Performance Benchmarking

FortiSOAR 7.4.2

FORTINET DOCUMENT LIBRARY
https://docs.fortinet.com

FORTINET VIDEO LIBRARY
https://video.fortinet.com

FORTINET BLOG
https://blog.fortinet.com

CUSTOMER SERVICE & SUPPORT
https://support.fortinet.com

FORTINET TRAINING & CERTIFICATION PROGRAM
https://www.fortinet.com/training-certification

FORTINET TRAINING INSTITUTE
https://training.fortinet.com

FORTIGUARD LABS
https://www.fortiguard.com

END USER LICENSE AGREEMENT
https://www.fortinet.com/doc/legal/EULA.pdf

FEEDBACK
Email: techdoc@fortinet.com

October, 2023
FortiSOAR 7.4.2 Performance Benchmarking
00-400-000000-20210112

TABLE OF CONTENTS

Change Log **4**
FortiSOAR Performance Benchmarking for v7.4.2 ..... **5**
Summary 5
Summary of tests performed 5
Environment 6
FortiSOAR Virtual System Specifications 6
Operating System Specifications 6
External Tools Used 6
Pre-test conditions on both the standalone FortiSOAR system and the FortiSOAR High Availability (HA) cluster 6
Test setups 7
Test 1: Ingest alerts into FortiSOAR using an ingestion playbook 7
Steps followed 7
Observations 7
Test 2: Ingest alerts into FortiSOAR followed by automated indicator extraction 8
Steps followed 8
Observations 8
Test 3: Ingest alerts into FortiSOAR, automated indicator extraction, followed by enrichment of the extracted indicators 9
Steps followed 9
Observations 9
Test 4: Data Ingestion Test on a sizable data set 10
Data Ingestion Test on a sizable data set for a single-node FortiSOAR system 10
Data Ingestion Test on a sizable data set for an active-active FortiSOAR HA cluster with two systems 13
Summary of the Data Ingestion Test on sizable data: Performance comparison of a single FortiSOAR system and a FortiSOAR HA cluster 17
Test 5: Sustained Invocation Test for a single-node FortiSOAR system 17
Steps followed 17
Results 17
Graphs 18
Test 6: Sustained Invocation Test for an active-active FortiSOAR with two systems 21
Steps followed 21
Results 21
Graphs 21
Appendix: Sample playbooks zip files 24

Change Log

Date	Change Description
2023-10-25	Initial release of 7.4.2

FortiSOAR Performance Benchmarking for v7.4.2

This document details the performance benchmark tests conducted in Fortinet labs. The objective of this performance test is to measure the time taken to create alerts in FortiSOAR, and complete the execution of corresponding playbooks on the created alerts in the following cases:

- Single-node FortiSOAR system
- Cluster setup of FortiSOAR

The data from this benchmark test can help you in determining your scaling requirements for a FortiSOAR system to handle the expected workload in your environment.

Summary

The performance of a single-node FortiSOAR system and a two-node active-active FortiSOAR high-availability (HA) cluster was evaluated by conducting performance tests. The objective of the tests was to provide an analysis of how various FortiSOAR configurations perform for different types of workloads. The tests were performed on FortiSOAR images with their default OVA configurations; no performance tuning was done.

During the tests, both the single-node FortiSOAR system and the FortiSOAR HA cluster were subjected to real-world scenario simulations with identical workloads. Different test cases were performed that involved ingestion of varying numbers of records at different ingestion rates. The subsequent processing of the ingested data was also evaluated.

The results of the performance tests demonstrated that a FortiSOAR HA cluster was able to process workloads more effectively when compared to a the single-node system. The HA cluster showcased its ability to efficiently process workloads and deliver faster processing times. By distributing the workload among multiple nodes in an active-active configuration, the HA cluster enabled parallel processing, leading to a substantial performance improvement. Specifically, the tests revealed that a two-node HA cluster was approximately 35-40% faster than a single-node FortiSOAR system as summarized in the Summary of the Data Ingestion Test on sizable data: Performance comparison of a single FortiSOAR system and a FortiSOAR HA cluster topic. This significant speed advantage highlights the benefits of deploying an HA cluster, particularly when time-sensitive processing is crucial.

Summary of tests performed

The following tests are performed on both a single FortiSOAR system, and a HA cluster of two active-active FortiSOAR nodes:

1. Ingestion of alerts into FortiSOAR.
2. Ingestion of alerts followed by extraction of associated indicators.
3. Ingestion of alerts, then extraction of associated indicators, followed by enrichment of the extracted indications using one source.
4. Data ingest a large number of alerts. On alert creation the default SOAR Framework Solution Pack (SFSP) playbooks are automatically triggered to perform various processes such as extraction and enrichment of indicators, calculation of SLA, etc. For every ingested alert, 16 playbooks get triggered.
   For more information on SFSP see its documentation on FortiSOAR Content Hub.

5. Sustain the processes of ingestion of alerts for approximately a 12-hour period. On alert creation the default SOAR Framework Solution Pack (SFSP) playbooks are automatically triggered to perform various processes such as extraction of indicators, enrichment of indicators, etc. For every ingested alert, 16 playbooks get triggered.
   Note that this test is performed only on a single FortiSOAR system.

Environment

FortiSOAR Virtual System Specifications

Component	Specifications
FortiSOAR Build Version	7.4.2-3279
CPU	8 CPUs
Memory	32 GB
Storage	445.98 GB virtual disk, HDD with IOPS 3000, attached to a VMware ESXI system

Operating System Specifications

Operating System	Kernel Version
Rocky Linux 8.8	4.18.0-477.13.1.el8_8.x86_64

External Tools Used

Tool Name	Version
Zabbix server used to monitor system usage during the tests	6.4.1

Pre-test conditions on both the standalone FortiSOAR system and the FortiSOAR High Availability (HA) cluster

At the start of each test run -

- The test environment contains zero alerts.
- The test environment contains only the FortiSOAR built-in SOAR Framework Solution Pack (SFSP). The performance tests were performed using version 2.2.0 of SFSP.
- The test environment must contain a configured enrichment connector such as VirusTotal, URL scan.in,

etc., for test cases that require alerts to be enrichment.
- The test environments for Test1, 2, and 3 did not contain any playbooks in the 'Running' or 'Active' state.

In a production environment the network bandwidth, especially for outbound connections for enrichment connectors might vary that could affect the observations.

Test setups

1. For a single-node FortiSOAR system- Configure the standalone FortiSOAR system according to the specifications mentioned in the Environment topic.
2. For the FortiSOAR HA cluster - Create an active-active HA cluster of two FortiSOAR systems. Configure both the FortiSOAR systems according to the specifications mentioned in the Environment topic.

Test 1: Ingest alerts into FortiSOAR using an ingestion playbook

This test is executed by manually triggering an ingestion playbook that creates alerts in FortiSOAR.

Steps followed

1. Created the alerts using the ingestion playbook.
   You can download and use the sample playbooks contained in "PerfBenchmarking_Test01_PB_Collection_7_4.zip" file if you want to run the tests in your environment. All the sample playbook collections are included in Appendix: Sample playbooks zip files.
2. Once the alerts are created, measure the total time taken to create all the alerts in FortiSOAR.

Observations

The data in the following table outlines the number of alerts ingested and the total time taken to ingest those alerts:

Number of alerts created in FortiSOAR	Total number of playbooks executed in FortiSOAR	Total time (in minutes) taken to create all alerts in a standalone FortiSOAR system	Total time (in minutes) taken to create all alerts in an active-active FortiSOAR HA cluster with two systems
100	1	0.7	0.8
500	1	0.39	0.39

1000	1	1.19	1.19
1500	1	1.59	1.57
2000	1	2.38	2.36
2500	1	3.18	3.18

Note: Once this test is completed, refer to the pre-test conditions before starting a new test.

