Open-Source Monitoring of HPE SimpliVity

How-to connect HPE SimpliVity to Prometheus

Technical white paper

Contents

Executive summary

Audience level: Executive, Technical, Sales, PreSales

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

At the same time CaaS platforms do have specific monitoring requirements that are addressed by the Open-Source monitoring solution Prometheus (www.prometheus.io). Prometheus together with Grafana (https://grafana.com/) for the visualization (see Figure 1) is the defacto standard for monitoring containerized environments. While there is a long list of exporters and integrations (https://prometheus.io/docs/instrumenting/exporters/) for multiple systems available, an integration to monitor HPE SimpliVity cluster and nodes was missing up to now.

Figure 1 SimpliVity Federation Overview Dashboard

This whitepaper describes a Prometheus connector for HPE SimpliVity, that can still be customized as needed and 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 manager, enterprise architects and deployment engineers and others that 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 in order to get an integrated container and infrastructure monitoring system in place.

Solution overview

Audience level: Technical, PreSales, Enterprise Architects and CTOs

All status and performance metrics of a HPE SimpliVity system are available via Rest API calls (see https://developer.hpe.com/platform/hpe-simplivity/home for a complete list). The connection between 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 a SimpliVity Federation and present it via a configurable TCP/IP port (port 9091 in the example in

Figure 2 Logical diagram of the SimpliVity monitoring environment of the HPE Customer Technology Center (CTC), Böblingen). If multiple SimpliVity Federation need to be monitored, then a separate connector is needed per federation. Prometheus is collecting metrics from multiple sources, defined in the prometheus.yml file, and stores it in the Prometheus timeseries 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. Prometheus and Grafana are available as container images and the Docker compose file to build the SimpliVity connector as a container is provided in

Appendix

Appendix A: SimpliVity Connector - Dockerfile. The SimpliVity monitoring environment of the Customer Technology Center (CTC) in Böblingen, Germany, is fully containerized and runs in a Kubernetes container orchestration environment. