Hewlett Packard Enterprise

Technical white paper

Check if the document is available in the language of your choice.

OPEN SOURCE MONITORING OF HPE SIMPLIVITY

How to connect HPE SimpliVity to Prometheus

CONTENTS

EXECUTIVE SUMMARY

More and more IT organizations are implementing containers to accelerate innovation cycles by simplifying DevOps. Using container fundamentally changed the way applications are delivered. Applications and functionality are nowadays split into multiple containers that are orchestrated by tools such as Kubernetes or OpenShift. Hyperconverged systems like HPE SimpliVity are a perfect platform to handle the unpredictable workloads and the fast development cycles of today's containerized environments. HPE Express Containers with Docker Enterprise Edition on HPE SimpliVity, a complete solution from Hewlett Packard Enterprise, combines HPE SimpliVity with Docker enterprise-grade container platform, along with Advisory and Deployment Services from HPE Pointnext Services. It simplifies the step into a private container-as-a-service (CaaS) platform. The HPE Reference Configuration for Red Hat® OpenShift Container Platform (OCP) on HPE SimpliVity is another example of the growing support of CaaS platforms on hyperconverged systems.

At the same time, CaaS platforms do have specific monitoring requirements that are addressed by the Prometheus open-source monitoring solution. Prometheus, together with Grafana for the visualization (see Figure 1) is the de-facto standard for monitoring containerized environments. While there is a long list of exporters and integrations for multiple systems available, an integration to monitor HPE SimpliVity cluster and nodes was missing up to now.

FIGURE 1. HPE SimpliVity Federation Overview Dashboard

This white paper describes a Prometheus connector for HPE SimpliVity that can still be customized as needed and it delivers a deep monitoring possibility. It will describe the basic concepts of the connector, how to implement the connector as a container, and discusses possible Grafana dashboards to visualize the collected HPE SimpliVity metrics.

Target audience—Chief technology officers (CTOs), data center managers, enterprise architects, presales, deployment engineers, and others who want to implement an open-source-based monitoring realm. A working knowledge of Python and Rest API is recommended for the actual implementation task.

Document purpose—The purpose of this document is to describe an implementation of an open-source monitoring environment based on Prometheus and Grafana to get an integrated container and infrastructure monitoring system in place.

SOLUTION OVERVIEW

All status and performance metrics of an HPE SimpliVity system are available via Rest API calls. The connection with the Prometheus monitoring system is implemented as a continuously running Python script that utilizes HPE SimpliVity Rest API calls to collect the necessary metrics of an HPE SimpliVity Federation and present it via a configurable TCP/IP port (port 9091 in Figure 2).

If multiple HPE SimpliVity Federation needs to be monitored, then a separate connector is needed per federation. Prometheus is collecting metrics from multiple sources, which is defined in the prometheus.yml file and stores it in the Prometheus time series database. It is possible to run queries and select metrics using Prometheus only by accessing the Prometheus system on TCP/IP port 9090

(http://<prometheus_system_ip_address>:9090). But advanced visualization capabilities including the possibility to define customized dashboards and customized alerts is the strength of Grafana, that does have a built-in connector for Prometheus. Hence, it is common to use Grafana to visualize the captured metrics, as it is shown in Figure 2.

The HPE SimpliVity monitoring environment of the customer technology center (CTC) in Böblingen, Germany, is fully containerized and it runs in a Kubernetes container orchestration environment. Prometheus and Grafana are available as container images on Docker Hub and the Docker compose file to build the HPE SimpliVity connector as a container is provided in Appendix A: HPE SimpliVity connector—Docker file. Only the HPE SimpliVity monitoring part of this environment is shown in the logical diagram in Figure 2, other monitoring pieces are being added at the time of writing the white paper.

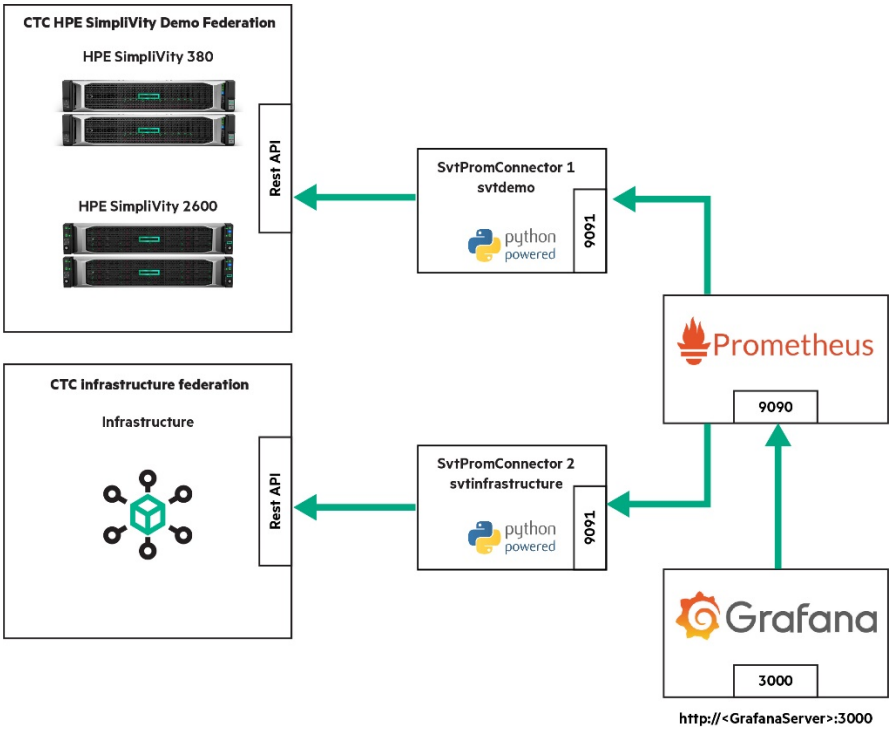

FIGURE 2. Logical diagram of the HPE SimpliVity monitoring environment of the HPE CTC, Böblingen

DESIGN PRINCIPLES

A flexible and adjustable connector was the design goal. The HPE SimpliVity system provides many metrics (capacity, performance, status, and many more) via a REST API, but the output of the REST API calls needed to be converted into the format that the Prometheus collector expects. The availability of a complete Prometheus client for Python and the HPE SimpliVity Rest API Python class library, developed for previous automation projects, made it easy to decide that Python would be the base scripting language of the connector.

The connector itself should have some adjustable input parameters such as the IP address that is used to connect to the HPE SimpliVity Federation, the encrypted credentials for the connection, and some measurement time parameters that are presented to the connector via a key and XML file-pair.

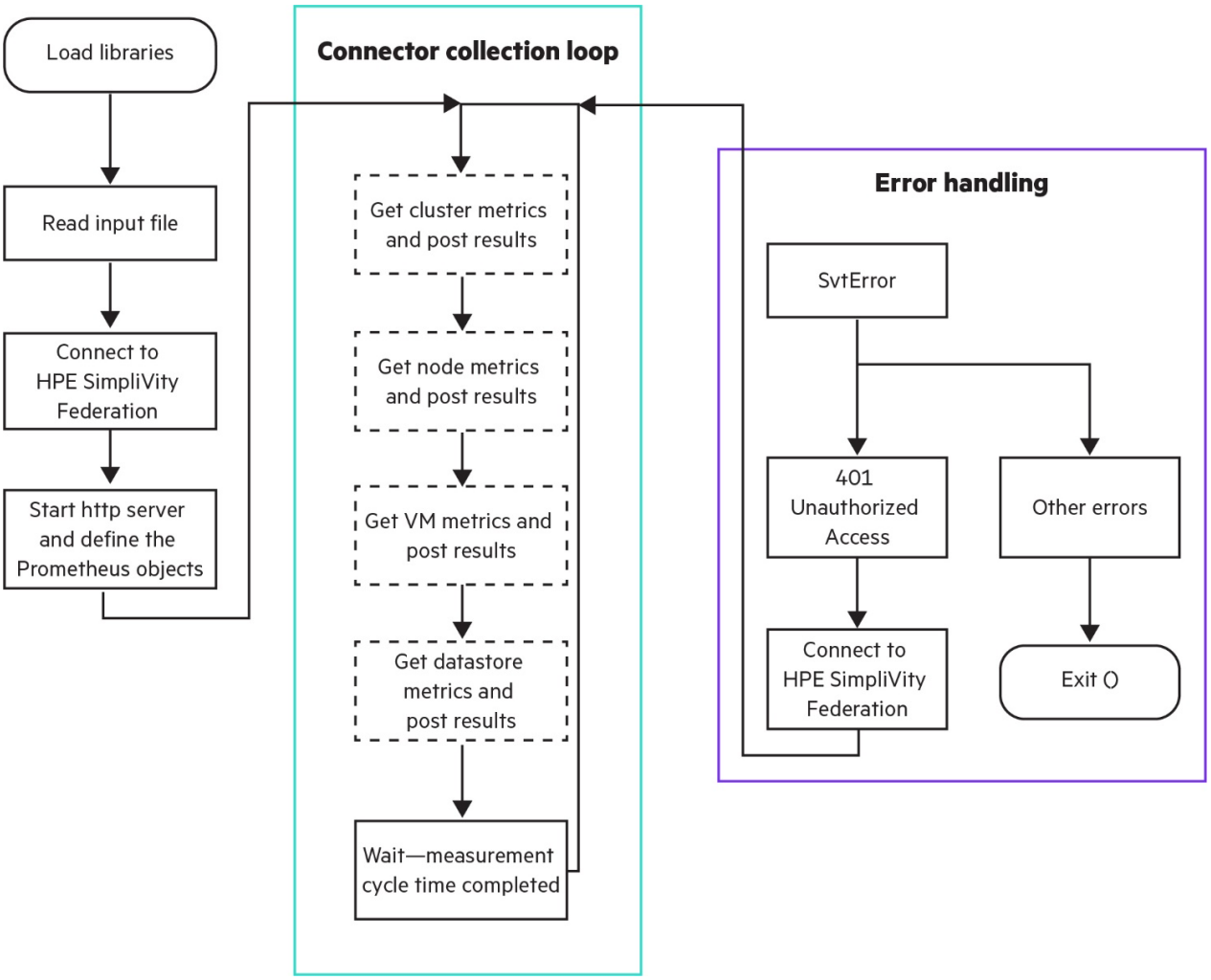

FIGURE 3. HPE SimpliVity Connector Process Flow

The connector reads these input files after the necessary Python libraries are loaded and before the first connection to the HPE SimpliVity system is established. In the next step, the http server is started to present the results to Prometheus and its objects will be defined before the measurement cycle loop is entered. Within the loop, the metrics are pulled via REST API calls from the HPE SimpliVity Federation and presented via the http server on the TCP/IP port defined in the XML input file. The only remaining challenge is that any HPE SimpliVity REST API access token expires either after 10 minutes of inactivity or latest after 24 hours, that is, the access token is to be refreshed at least once per day. The solution here is to implement an error handling for the possible errors thrown by the HPE SimpliVity REST API Python class library. If the unauthorized access errors (error code 401) are thrown, then a reconnection to the HPE SimpliVity Federation will take place. All other errors will result in a controlled exit of the program.

The total runtime of a single collection cycle obviously depends a lot on the number of entities (cluster, node, VMs, and so on) that should be collected. Therefore, the measurement cycle time needs to be adjusted according to the monitored environment. The connector will report the actual time spent for the last measurement cycle as a separate metric:

```
# HELP ConnectorRuntime Time required for last data collection in seconds
# TYPE ConnectorRuntime gauge
ConnectorRuntime 5.380947828292847
```

FIGURE 4. Connector runtime in seconds

Experienced Python developers can do an adjustment of the different metrics that should be collected by editing the script and commenting/uncommenting the different metric collection.

SOLUTION COMPONENTS

Hardware

Testing was performed on two HPE SimpliVity Federation, one with two 2-node cluster and one with a 3-node cluster.

HPE SimpliVity 380	Configuration
Memory	256 GB
CPU	2 x Intel® Xeon® Silver 4114 CPU (2.2 GHz, 10-core)
Networking	Storage and federation: 10GbE Management: 1GbE
Storage	5 x 1920 GB SSD—Small
HPE OmniStack software version	3.7.9

HPE SimpliVity 2600	Configuration
Server	2 x HPE SimpliVity 170r Gen10 Server
Memory	384 GB
CPU	2 x Intel® Xeon® Gold 5118 CPU (2.3 GHz, 12-core)
Networking	Storage and federation: 10GbE Management: 1GbE
Storage	6 x 1920 GB SSD—Small
HPE OmniStack software version	3.7.9

Software

The following software was used to build the solution:

Software component	Version
Docker Community Edition	18.09.2
HPE SimpliVity Rest API	3.7.9
Prometheus	2.12.0
Grafana	5.4.3
Python	3.6
Ubuntu Linux®	18.04

BEST PRACTICES AND CONFIGURATION GUIDANCE FOR THE SOLUTION

The monitoring environment can be built on a separate physical server or virtual machine or as a fully containerized solution. The CTC in Böblingen, Germany, decided to deploy the monitoring environment as container. And the rest of the document will describe this implementation approach. If you decide to deploy it as a virtual machine or physical server, then consult the corresponding Prometheus and Grafana documentation. The following components are required to build the containerized solution:

- A Docker container runtime environment
- Prometheus container image
- Grafana container image
- HPE SimpliVity connector and create credentials scripts

We will assume that you do have a Docker container runtime environment available and will describe the steps necessary to implement the HPE SimpliVity monitoring environment as it is highlighted in Figure 2. Each HPE SimpliVity connector, Prometheus, and Grafana will run as a separate container.

Application network

The HPE SimpliVity connector together with the Prometheus and Grafana container are building the monitoring app. It is best practice to limit container cross talk to the necessary minimum by running different apps on separate networks. Therefore, a bridged network for the monitoring app was created first:

```
# docker network create svtmonitor
```

Each container that is part of the monitoring app will use this network.

HPE SimpliVity credentials

A Python script (CreateCredentials.py) is used to create the necessary input parameter for the HPE SimpliVity connector. This script can run on any Python 3 system that does have the necessary Python packages (fernet, getpass, and lxml) installed.

```
# python CreateCredentials.py
```

The script will ask for the necessary input parameters:

- User name and password credentials for connecting into the HPE SimpliVity Federation,
- The time range in seconds
- Resolution (seconds or minutes)
- Monitoring interval in seconds
- Logfile name
- http server port that should be used
- OVC IP address
- Filename

The connector will capture the data in every monitoring interval for the given past time range with the defined resolution and it will build an average of all available data points for the given time range.

Two files (filename.xml and filename.key) are the output of the CreateCredentials.py script. The first one, <filename>.xml (see page 10), contains the input parameter (user name and password encrypted) for the HPE SimpliVity connector. The other one, <filename>.key, provides the key that is needed to decrypt the encrypted user name and password.

Container images

Next, the official Prometheus, Grafana, and Ubuntu images need to be pulled from the Docker hub:

```
# docker pull prom/Prometheus

# docker pull grafana/grafana

# docker pull ubuntu
```

The HPE SimpliVity connector Docker image is based on an Ubuntu image where Python with the necessary packages is installed. Also, the HPE SimpliVity connector script together with the input files are copied into the image and the script itself is started, once the HPE SimpliVity connector container is booted.

We decided to have separate container for each monitored HPE SimpliVity Federation since we do use different credentials on each federation. Each container was built using a Dockerfile (see Appendix A: HPE SimpliVity connector—Docker file) where the only difference is the provided input files (<filename>.xml and <filename>.key). The following commands were used to create the HPE SimpliVity connector images for the HPE SimpliVity CTC demo and the CTC infrastructure federation:

```
# docker build -t svtdemo -f svtconnect.dockerfile

# docker build -t infrastructure -f infrastructure.dockerfile
```

The only difference between the two dockerfiles svtconnect.dockerfile and infrastructure.dockerfile are the different connector input files (*.key and *.xml) for the two federations. As a result, we do have now a separate connector image for each federation we wanted to monitor.

Prometheus configuration file

The Prometheus configuration file contains entries for every single process or service that should be monitored. In our case, we do have a separate entry for each HPE SimpliVity Federation that should be monitored. The entry for the HPE SimpliVity demo environment of the CTC looks like:

```
- job_name: simpliVity

  honor_timestamps: true

  scrape_interval: 20s

  scrape_timeout: 10s

  metrics_path: /metrics

  static_configs:

      - targets: ['simplivity:9091']
```

The target is defined as <connector container name>:<port>; that is, in the previous example, the connector container for the HPE SimpliVity CTC demo environment has the name **simpliVity** and provides the collected metrics on port 9091. The Prometheus job for the CTC infrastructure HPE SimpliVity Federation looked similar to a different target. The job name, scrape interval and timeout, the metrics path, and the targets define each Prometheus scrape job. The previous example is defining that the metrics of the HPE SimpliVity system should be collected every 20s from the address simplivity:9091/metrics and the collection times out if, it takes longer than 10s. It is possible to define multiple targets per job but we decided to define multiple job entries in order to be more flexible in the settings. See Appendix B: Prometheus Configuration File for the complete Prometheus configuration file is used to monitor the HPE SimpliVity Federations in the CTC Böblingen.

Persistent Storage

Prometheus stores the metrics in its time series database. In order to have the data persistence across container reboots or container moves, it is necessary to store the database on persistent storage. There are multiple options available to provide persistent storage to a container—one can use HPE 3PAR or HPE Nimble Storage arrays together with the corresponding Docker volume plug-in or use an NFS fileserver to provide persistent storage to a Docker container.

We used an NFS file share (10.0.44.44:/docker/prometheus) provided by the CTC infrastructure file server as persistent storage in order to be independent of the CTC demo environment where HPE 3PAR or HPE Nimble Storage array would be available too. A persistent Docker volume named prom-data, based on the infrastructure fileserver at IP address 10.0.44.44, was defined with the following command:

```
# docker volume create --driver local -o type=nfs -o device:/docker/prometheus -o o=addr=10.0.44.44
prom-data
```

Starting the container

Once the container images and the persistent volume are prepared, the Docker container can be started:

1. Start the HPE SimpliVity connector container for the CTC demo federation

   ```
   # docker run -d --restart unless-stopped --name svtdemo --network svtmonitor simplivity
   ```

2. Start the HPE SimpliVity connector container for the CTC infrastructure federation

   ```
   # docker run -d --restart unless-stopped --name infrastructure --network svtmonitor
   svtinfrastructure
   ```

3. Start the Prometheus container

   ```
   # docker run -d --restart unless-stopped --name prometheus --network svtmonitor \

       -v prom-data:/prometheus-data -p 9090:9090 prom/prometheus \

       --config.file=/prometheus-data/prometheus.yml \

       --storage.tsd.path=/prometheus-data/db
   ```

4. Start the Grafana container

   ```
   # docker run -d  --restart unless-stopped --name grafana --network prometheus -p 3000:3000
   grafana/grafana
   ```

We did start all container with the restart unless-stopped flag, to make sure, even if for some reason the container is stopped, that it is restarted. The collected data is now available on the server, where the container is running, at the following web addresses:

- Prometheus: https://<hostname>:9090
- Grafana: https://<hostname>:3000

During the initial testing we did make even the connector output available by using the -p option of the Docker run command, but once the system is running without any issues it is not necessary to export the IP port used by the Docker container to an external host port.

The only step that is missing is to define the Grafana dashboards to visualize the collected data. Some example dashboards (Federation Overview, Cluster Overview, and Node Overview) can be retrieved from the HPE SimpliVity Prometheus Connector GitHub location. Nevertheless, it is still possible either to customize the example dashboards or to create new ones. Review the Grafana documentation for the details on how to build the Grafana dashboards.