Test 2: Ingest alerts into FortiSOAR followed by automated indicator extraction

This test is executed by manually triggering an ingestion playbook that creates alerts in FortiSOAR. Once the alerts are created in FortiSOAR, an "Extraction" playbook is triggered and the total time taken for all the extraction playbooks to complete their execution is calculated.

Steps followed

1. Created the alerts using an ingestion playbook.
2. On alert creation, the "Extraction" playbooks are triggered. You can download and use the sample playbooks contained in the "PerfBenchmarking_Test02_PB_Collection_7_4.zip" file if you want to run the tests in your environment. All the sample playbook collections are included in Appendix: Sample playbooks zip files.

Observations

The data in the following table outlines the number of alerts ingested, the total time taken to ingest those alerts, and the total time taken for all the triggered playbooks to complete their execution:

Number of alerts created in FortiSOAR	Total number of playbooks executed in FortiSOAR	Total time (in minutes) taken to create all alerts in a standalone FortiSOAR system	Total time (in minutes) taken to create all alerts in an active-active FortiSOAR HA cluster with two sys-tems
100	101	0.24	0.19
500	501	1.40	1.06
1000	1001	3.12	2.19
1500	1501	4.51	3.3
2000	2001	6.30	4.39
2500	2501	8.6	4.56

Note: Once this test is completed, refer to the pre-test conditions before starting a new test.

Test 3: Ingest alerts into FortiSOAR, automated indicator extraction, followed by enrichment of the extracted indicators

The test was executed using an automated testbed that starts the ingestion which in turn creates alerts in FortiSOAR. Once the alerts are created in FortiSOAR, "Extraction" and "Enrichment" playbooks are triggered, and the total time taken for all the extraction and enrichment playbooks to complete their execution is calculated.

The setup for this test is exactly the same as Test 2, however this test additionally requires any enrichment connector to be configured.

Steps followed

1. Created the alerts using an ingestion playbook.
2. On alert creation, the "Extraction" and "Enrichment" playbooks are triggered. You can download and use the sample playbooks contained in the "PerfBenchmarking_Test03_PB_Collection_7_4.zip" file if you want to run the tests in your environment. All the sample playbook collections are included in Appendix: Sample playbooks zip files.

Observations

The data in the following table outlines the number of alerts ingested, the total time taken to ingest those alerts, and the total time taken for all the triggered playbooks to complete their execution.

Number of alerts created in FortiSOAR	Total number of playbooks executed in FortiSOAR	Total time (in minutes) taken to create all alerts in a standalone FortiSOAR system	Total time (in minutes) taken to create all alerts in an active-active FortiSOAR HA cluster with two sys-tems
100	1001	2.23	1.36
500	5001	11.53	7.44
1000	10001	23.13	14.26
1500	15001	35.2	21.57
2000	20001	47.20	29.34
2500	25001	59.13	37.9

Note: Once this test is completed, refer to the pre-test conditions before starting a new test. Also, note that enrichment playbooks make API calls over the Internet, and the times mentioned in this table to execute playbooks is inclusive of this time.

Test 4: Data Ingestion Test on a sizable data set

This test aims to evaluate FortiSOAR's performance during a prolonged data ingestion. The primary focus of these tests is to assess the efficiency and performance of the alert creation process and the execution of SFSP playbooks executions on a large data set. The objective is to gather data on the time taken to complete these operations in both a single-node FortiSOAR system and an active-active FortiSOAR HA cluster with two systems. The data gathered allows for a comparison between a single-node FortiSOAR system and a FortiSOAR HA cluster in regard to their performance in handling a significant number of alerts and playbook executions; summary of which is added in the Summary of the Data Ingestion Test on sizable data: Performance comparison of a single FortiSOAR system and a FortiSOAR HA cluster topic.

Data Ingestion Test on a sizable data set for a single-node FortiSOAR system

Steps followed

1. Created the alerts using an ingestion playbook.
   You can download and use the sample playbooks contained in the "PerfBenchmarking_Test04_PB_Collection_7_4_2.zip" file if you want to run the tests in your environment. All the sample playbook collections are included in Appendix: Sample playbooks zip files.
2. On alert creation, the default SFSP playbooks for extraction and enrichment of indicators, calculation of SLA, etc. are automatically triggered.
3. Record the total execution time for all these playbooks.

This test followed an ingestion rate of 50 alerts in bulk with a delay between each batch. Each alert generates 3 unique indicators in the FortiSOAR system, and the indicator enrichment process sets their reputation to 'Malicious'. Additionally, hourly, a burst of 100 alerts is ingested in bulk, which maintain the same indicator values as the previously injected data.

Number of alerts ingested: **29700**

Extracted indicator count: **85500**

Duration: **18 hours, 30 minutes, and 48 seconds**

Total number of playbooks executed: **474556**

Results

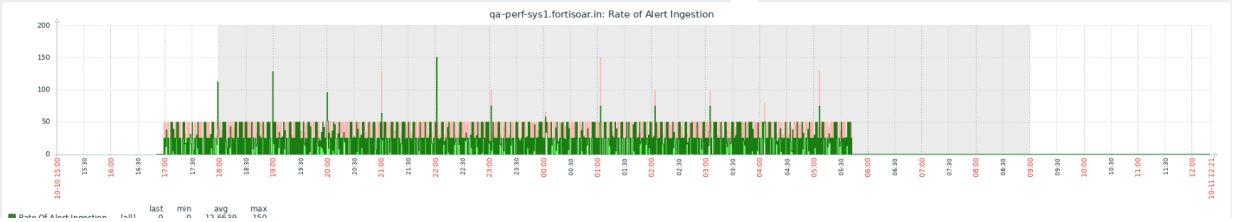
The system performed well under the sustained load. All 29700 alerts were successfully ingested and all the SFSP playbooks executed without any issues.

Graphs

The following graphs are plotted for the vital statistics for the system that was under test during the period of the test run.

Rate of Alert Ingestion

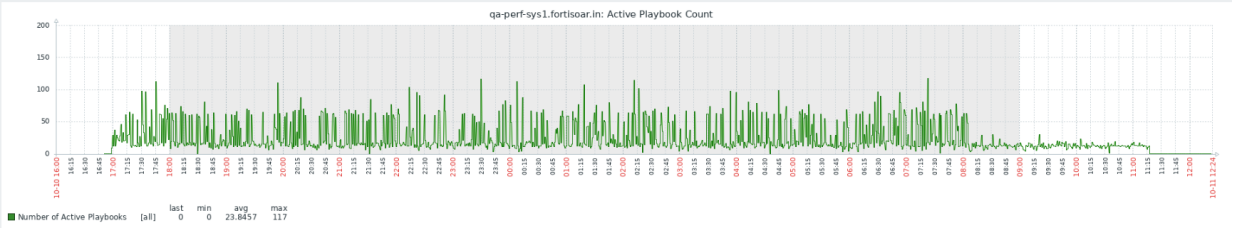
Analysis of alert ingestion when the test run was in progress on the system:



50 alerts were ingested in bulk with a delay of 50 seconds, followed by a burst of 100 alerts after 1 hour. The total number of alerts ingested was 29,700.

Active Playbook Graph

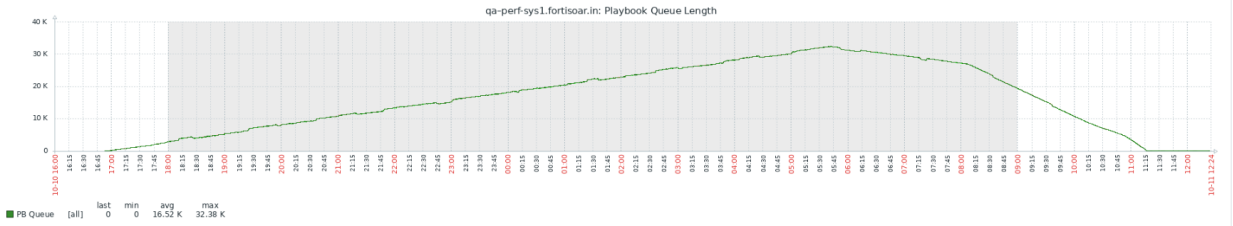
Analysis of active playbooks that were running when the test run was in progress on the system:



Celery workers were responsible for distribution of the tasks and execution of playbooks in parallel. On an average 24 playbooks were executed in a minute.

RabbitMQ Playbook Queue Graph

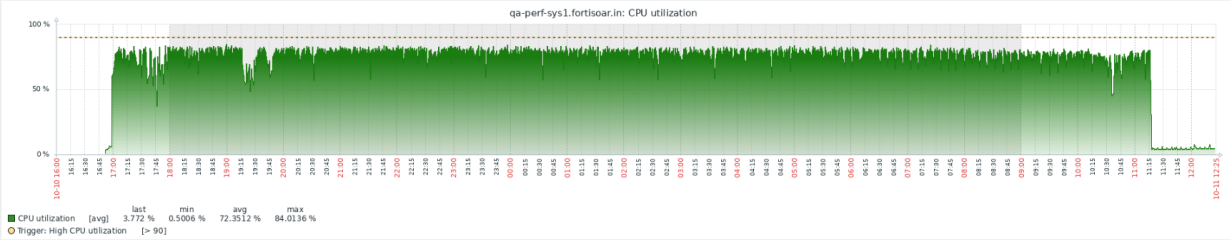
Analysis of RabbitMQ Playbook Queue Count when the test run was in progress on the system:



It was observed that during the test the `rabbitmq_pb_queue` rose to a maximum of 32.38k playbooks queued on the system for some time but eventually went back to 0. This means that no playbooks remained in the queue and that all the required playbooks associated with the alerts were getting completed, i.e., all alerts were created, and their indicators were extracted and enriched.

CPU Utilization Graph

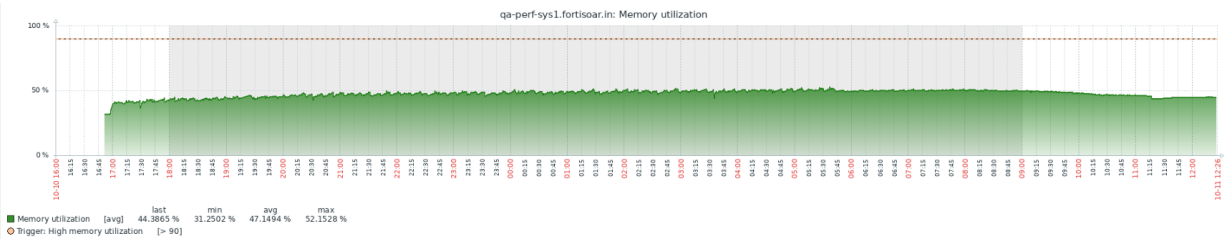
Analysis of CPU Utilization when the test run was in progress on the system:



Using the system resources specified in the "Environment" topic, it was observed that this test was running the CPU utilization was around 84% on average.

Memory Utilization Graph

Analysis of Memory Utilization when the test run was in progress on the system:

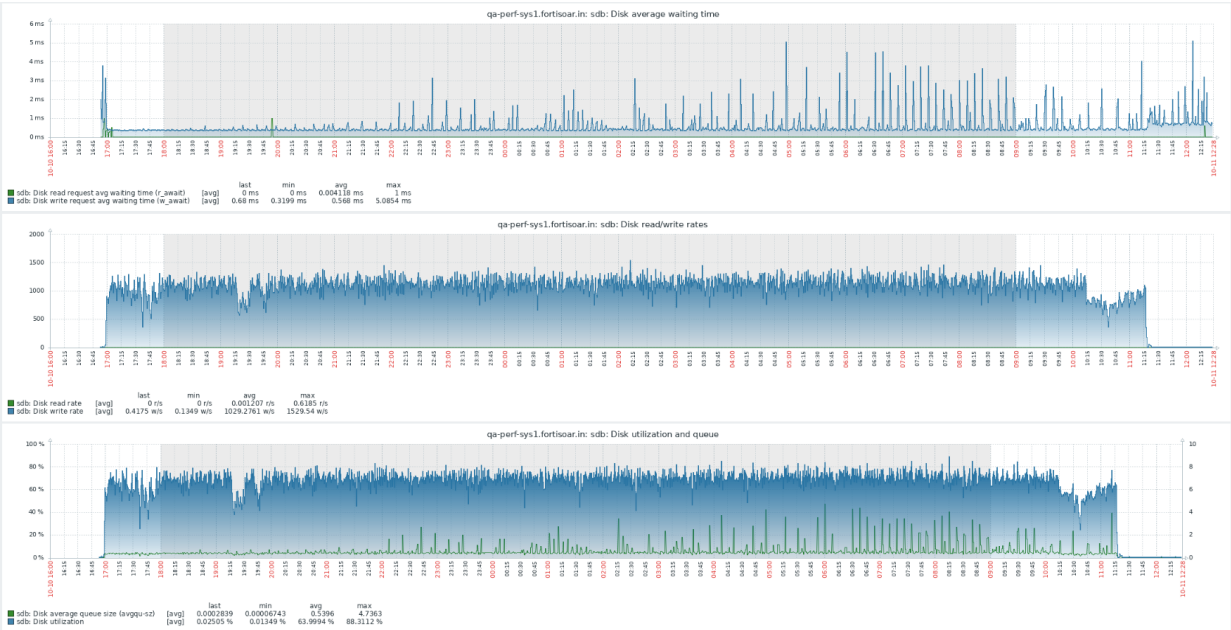


Using the system resources specified in the "Environment" topic, it was observed that this test was running the Memory utilization was around 52.15% on average.

PostgreSQL Disk Utilization and Read/Write IO Wait Graph

The Disk Utilization graph represent the percentage of elapsed time during which the PostgreSQL disk drive was busy while servicing 'Read' or 'Write' requests.

Analysis of PostgreSQL Disk Utilization when the test run was in progress on the system:

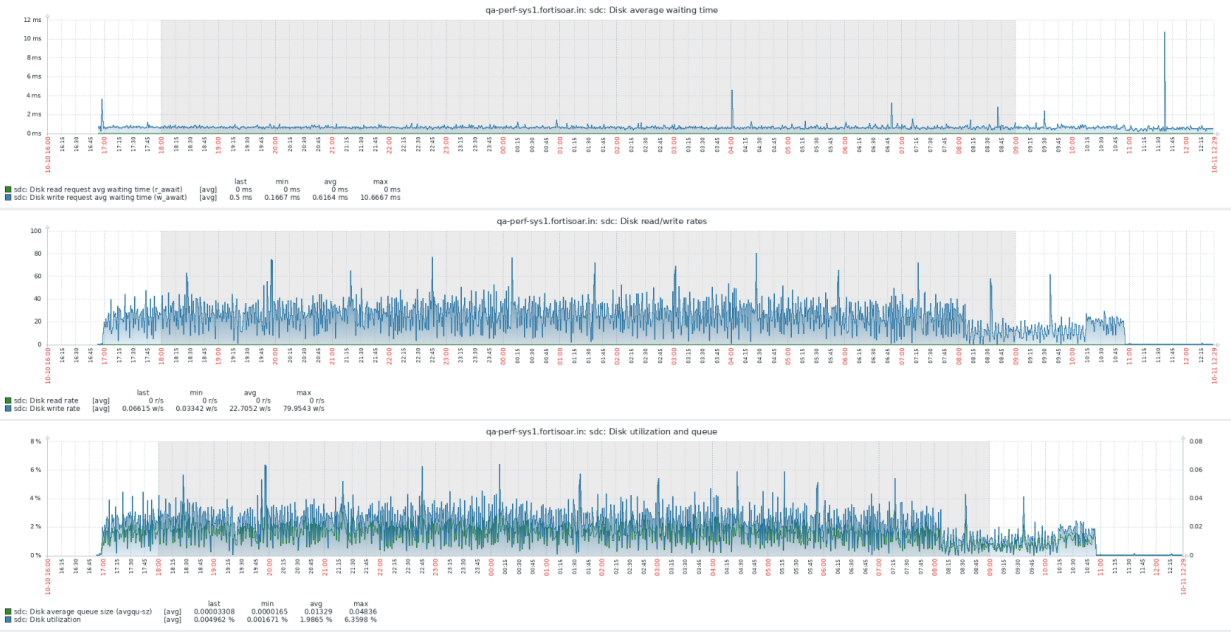


Using the system resources specified in the "Environment" topic, it was observed that when the test was running the PostgreSQL disk utilization averaged at 63.9%. The "Read" Wait for the PostgreSQL disk averaged around 0.001 milliseconds. The "Write" Wait for the PostgreSQL disk averaged around 1029.27 millisecond, with the maximum wait of 5.08 milliseconds and the minimum wait of 0.3 milliseconds.

ElasticSearch Disk Utilization and Read/Write IO Wait Graph

The Disk Utilization graph represent the percentage of elapsed time during which the ElasticSearch disk drive was busy while servicing 'Read' or 'Write' requests.

Analysis of ElasticSearch Disk Utilization when the test run was in progress on the system:



Using the system resources specified in the "Environment" topic, it was observed that when the test was running the ElasticSearch disk utilization averaged at 1.9%. The "Read" Wait for the ElasticSearch disk averaged around 0 milliseconds. The "Write" Wait for the ElasticSearch disk averaged around 22.7 millisecond, with the maximum wait of 10.66 milliseconds and the minimum wait of 0.1 milliseconds.

Data Ingestion Test on a sizable data set for an active-active FortiSOAR HA cluster with two systems

Steps followed

1. Created the alerts using an ingestion playbook.
   You can download and use the sample playbooks contained in the "PerfBenchmarking_Test04_PB_Collection_7_4_2.zip" file if you want to run the tests in your environment. All the sample playbook collections are included in Appendix: Sample playbooks zip files.
2. On alert creation, the default SFSP playbooks for extraction and enrichment of indicators, calculation of SLA, etc. are automatically triggered.
3. Record the total execution time for all these playbooks.

This test followed an ingestion rate of 50 alerts in bulk with a delay between each batch. Each alert generates 3 unique indicators in the FortiSOAR system, and the indicator enrichment process sets their reputation to 'Malicious'. Additionally, hourly, a burst of 100 alerts is ingested in bulk, which maintain the same indicator values as the previously injected data.

Number of alerts ingested: **29700**

Extracted indicator count: **85500**

Duration: **12 hours, 9 minutes, 24 seconds**

Total number of playbooks executed: **474780**

Results

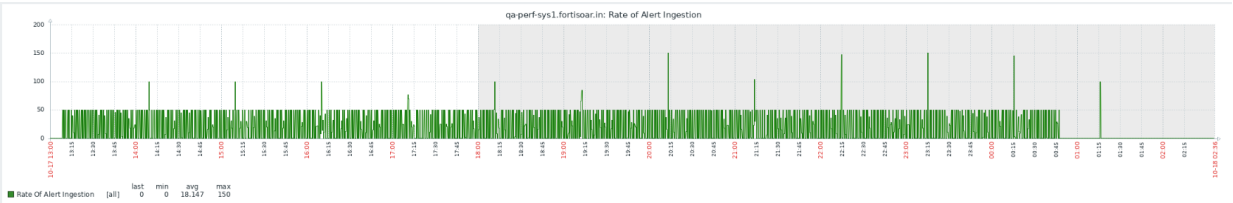
The system performed well under the sustained load. All 29700 alerts were successfully ingested and all the SFSP playbooks executed without any issues.

Graphs

The following graphs are plotted for the vital statistics for the system that was under test during the period of the test run.

Rate of Alert Ingestion

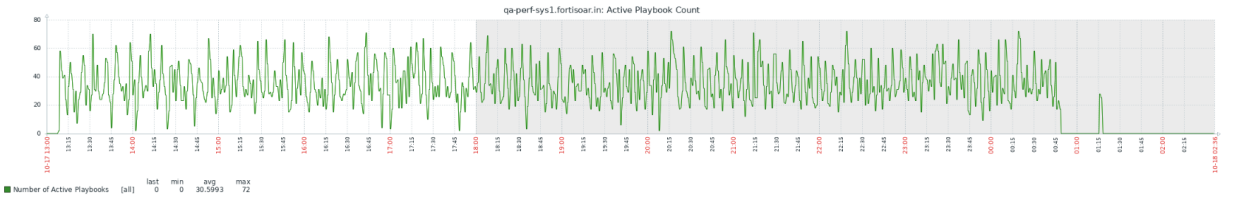
Analysis of alert ingestion when the test run was in progress on the system:



50 alerts were ingested in bulk with a delay of 50 seconds, followed by a burst of 100 alerts after 1 hour. The total number of alerts ingested was 29,700.

Active Playbook Graph

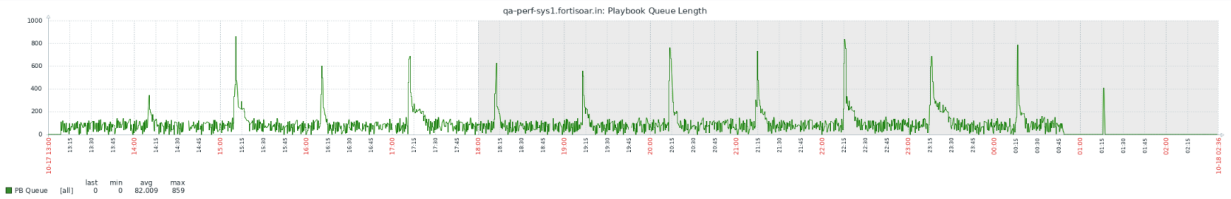
Analysis of active playbooks that were running when the test run was in progress on the system:



Celery workers were responsible for distribution of the tasks and execution of playbooks in parallel. On an average 72 playbooks were executed in a minute.

RabbitMQ Playbook Queue Graph

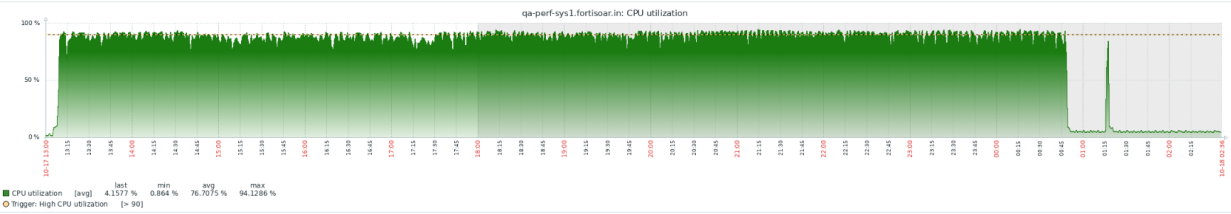
Analysis of RabbitMQ Playbook Queue Count when the test run was in progress on the system:



It was observed that during the test the `rabbitmq_pb_queue` rose to a maximum of 859 playbooks queued on the system for some time but eventually went back to 0. This means that no playbooks remained in queue and that all the required playbooks associated with the alerts were getting completed, i.e., all alerts were created, and their indicators were extracted and enriched.

CPU Utilization Graph

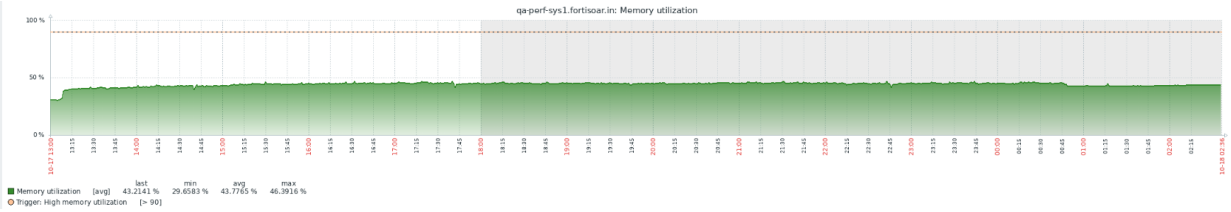
Analysis of CPU Utilization when the test run was in progress on the system:



Using the system resources specified in the "Environment" topic, it was observed that this test was running the CPU utilization was around 76.70% on average.

Memory Utilization Graph

Analysis of Memory Utilization when the test run was in progress on the system:



Using the system resources specified in the "Environment" topic, it was observed that this test was running the Memory utilization was around 43.77%.

PostgreSQL Disk Utilization and Read/Write IO Wait Graph

The Disk Utilization graph represent the percentage of elapsed time during which the PostgreSQL disk drive was busy while servicing 'Read' or 'Write' requests.

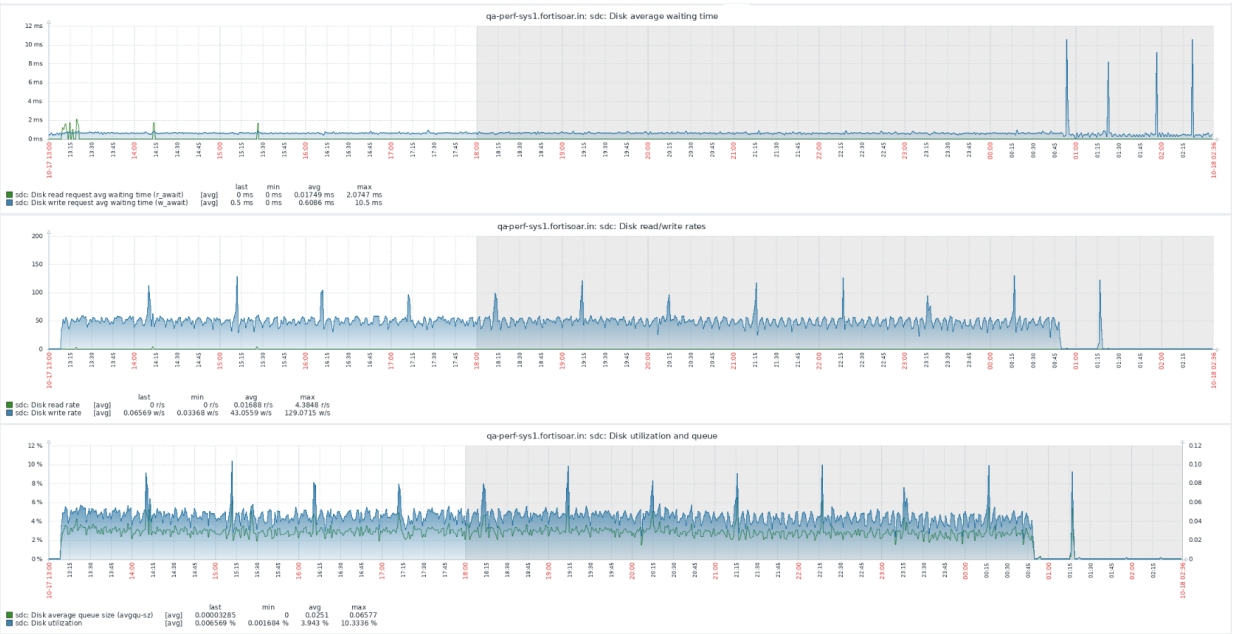
Analysis of PostgreSQL Disk Utilization when the test run was in progress on the system:

Using the system resources specified in the "Environment" topic, it was observed that when the test was running the PostgreSQL disk utilization averaged at 68.22%. The "Read" Wait for the PostgreSQL disk averaged around 0.7 milliseconds. The "Write" Wait for the PostgreSQL disk averaged around 1135.7898 millisecond, with the maximum wait of 6.9 milliseconds and the minimum wait of 0.29 milliseconds.

ElasticSearch Disk Utilization and Read/Write IO Wait Graph

The Disk Utilization graph represent the percentage of elapsed time during which the ElasticSearch disk drive was busy while servicing 'Read' or 'Write' requests.

Analysis of ElasticSearch Disk Utilization when the test run was in progress on the system:



Using the system resources specified in the "Environment" topic, it was observed that when the test was running the ElasticSearch disk utilization averaged at 3.9%. The "Read" Wait for the ElasticSearch disk averaged around 0 milliseconds. The "Write" Wait for the ElasticSearch disk averaged around 43.05 milliseconds, with the maximum wait of 10 milliseconds and the minimum wait of 0 milliseconds.

Summary of the Data Ingestion Test on sizable data: Performance comparison of a single FortiSOAR system and a FortiSOAR HA cluster

In the standalone FortiSOAR system, it took 18 hours, 30 minutes, and 48 seconds to process the 29,700 alerts. However, in the active-active FortiSOAR HA cluster with two systems, the same set of alerts was processed in 12 hours, 9 minutes, and 24 seconds. Therefore, for the same workload, a two-node HA cluster saves 6 hours, 41 minutes, and 24 seconds while ingesting data, making the FortiSOAR HA cluster around 35-40% faster when compared to a single node FortiSOAR system.

Test 5: Sustained Invocation Test for a single-node FortiSOAR system

The objective of this test is to process the ingested alert data in a minute without building up the queue.

Steps followed

1. Created the alerts using an ingestion playbook.
   You can download and use the sample playbooks contained in the "PerfBenchmarking_Test05_PB_Collection_7_4_2.zip" file if you want to run the tests in your environment. All the sample playbook collections are included in Appendix: Sample playbooks zip files.
2. On alert creation, the default SFSP playbooks for extraction and enrichment of indicators, calculation of SLA, etc. are automatically triggered.
3. Record the total execution time for all these playbooks.

This test followed an ingestion rate of 20 alerts in bulk with a delay between each batch. Each alert generates 3 unique indicators in the FortiSOAR system, and the indicator enrichment process sets their reputation to 'Malicious'.

Number of alerts ingested: **14400**

Duration: **12 hours 27 minutes**

Total number of playbooks executed: **232977**

Results

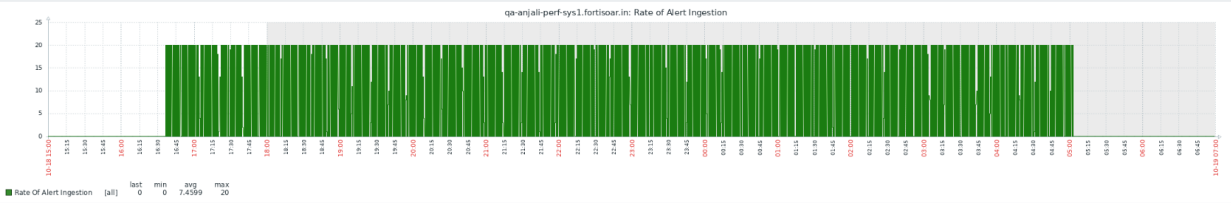
The system performed well under the sustained load. All 14400 alerts were successfully ingested and all the SFSP playbooks executed without any queuing.

Graphs

The following graphs are plotted for the vital statistics for the system that was under test during the period of the test run.

Rate of Alert Ingestion

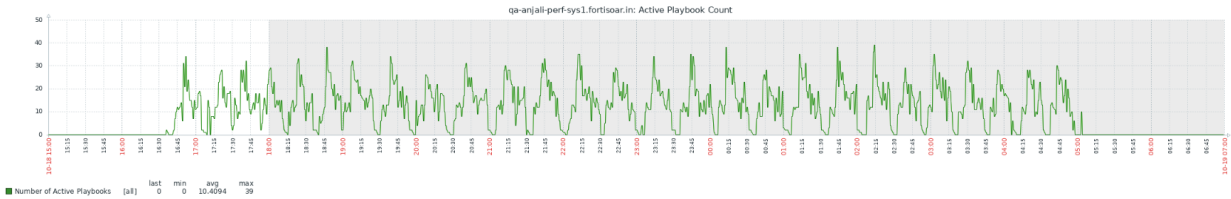
Analysis of alert ingestion when the test run was in progress on the system:



20 alerts were ingested in bulk with a delay of 55 seconds for 12.30 hours.

Active Playbook Graph

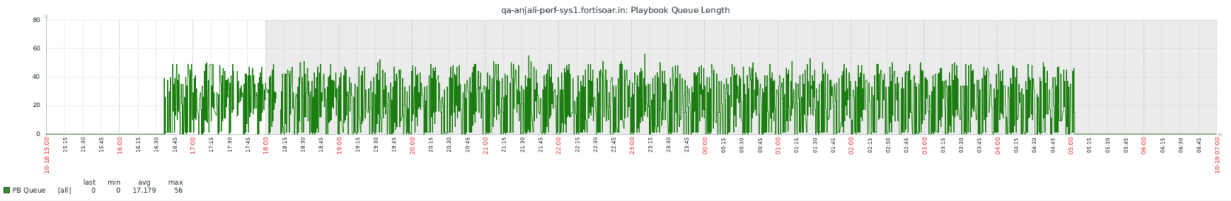
Analysis of active playbooks that were running when the test run was in progress on the system:



Celery workers were responsible for distribution of the tasks and execution of playbooks in parallel. On an average 39 playbooks were executed in a minute.

RabbitMQ Playbook Queue Graph

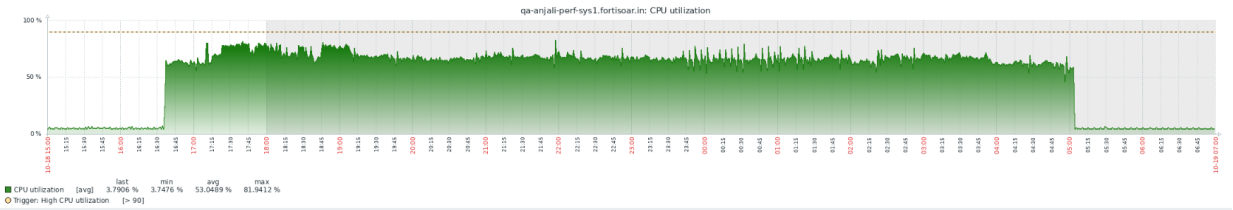
Analysis of RabbitMQ Playbook Queue Count when the test run was in progress on the system:



It was observed that during the test the `rabbitmq_pb_queue` rose to a maximum of 56 playbooks queued on the system for some time but eventually went back to 0. This means that no playbooks remained in queue and that all the required playbooks associated with the alerts were getting completed, i.e., all alerts were created, and their indicators were extracted and enriched.

CPU Utilization Graph

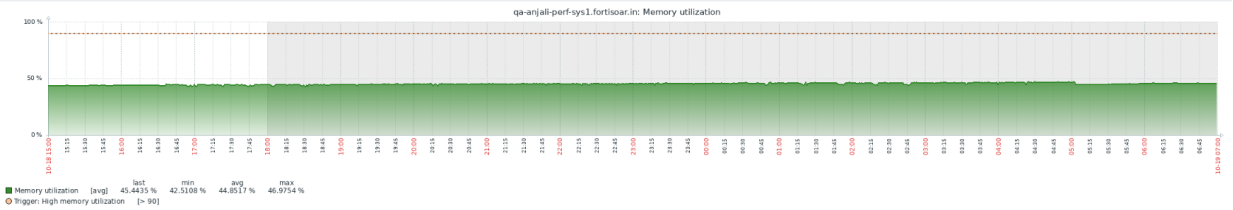
Analysis of CPU Utilization when the test run was in progress on the system:



Using the system resources specified in the "Environment" topic, it was observed that this test was running the CPU utilization was around 53% on average.

Memory Utilization Graph

Analysis of Memory Utilization when the test run was in progress on the system:

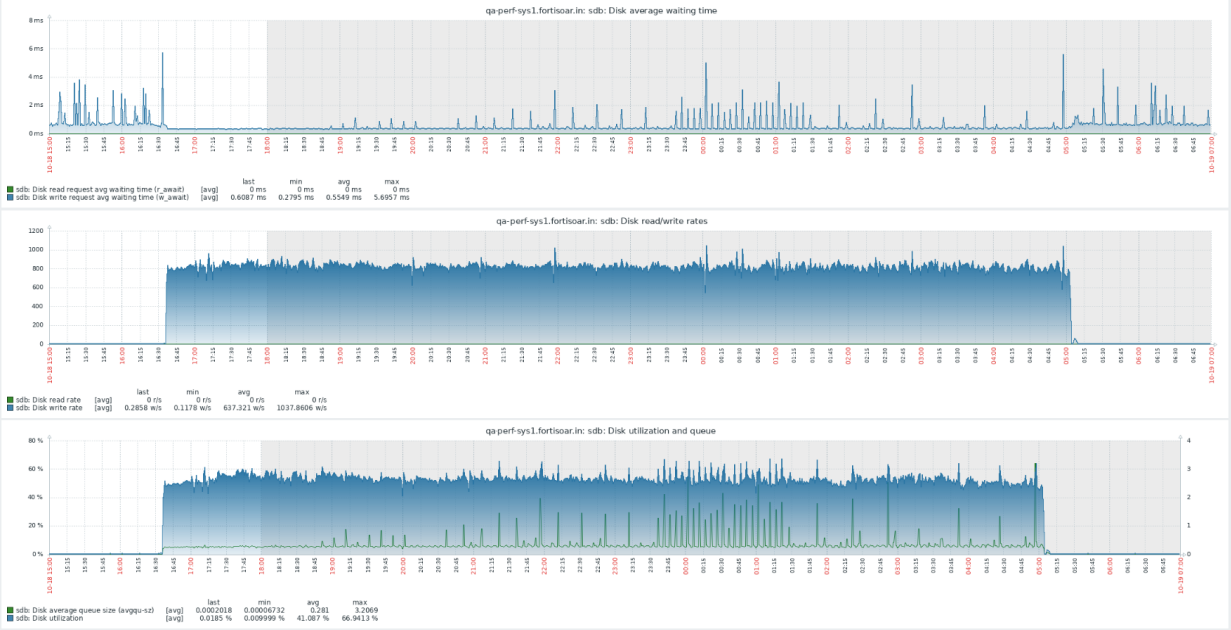


Using the system resources specified in the "Environment" topic, it was observed that this test was running the Memory utilization was around 44% on average.

