **Manage > Defenders > Names**.
2. Click **Add SAN**.
3. Add the Supervisor Console’s FQDN. In this example, it is `console.39apps.jonathan.lab.twistlock.com`.
4. Click **Add**.

**STEP 4** | Log into the Central Console with your admin user.

**STEP 5** | Enable Projects by going to **Manage > Projects > Manage** and setting **Use Projects** to **On**.

**STEP 6** | Click the **Provision** tab and to provision a tenant Console.

1. Under **Select Project type**, choose **Tenant**.
2. In **Project name**, give your project a name.
3. In **Supervisor address**, add the FQDN of the Supervisor. In this example, it is `https://console.39apps.jonathan.lab.twistlock.com`.
4. Add the **Admin credentials for Supervisor**.
5. Click **Provision**.

Your Supervisor Console should be successfully provisioned.
<table>
<thead>
<tr>
<th>Type</th>
<th>Supervisor</th>
<th>Connected Defenders</th>
<th>Date Created</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central Console</td>
<td>console.apps.jonathan.lab.twistlo</td>
<td>0 / 0</td>
<td></td>
</tr>
<tr>
<td>Tenant</td>
<td>console.39apps.jonathan.lab.twis</td>
<td>4 / 4</td>
<td>Jan 31, 2019</td>
</tr>
</tbody>
</table>
Configure Prisma Cloud to use Istio Ingress Gateway

Ingress is a Kubernetes capability that lets you expose your services to HTTP, HTTPS, and other types of traffic outside the cluster. When configured correctly, the Ingress endpoint handles all routing from external clients to your Kubernetes services. In the setup described here, the Istio Ingress controller forwards external traffic to the Prisma Cloud Console, for both HTTPS connections and Defender traffic.

**Ingress versus LoadBalancers**

An Ingress offers more options than LoadBalancers. Extended capabilities for SSL, authentication, and routing are available with Ingress resources, but not LoadBalancers. Ingress can help with cost management in cloud environments. A single Ingress Controller can reduce the number of LoadBalancers you need to provision and can instead share one Ingress endpoint. Additionally, there are lots of integrations with automated certificate management tools that can help you manage certificates for your services.

**Istio versus other Ingress Controllers**

Istio provides lots of flexibility around how your deployed services communicate. Using sidecars to create a service mesh enables capabilities at the network layer that can be useful for advanced routing. This can be especially true if you want to deploy services across multiple clusters, or increase security between services with mutual TLS. Istio’s traffic management features lets you set up circuit breakers and A/B or canary testing workflows, that dynamically route traffic between various deployed versions of your software.

If you have started adopting Istio, and wish to use it as the main Ingress point for your services, this guide helps you expose your Prisma Cloud installation using Istio.

**Setting up Istio**

This example is built on a self-managed Kubernetes cluster running on Google Cloud Platform using Istio v1.1. This should work on any Istio environment, as long as ports are properly configured in the istio-ingressgateway.

*MARCH 6, 2019 - GKE uses Istio v1.03, and there may be a potential issue with TCP routing. For more information, see https://github.com/istio/istio/issues/6574.*

The following diagram shows the components in the solution:
Setting up sidecar injection

Set up Istio sidecar injection for the `twistlock` namespace before deploying Prisma Cloud Console.

**STEP 1 | Create the twistlock namespace.**

```
$ kubectl create namespace twistlock
```

**STEP 2 | Enable sidecar injection for the `twistlock` namespace.**

```
$ kubectl label namespace twistlock istio-injection=enabled
```

**STEP 3 | Validate the setup.**

```
$ kubectl get namespace -L istio-injection
NAME           STATUS   AGE   ISTIO-INJECTION
default        Active   13m   
istio-system   Active   13m   disabled
kube-public    Active   13m
kube-system    Active   13m
```

Installing Prisma Cloud Console

Generate the YAML configuration file for Console, then deploy it.

**STEP 1 | Generate the Prisma Cloud Console YAML. You will see an error that says the namespace already exists, but you can safely ignore it.**

```
$ <platform>/twistcli console export kubernetes
```

**STEP 2 | Deploy Console.**

```
$ kubectl create -f twistlock_console.yaml -n twistlock
configmap/twistlock-console created
service/twistlock-console created
persistentvolumeclaim/twistlock-console created
serviceaccount/twistlock-console created
replicationcontroller/twistlock-console created
```
Error from server (AlreadyExists): error when creating "twistlock_console.yaml": namespaces "twistlock" already exists

STEP 3 | Validate your setup.

You should see two containers in the Prisma Cloud Console pod. This indicates that you have successfully deployed both the Prisma Cloud Console and the Istio sidecar.

$ kubectl get pods -n twistlock

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>twistlock-console-6fdsx</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>5m</td>
</tr>
</tbody>
</table>

Egress Controller for Prisma Cloud Intelligence Stream

Prisma Cloud Console connects to https://intelligence.twistlock.com (35.238.214.241) with a secure web socket to download updated threat data. In the YAML for the Prisma Cloud-Console replicationController, add the following Istio egress annotation.

```yaml
spec:
  replicas: 1
  selector:
    name: twistlock-console
  template:
    metadata:
      annotations:
        traffic.sidecar.istio.io/excludeOutboundIPRanges: 35.238.214.241/32
```

Creating Istio Ingress and VirtualService resources for Console and Defender traffic

Set up two ingress points: one for Console's HTTPS web and API interface, and one for the WebSocket channel between Console and Defender.

Set up the certificates following the steps in Istio's documentation.

STEP 1 | Set up your certificate.

The high level commands are shown here. Full details can be found in Istio's documentation. These steps assume that your Console lives at https://twistlock.example.com. If you have your own certs, you will want to replace the certificates in the steps below with your own. For a quick test setup however, the following procedure will work.

$ git clone https://github.com/nicholasjackson/mtls-go-example
$ pushd mtls-go-example
$ ./generate.sh twistlock.example.com secretpassword
$ mkdir ~+1/twistlock.example.com && mv 1_root/ 2_intermediate/ 3_application/ 4_client/ ~+1/twistlock.example.com
$ popd

STEP 2 | Create a secret for your certificate.

$ kubectl create -n istio-system secret tls istio-ingressgateway-certs
  --key twistlock.example.com/3_application/private/twistlock.example.com.key.pem
  --cert twistlock.example.com/3_application/certs/twistlock.example.com.cert.pem
**STEP 3** | Set up an ingress point that forwards HTTPS traffic to Console.

1. Define a Gateway to expose port 443 at the edge of the mesh network to receive incoming HTTPS traffic.

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: twistlock-console-gateway
spec:
  selector:
    istio: ingressgateway # use Istio default gateway implementation
  servers:
    - port:
        number: 443
        name: https
        protocol: HTTPS
        tls:
          mode: SIMPLE
          serverCertificate: /etc/istio/ingressgateway-certs/tls.crt
          privateKey: /etc/istio/ingressgateway-certs/tls.key
      hosts:
        - "twistlock.example.com"
```

2. Define a VirtualService route incoming HTTPS traffic on port 443 to Prisma Cloud Console.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: twistlock-console
spec:
  gateways:
    - twistlock-console-gateway
  hosts:
    - "twistlock.example.com"
  tcp:
    - match:
        route:
          - destination:
              port:
                number: 8083
              host: twistlock-console
```

3. Deploy the HTTPS Gateway and VirtualService.

```bash
$ kubectl create -f console-ingress.yaml -n twistlock
$ kubectl create -f console-virtualservice.yaml -n twistlock
```

You should now be able to access Prisma Cloud Console at https://twistlock.example.com as long as its DNS resolves to the same IP as you have specified by the external IP in kubectl get svc istio-ingressgateway -n istio-system.

**STEP 4** | Set up an ingress point that will forward 8084 WebSocket traffic to the Console.

You can use an alternative port if that is what you have opened in your Istio ingress gateway, but you will then need to make sure that your Defender DaemonSet reflects the updated port. The only port that must remain 8084 will be the `spec.tcp.route.destination.port.number` setting that routes to the actual `twistlock-console` Kubernetes service. In the example below, you can set it up with the following ingress gateway and virtual service using the default 8084 port for your backend service. If you are using a specific SAN in the Prisma Cloud Console for Defender traffic, the wildcard can be replaced with an appropriate DNS hostname or IP address.
1. Define a Gateway to expose port 8084 at the edge of the mesh network for WebSocket traffic.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: twistlock-defender-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - '*'
  - name: communication-port
    port:
      name: communication-port
      number: 8084
      protocol: TCP
```

2. Define a VirtualService route WebSocket traffic from port 8084 to Prisma Cloud Console.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: twistlock-defender
spec:
  gateways:
  - twistlock-defender-gateway
  hosts:
  - '*'
  tcp:
  - match:
    - port: 8084
      route:
        destination:
          host: twistlock-console.twistlock.svc.cluster.local
          port:
            number: 8084
            weight: 100
```

3. Deploy the WebSocket Gateway and VirtualService.

```bash
$ kubectl create -f defender-ingress.yaml -n twistlock
$ kubectl create -f defender-virtualservice.yaml -n twistlock
```

**Installing the Defender DaemonSet**

Install Defender as a DaemonSet.

**STEP 1** | Generate the YAML for the Defender DaemonSet.

```bash
$ <platform>/twistcli defender export kubernetes \
  --address=https://twistlock.example.com \
  --cluster-address=<istio_ingress_gateway_external_ip>
```

**STEP 2** | Apply the new configuration.

```bash
$ kubectl create -f defender.yaml -n twistlock
```

You should now see your Defenders connect in Prisma Cloud Console.
Configuring Prisma Cloud Projects through Istio Ingress Controllers

The Prisma Cloud Projects feature can be implemented when the Supervisor Prisma Cloud Console is accessed through an Istio Ingress Controller. This is very similar to implementing Projects in OpenShift clusters. The Prisma Cloud Central Console must validate the Supervisor Prisma Cloud Console's TLS certificate. That certificate must be issued by Prisma Cloud. Therefore Istio is configured to allow TCP passthrough for the Supervisor Prisma Cloud Console's API endpoint. The Central Console's ingress configuration can still use the Istio certificates and HTTPS protocol as described above.

**STEP 1 | Supervisor** Console’s ingress controller Gateway.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: twistlock-console-gateway
spec:
  selector:
    istio: ingressgateway # use Istio default gateway implementation
  servers:
    - port:
        number: 443
        name: https
        protocol: TCP
        hosts:
          - "twistlock.example.com"
```

**STEP 2 | Supervisor** Console’s ingress controller VirtualService.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: twistlock-console
spec:
  gateways:
    - twistlock-console-gateway
  hosts:
    - "twistlock.example.com"
  tcp:
    - match:
        - port: 443
  route:
    - destination:
        port:
          number: 8083
        host: twistlock-console
```

Implementing SAML federation with a Prisma Cloud Console through Istio Ingress Controllers

When federating the Prisma Cloud Console that is accessed through an Istio Ingress Controller with a SAML v2.0 Identity Provider (IdP), the SAML authentication request's AssertionConsumerServiceURL value must be modified. Prisma Cloud automatically generates the AssertionConsumerServiceURL value sent in a SAML authentication request based on Console's configuration. When the Console is accessed through an Istio Ingress Controller, the URL for Console's API endpoint is most likely not the
same as the automatically generated AssertionConsumerServiceURL. Therefore, you must configure the AssertionConsumerServiceURL value that Prisma Cloud sends in the SAML authentication request.

**STEP 1** | Log into Prisma Cloud Console.

**STEP 2** | Go to Manage > Authentication > SAML.

**STEP 3** | In Console URL, define the AssertionConsumerServiceURL.
   - In this example, enter twistlock.example.com/api/v1/authenticate.
Disable automatic learning

Prisma Cloud lets you disable automatic learning to give you full control over creating, managing, and maintaining runtime rules for your apps.

Disabling automatic runtime learning is strongly discouraged. Prisma Cloud has been architected and optimized to automatically learn known good runtime behaviors, then create models that explicitly allow those behaviors. Disabling learning requires creating manual rules for all of these behaviors and greatly increases the likelihood of encountering false positive events.

If you have a regimented deployment process that must guarantee consistency between your test environment and your production environment, then you might want to disable automatic runtime learning, and manually create runtime rules instead. With this approach, the full range of runtime behaviors is locked down in production, and cannot be extended without manually adding new rules.

Models and learning

When a model is created for an entity, it's initially empty. Empty models don't allow any runtime behaviors. In a default installation, Prisma Cloud uses machine learning to compose models that encapsulate all known good behaviors. Models are sets of rules that allow process, network, and file system activity.

When learning is disabled, newly created models are empty. Since empty models don't allow any behaviors, you must manually create rules that explicitly allow process, network, and file system activity. Remember that rules come from two places: models (automatically created) and runtime rules (manually created). Manually created rules are designed to augment models when learning does not capture the full range of known good behaviors. When automatic learning is disabled, they must fully specify the full range of known good behaviors.

Deploying Prisma Cloud

Models created before automatic learning is disabled might still contain learned content. To guarantee all models are empty, disable automatic learning before deploying Defenders to your environment. Deploy Prisma Cloud in the following order:

1. Deploy Console.
2. Disable automatic learning.
3. Deploy Defenders.

Workflow

You should have two environments: test and production. Deploy Prisma Cloud Console to each environment. In the test environment, enable automatic learning. You'll use automatic learning to assist with the creation of rules. In the production environment, disable automatic learning. You'll port the rules from the test environment to the production environment.

The recommended workflow is:

1. Deploy your app to the test environment, and fully exercise it.
2. Validate models that were automatically created.
3. Export models from the test environment as rules.
4. Optionally store the rules in a source control system.
5. Import the rules into your production environment, where automatic learning is disabled.
Exporting and importing rules from the Console UI

After your app has been fully exercised in the test environment, create a rule from the runtime model. In Monitor > Runtime > Container Models, find your model, click Actions, then click Copy Into Rule.

Next, download the rule in JSON format. Go to Defend > Runtime > Container Policy, find your rule, and in the Actions menu, click Export.

Finally, import your rule into Console in your production environment. Go to Defend > Runtime > Container Policy, and click Import rule.

Exporting and importing rules programmatically

After your app has been fully exercised in the test environment, retrieve the model as a runtime rule. Use the GET /profiles/container/{id}/rule endpoint, where {id} is the profile ID.

A list of profiles (models) can be retrieved from GET /api/v1/profiles/container. Profile IDs can be found in the _id field. Profile ID is simply the concatenation of the image ID and an underscore.

```
$ curl -k
  -u ian
  -H 'Content-Type: application/json'
  -X GET
  https://<TEST-CONSOLE>:8083/api/v1/profiles/container/{id}/rule
  | jq '.' > model_rules.json
```

Then push the rule to Console in your production environment. When a rule is pushed with this endpoint, it is ordered first in the policy. Rule order is important, so be sure you’re pushing rules in the right order. The version of Console where the rule was exported must match the version of Console where it’s imported.

```
$ curl -k
  -u <USER>
  -X POST
  -H "Content-Type:application/json"
  https://<PROD-CONSOLE>:8083/api/v1/policies/runtime/container
  --data-binary @model_rules.json
```
The POST /api/v1/policies/runtime/container endpoint pushes one rule at a time. The PUT /api/v1/policies/runtime/container endpoint pushes the entire policy (i.e. all rules) in a single shot.
Performing a rolling upgrade of Defenders

Prisma Cloud requires a strict version matching of Console to Defender. After upgrading your Prisma Cloud Console, any Defenders which have not been upgraded to a matching version will be in a read-only state, and continue to enact the last policy prior to the Console upgrade. They will not send any information back to the Console until the Defender has been upgraded. While your Defenders will continue to protect your environment, they must be upgraded to become fully functional Defenders.