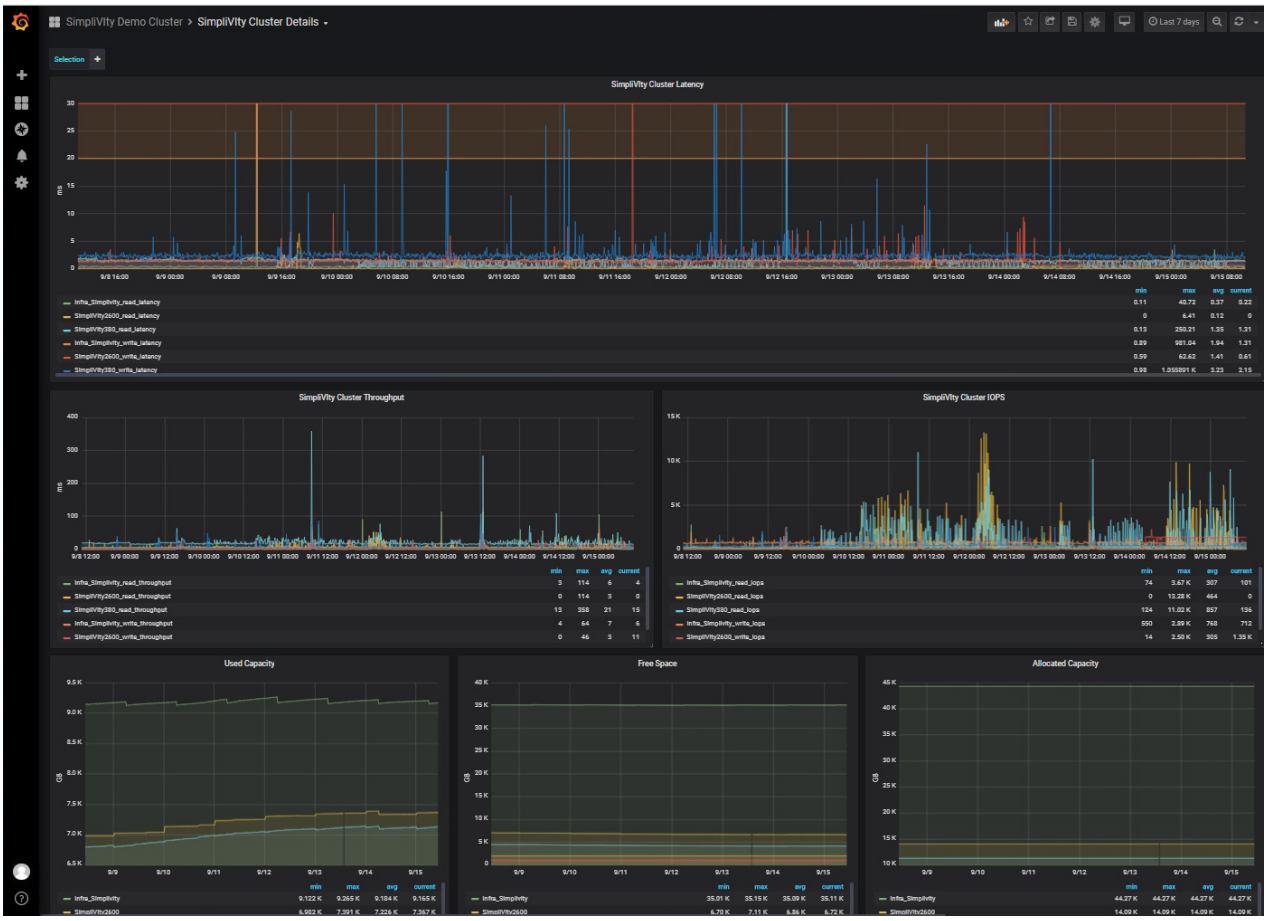

FIGURE 5. HPE SimpliVity Cluster Details Dashboard

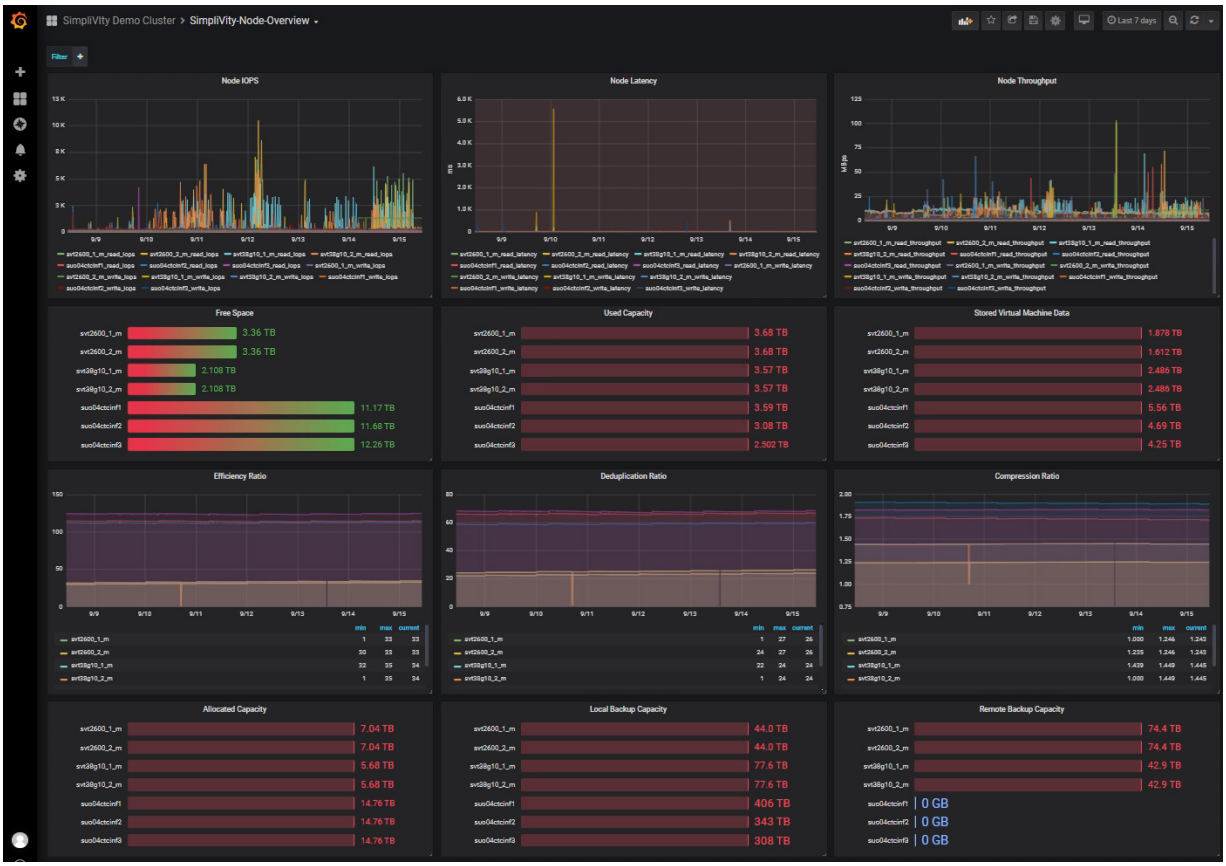

FIGURE 6. HPE SimpliVity Node Overview Dashboard

Metrics and data collected

The connector collects data for HPE SimpliVity cluster, HPE SimpliVity node (host), and HPE SimpliVity virtual machine from HPE SimpliVity APIs and process the data into metric information to Prometheus, which in turn are used for Grafana dashboards.

The metrics that are available for the dashboards include:

- Free space
- Allocated capacity
- Capacity savings
- Used capacity
- Used logical capacity
- Local backup capacity
- Remote backup capacity
- Stored compressed data
- Stored uncompressed data
- Stored virtual machine data
- Compression ratio
- Deduplication ratio
- Efficiency ratio
- Read IOPS
- Write IOPS

- Read throughput
- Write throughput
- Read latency
- Write latency

Analysis and recommendations

The data collection and processing takes some time and is the critical piece that might require an adjustment of the Prometheus scrape interval to get reasonable data. The Prometheus connector script will average the performance metrics for the time range given in input file. Nevertheless, it will be necessary to monitor the data collection time. If the data collection time is larger than the Prometheus scrape interval time, then the data collection or the Prometheus parameters need to be adjusted. Even in our relatively small HPE SimpliVity demo federation with two clusters, four nodes, and approximately 70 VMs, the connector data collection time exceeded the 20s monitoring interval and Prometheus scrape interval. We decided to skip at the moment the VM and datastore detail metrics, to get to a data collection time in the range of 12s. If there is a need to collect the VM metrics, then either we would need to increase the monitoring and scrape interval accordingly or the other alternative would be to add another connector container that is, collecting only the VM metrics data. We will go and test the feasibility of the second option.