PostgreSQL Disk Utilization and Read/Write IO Wait Graph

The Disk Utilization graph represent the percentage of elapsed time during which the PostgreSQL disk drive was busy while servicing 'Read' or 'Write' requests.

Analysis of PostgreSQL Disk Utilization when the test run was in progress on the system:



Using the system resources specified in the "Environment" topic, it was observed that when the test was running the PostgreSQL disk utilization averaged at 41.08%. The "Read" Wait for the PostgreSQL disk averaged around 0 milliseconds. The "Write" Wait for the PostgreSQL disk averaged around 637.321 milliseconds, with the maximum wait of 5.6 milliseconds and the minimum wait of 0.2 milliseconds.

ElasticSearch Disk Utilization and Read/Write IO Wait Graph

The Disk Utilization graph represent the percentage of elapsed time during which the ElasticSearch disk drive was busy while servicing 'Read' or 'Write' requests.

Analysis of ElasticSearch Disk Utilization when the test run was in progress on the system:

Using the system resources specified in the "Environment" topic, it was observed that when the test was running the ElasticSearch disk utilization averaged at 2.4%. The "Read" Wait for the ElasticSearch disk averaged around 0 milliseconds. The "Write" Wait for the ElasticSearch disk averaged around 26.06 milliseconds, with the maximum wait of 1.3 milliseconds and the minimum wait of 0 milliseconds.

Test 6: Sustained Invocation Test for an active-active FortiSOAR with two systems

The objective of this test is to process the ingested alert data in a minute without building up the queue.

Steps followed

1. Created the alerts using an ingestion playbook.
   You can download and use the sample playbooks contained in the "PerfBenchmarking_Test06_PB_Collection_7_4_2.zip" file if you want to run the tests in your environment. All the sample playbook collections are included in Appendix: Sample playbooks zip files.
2. On alert creation, the default SFSP playbooks for extraction and enrichment of indicators, calculation of SLA, etc. are automatically triggered.
3. Record the total execution time for all these playbooks.

This test followed an ingestion rate of 29 alerts in bulk with a delay between each batch. Each alert generates 3 unique indicators in the FortiSOAR system, and the indicator enrichment process sets their reputation to 'Malicious'.

Number of alerts ingested: **20880**

Duration: **12 hours 28 minutes**

Total number of playbooks executed: **336472**

Results

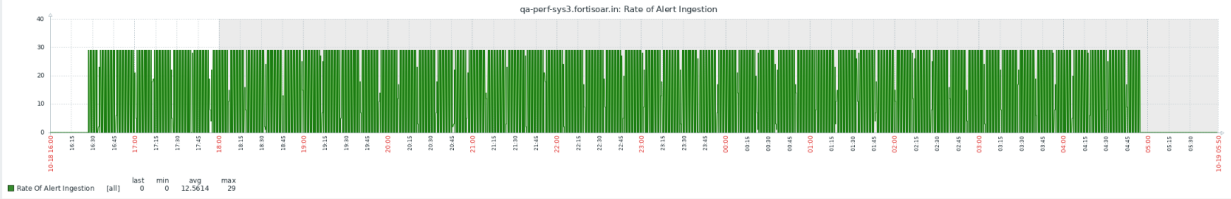
The system performed well under the sustained load. All 20880 alerts were successfully ingested and all the SFSP playbooks executed without any queuing.

Graphs

The following graphs are plotted for the vital statistics for the system that was under test during the period of the test run.

Rate of Alert Ingestion

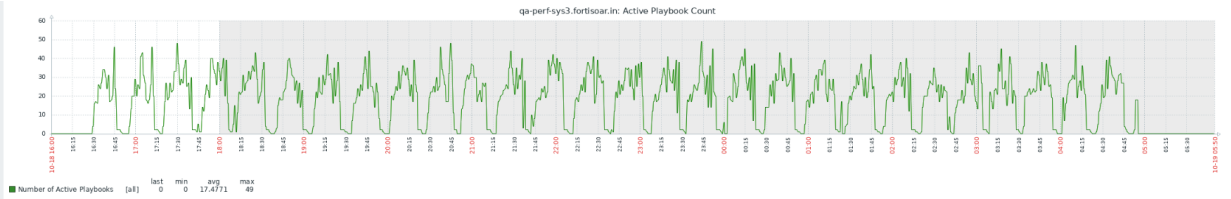
Analysis of alert ingestion when the test run was in progress on the system:



29 alerts were ingested in bulk with a delay of 55 seconds for 12.30 hours.

Active Playbook Graph

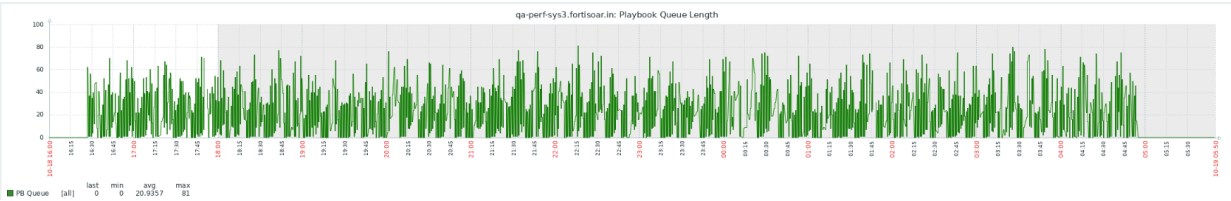
Analysis of active playbooks that were running when the test run was in progress on the system:



Celery workers were responsible for distribution of the tasks and execution of playbooks in parallel. On an average 49 playbooks were executed in a minute.

RabbitMQ Playbook Queue Graph

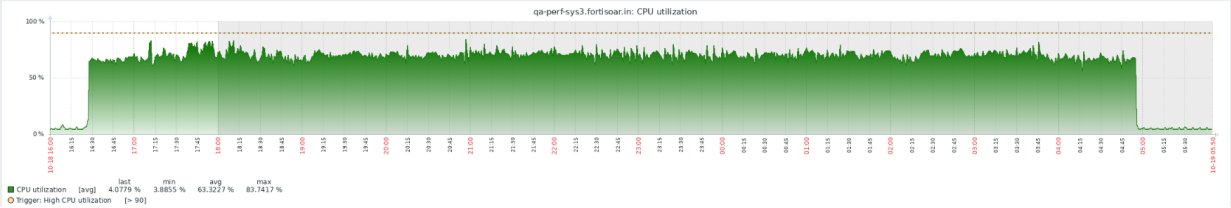
Analysis of RabbitMQ Playbook Queue Count when the test run was in progress on the system:



It was observed that during the test the `rabbitmq_pb_queue` rose to a maximum of 81 playbooks queued on the system for some time but eventually went back to 0. This means that no playbooks remained in queue and that all the required playbooks associated with the alerts were getting completed, i.e., all alerts were created, and their indicators were extracted and enriched.

CPU Utilization Graph

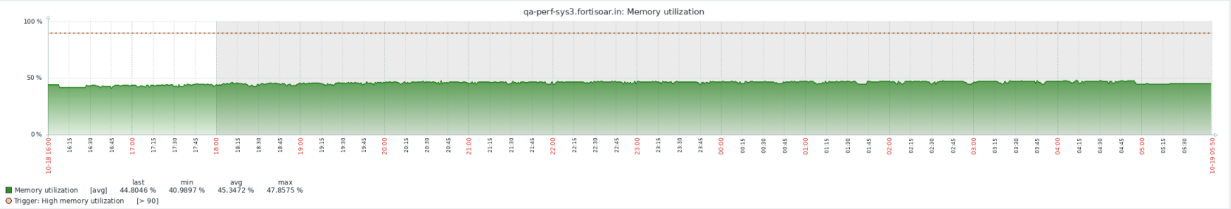
Analysis of CPU Utilization when the test run was in progress on the system:



Using the system resources specified in the "Environment" topic, it was observed that this test was running the CPU utilization was around 63% on average.