For large deployments, you may want to schedule a "rolling" upgrade of your Defenders over a period of time. While this is not a common need, this may be required due to change control requirements in your organization or due to issues with coordinating the upgrade of many clusters which may be owned by different teams.

The strategy described in this article details a plan to stand up a second Console during upgrade, replacing existing Defenders with upgraded Defenders pointed to the upgraded Console, and, eventually, retiring the initial Console. This will have all Defenders remain in an active state regardless of the time window of your upgrade.

Prerequisites:

- You have an existing Prisma Cloud Console at version \( n-1 \).
- You have downloaded the release tarball for target version \( n-1 \) and \( n \).
- You have the ability to create DNS records.
- You can install Defenders on your nodes.
- You have a second location where you can install a Prisma Cloud Console, which we will call the "upgrade host" (this second deployment could also be in a Kubernetes or Swarm cluster). This upgrade host must meet the minimum system requirements for a Prisma Cloud Console. Typically, the upgrade host will match the setup of your existing Prisma Cloud Console. After this procedure, this will be the new location for your Console.

If you are using custom certificates for authorization, you may have additional steps to add your custom certs with the correct DNS addresses. This can be avoided by using a wildcard cert (e.g. *.company.com) so that individual certificates do not need to be changed for new DNS records on your domain.

STEP 1 | On the upgrade host, first install a Prisma Cloud Console at version \( n-1 \) (the same version that your current Console is running).

STEP 2 | On the pre-upgrade Console host, Create a manual backup of your existing Prisma Cloud Console.

STEP 3 | On your upgrade host, restore the backup using twistcli from the backup created in the previous step.

STEP 4 | On your upgrade host, perform an upgrade to version \( n \).

STEP 5 | You now have two Consoles: your pre-upgrade Console and your upgraded Console. Create a new DNS record to point to your upgraded Console.

- For instance, if your pre-upgrade Console DNS was twistlock-18-11-127.my.company.com, your upgraded Console DNS may be twistlock-19-03-307.my.company.com.
STEP 6 | In your upgraded console, add the new DNS record entry into the SAN list.

STEP 7 | For each environment running Defenders, redeploy the Defenders to connect to the new upgraded Console. Typically, you’d use the same method you used in the initial environment, such as a Daemon Set.

- For each Defender, you must follow the instructions to install Defenders on your nodes. You should select the upgraded Console SAN as the name that clients and Defenders use to access this Console.
- The individual upgrades may be timed according to your needs. Since all Defenders are communicating with a Console at the corresponding version, all Defenders are fully functional.

STEP 8 | Eventually, all Defenders will be redeployed and no Defenders will be connecting into the pre-upgrade Console. At this point, the pre-upgrade Console can be decommissioned.

To maintain a single DNS record from which your users access the Console, consider using a CNAME record to point at the correct Console address at all times. For example, you may instruct your users to bookmark twistlock.my.company.com which would be a CNAME record that could be re-pointed between different Console versions as you complete the upgrades.
Debug data

Console and Defender emit logs as they run. These logs, also known as debug data, are designed to help troubleshoot operational issues. They're different than audits, which are designed to report significant security events.

If you contact Prisma Cloud Support with an issue, you’ll be asked to collect debug data from your setup and send it to us. Debug data helps us find the root cause of problems, and provide timely resolutions.

For more information about how to collect and send us your debug data, see Troubleshooting.

Reviewing Console’s debug logs

The simplest way to view Console’s debug logs is from within the UI itself. Go to Manage > View Logs > Console.

Reviewing Defender’s debug logs

To view Defender’s debug logs, go to Manage > Defenders > Manage. Find the Defender of interest in the table of deployed Defenders, and then click Actions > Logs.