In general, the ongoing centralized monitoring of the environment already helped us once to recognize a container that was creating an unexpected load by simply adding Grafana alerts after establishing a load baseline.

SUMMARY

Having an end-to-end view of a container DevOps environment will help to manage your environment efficiently. The solution presented in this white paper closed the existing gap for monitoring the HPE SimpliVity cluster with Prometheus and Grafana, the de facto standard open-source monitoring tools for containerized solutions. It presented a flexible Prometheus connector, based on the HPE SimpliVity Rest API that can be easily adjusted to the current needs.

This technical white paper describes solution testing performed in July and August 2019.

APPENDIX

Appendix A: HPE SimpliVity connector—Docker file

```
# Dockerfile to build a SimpliVity Connector container
# docker build -t svtconnector -f svtPrometheusConnector.Dockerfile .
FROM ubuntu
#
LABEL maintainer="Hewlett Packard Enterprise"
LABEL version="1.0"
LABEL copyright="Hewlett Packard Enterprise, 2019"
LABEL license="GNU General Public License v3"
LABEL DESCRIPTION="CTC SimpliVity Python container based on Ubuntu"
#
RUN apt-get update
RUN apt-get -y install python3.6 && \
    apt-get -y install python3-pip && \
    apt-get -y install vim && \
    apt-get -y install cron
RUN /usr/bin/pip3 install requests && \
    /usr/bin/pip3 install fernet && \
    /usr/bin/pip3 install cryptography && \
```

```
    /usr/bin/pip3 install lxml && \
    /usr/bin/pip3 install prometheus_client
# copy the necessary python files to the container
RUN mkdir /opt/svt
COPY SimpliVityClass.py /opt/svt
COPY svtPromConnector.py /opt/svt
COPY SvtPromConnector.key /opt/svt
COPY SvtPromConnector.xml /opt/svt
# Start the collector
CMD /usr/bin/python3.6 /opt/svt/svtPromConnector.py
```

Appendix B: Prometheus Configuration File

```
global:
  scrape_interval: 15s
  scrape_timeout: 10s
  evaluation_interval: 15s
alerting:
  alertmanagers:
  - static_configs:
    - targets: []
    scheme: http
    timeout: 10s
    api_version: v1
rule_files:
  # - "first.rules"
  # - "second.rules"

scrape_configs:
- job_name: prometheus
  honor_timestamps: true
  scrape_interval: 15s
  scrape_timeout: 10s
  metrics_path: /metrics
  scheme: http
  static_configs:
  - targets:
    - localhost:9090
```

```
- job_name: simpliVity
  honor_timestamps: true
  scrape_interval: 20s
  scrape_timeout: 10s
  metrics_path: /metrics
  static_configs:
      - targets: ['simplivity:9091']

- job_name: grafana
  honor_timestamps: true
  scrape_interval: 15s
  scrape_timeout: 10s
  metrics_path: /metrics
  static_configs:
      - targets: ['grafana:3000']

- job_name: infrastructure
  honor_timestamps: true
  scrape_interval: 20s
  scrape_timeout: 10s
  metrics_path: /metrics
  static_configs:
      - targets: ['svtinfrastructure:9091']
```

Appendix C: Connector input file

```
<data>
  <user>encryptedUserName</user>
  <password>encryptedPassword</password>
  <ovc>10.0.40.15</ovc>
  <timerange>20</timerange>
  <resolution>SECOND</resolution>
  <monitoringintervall>5</monitoringintervall>
  <logfile>svtPromCollector.log</logfile>
  <port>9091</port>
</data>
```

Resources and additional links

HPE SimpliVity

HPE SimpliVity Rest API

Digital transformation services

Prometheus

Grafana

HPE SimpliVity and Prometheus connector

HPE Reference Configuration for Red Hat OpenShift Container Platform (OCP) on HPE SimpliVity

HPE Express Containers with Docker Enterprise Edition on HPE SimpliVity

LEARN MORE AT

hpe.com/info/simplivity

a50000514ENW, January 2020

Hewlett Packard Enterprise