Performance planning

This article describes the run-time characteristics of a typical Prisma Cloud deployment. The information provided is for planning and estimation purposes.

System performance depends on many factors outside of our control. For example, heavily loaded hosts have fewer available resources than hosts with balanced workloads.

Scale

Prisma Cloud has been tested and optimized to support up to 5,000 Defenders per Console.

If you have an environment that requires more than 5,000 Defenders, and you're using Compute Edition, deploy a scale project.

Scanning performance

This section describes the resources consumed by Prisma Cloud Defender during a scan. Measurements were taken on a test system with 1GB RAM, 8GB storage, and 1 CPU core.

Host scans

Host scans consume the following resources:

Resource	Measured consumption
Memory	10-15%
CPU	1%
Time to complete a host scan	1 second

Container scans

Container scans consume the following resources:

Resource	Measured consumption
Memory	10-15%
CPU	1%
Time to complete a container scan	1-5 seconds per container

Image scans

When an image is first scanned, Prisma Cloud caches its contents so that subsequent scans run more quickly. The first image scan, when there is no cache, consumes the following resources:

Resource	Measured consumption
Memory	10-15%
CPU	2%
Time to complete an image scan.	1-10 seconds per image. (Images are estimated to be 400-800 MB in size.)

Scans of cached images consume the following resources:

Resource	Measured consumption
Memory	10-15%
CPU	2%
Time to complete an image scan	1-5 seconds per image. (Images are estimated to be 400-800 MB in size.)

Real-world system performance

Each release, Prisma Cloud tests performance in a scaled out environment that replicates a real-world workload and configuration. The test environment is built on a Kubernetes cluster with the following properties:

- **Hosts:** 5,000
- **Hardware:**
  - **Console:** 8 vCPUs, 30 GB memory
  - **Defenders:** 2 vCPUs, 7.5 GB memory
- **Operating system:** Container-Optimized OS
- **Images:** 1,100
- **Containers:** 47,300 (density of 9.5 containers per host)

The results are collected over the course of a day. The default vulnerability policy (alert on everything) and compliance policy (alert on critical and high issues) are left in place. CNNF is enabled.

Resource consumption:

The following table shows normal resource consumption.

Component	Memory (RAM)	CPU (single core)
Console	1,924 MiB	4.0%
Defender	132 MiB	0.0 - 1.0%

CNAF Performance Benchmark

Minimum Requirements

Results detailed in this document assume a Defender instance complying with these minimum requirements.

Methodology

Benchmark Target Servers

Benchmark target servers were run on AWS EC2 instances running Ubuntu Server 18.04 LTS

Instance type	Environment	Compared servers	Versions
t2.large	Docker	Nginx vs CNAF	Nginx/1.19.0
t2.large	Host	Nginx vs CNAF	Nginx/1.14.0
t2.large	Kubernetes	Nginx vs CNAF	Nginx/1.17.10

Benchmarking Client

Benchmarking was performed using the hey load generating tool deployed on a 't2.large' instance running Ubuntu Server 18.04 LTS

Benchmark Scenarios

Test scenarios were run using hey against each server:

Scenario	HTTP Requests	Concurrent Connections
HTTP GET request	5,000	10, 100, 250, 500, 1,000
HTTP GET request with query parameters	5,000	10, 100, 250, 500, 1,000
HTTP GET request with an attack payload in a query parameter	5,000	10, 100, 250, 500, 1,000
HTTP GET with 1 MB response body	1,000	10, 100, 250, 500, 1,000
HTTP GET with 5 MB response body	1,000	10, 100, 250, 500, 1,000
HTTP POST request with body payload size of 100 bytes	5,000	10, 100, 250, 500, 1,000
HTTP POST request with body payload size of 1 KB	5,000	10, 100, 250, 500, 1,000
HTTP POST request with body payload size of 5 KB	5,000	10, 100, 250, 500, 1,000

NOTE	In order to support 1,000 concurrent connections in large file scenarios, CNAF HTTP body inspection size limit needs to be set to 104,857 bytes

Results

HTTP Transaction Overhead

The following table details request average **overhead** (in milliseconds):

Environment		Concurrent Connections				
		10	**100**	**250**	**500**	**1,000**
Docker	HTTP GET request	3	30	70	99	185
	HTTP GET request with query parameters	4	34	70	100	151
	GET w/ attack payload	1	6	6	26	96
	GET - 1MB Response	1	-268	-1314	-3211	-5152
	GET - 5MB Response	15	-1,641	-6,983	-9,262	-18,231
	POST w/ 100B body	5	42	84	119	194
	POST w/ 1KB body	12	106	245	430	800
	POST w/ 5KB body	42	402	970	1,853	3,189
Host	HTTP GET request	2	22	53	82	217
	HTTP GET request with query parameters	3	27	63	93	212
	GET w/ attack payload	0	6	17	78	104
	GET - 1MB Response	-1	-6	32	131	-681
	GET - 5MB Response	7	-45	-638	-2,677	-9,099
	POST w/ 100B body	3	29	66	114	300
	POST w/ 1KB body	10	97	234	436	774
	POST w/ 5KB body	39	407	940	1,831	3,196
Kubernetes	HTTP GET request	3	29	58	78	155
	HTTP GET request with query parameters	4	33	79	114	288
	GET w/ attack payload	0	5	15	63	177
	GET - 1MB Response	-4	-252	-981	-2827	-5754
	GET - 5MB Response	15	-1,653	-5,254	-14,966	-23,828
	POST w/ 100B body	5	39	92	130	280
	POST w/ 1KB body	11	109	252	498	907
	POST w/ 5KB body	43	421	1,013	2,005	3,557

NOTE CNAF response time can be faster than origin-server response time when attacks are blocked and not forwarded to the origin server.

Load Testing

The following table details average request time (in milliseconds) of 1,000,000 request benchmarking load (includes response time for both CNAF and underlying origin):

Environment		Concurrent Connections				
		10	**100**	**250**	**500**	**1,000**
Docker	HTTP GET request	4	36	90	177	358
	HTTP POST request, 100 Byte body	5	47	116	232	472
Host	HTTP GET request	3	28	70	140	298
	HTTP POST request, 100 Byte body	4	40	99	197	397
Kubernetes	HTTP GET request	4	38	92	181	363
	HTTP POST request, 100 Byte body	5	49	119	236	460