Memory Utilization Graph

Analysis of Memory Utilization when the test run was in progress on the system:

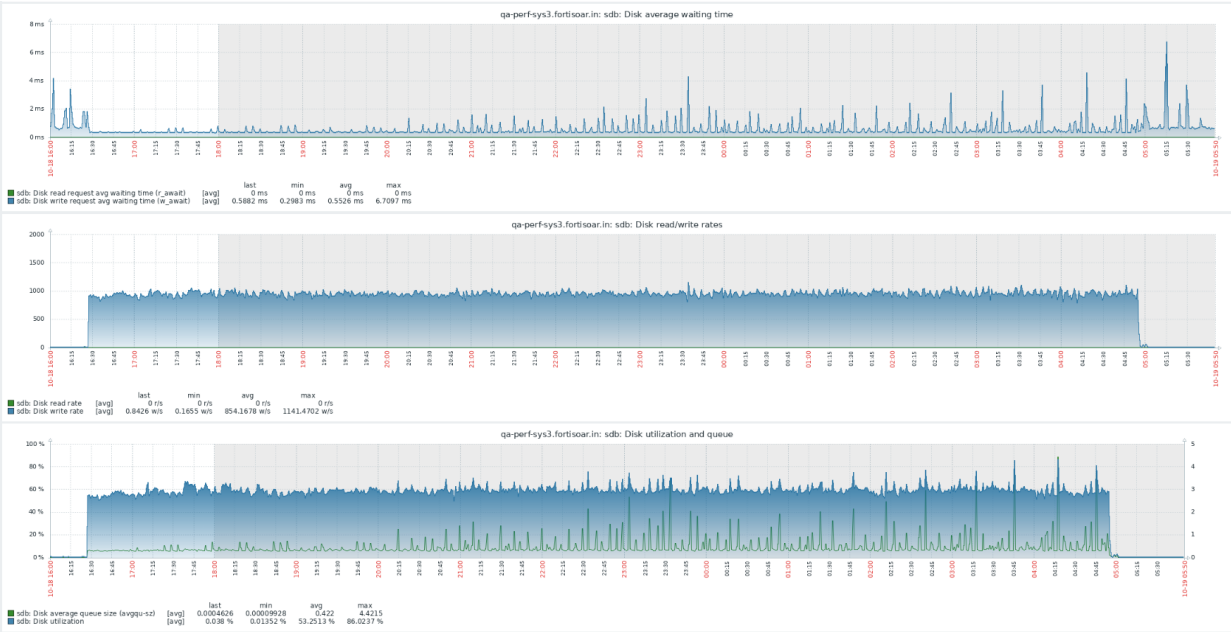


Using the system resources specified in the "Environment" topic, it was observed that this test was running the memory utilization was around 45% on average.

PostgreSQL Disk Utilization and Read/Write IO Wait Graph

The Disk Utilization graph represent the percentage of elapsed time during which the PostgreSQL disk drive was busy while servicing 'Read' or 'Write' requests.

Analysis of PostgreSQL Disk Utilization when the test run was in progress on the system:

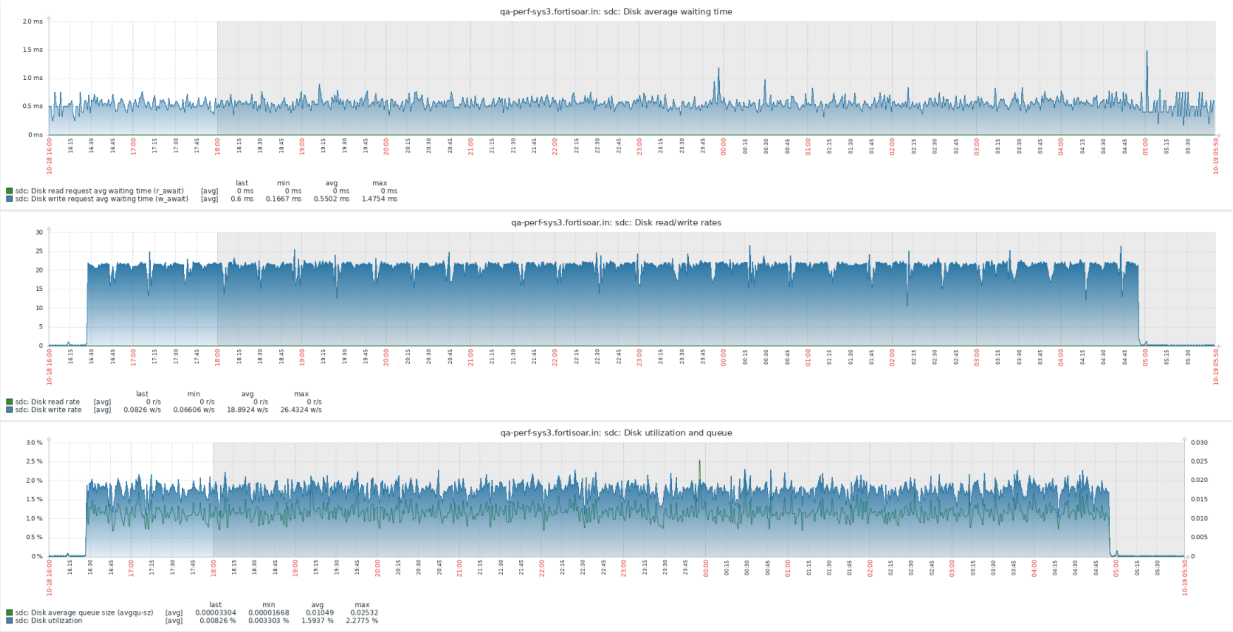


Using the system resources specified in the "Environment" topic, it was observed that when the test was running the PostgreSQL disk utilization averaged at 53.25%. The "Read" Wait for the PostgreSQL disk averaged around 0 milliseconds. The "Write" Wait for the PostgreSQL disk averaged around 854.16 millisecond, with the maximum wait of 6.7 milliseconds and the minimum wait of 0.29 milliseconds.

ElasticSearch Disk Utilization and Read/Write IO Wait Graph

The Disk Utilization graph represent the percentage of elapsed time during which the ElasticSearch disk drive was busy while servicing 'Read' or 'Write' requests.

Analysis of ElasticSearch Disk Utilization when the test run was in progress on the system:



Using the system resources specified in the "Environment" topic, it was observed that when the test was running the ElasticSearch disk utilization averaged at 1.5%. The "Read" Wait for the ElasticSearch disk averaged around 0 milliseconds. The "Write" Wait for the ElasticSearch disk averaged around 18.89 millisecond, with the maximum wait of 1.4 milliseconds and the minimum wait of 0.16 milliseconds.

Appendix: Sample playbooks zip files

You can download the following sample playbook collections so that you can run the same tests in your environment to see the performance in your version/hardware platforms. Or, if you want to make some additions that are specific to your environment, you can also tweak the existing playbooks.

- PerfBenchmarking_Test01_PB_Collection_7_4.zip
- PerfBenchmarking_Test02_PB_Collection_7_4.zip
- PerfBenchmarking_Test03_PB_Collection_7_4.zip
- PerfBenchmarking_Test04_PB_Collection_7_4_2.zip
- PerfBenchmarking_Test05_PB_Collection_7_4_2.zip
- PerfBenchmarking_Test06_PB_Collection_7_4_2.zip

www.fortinet.com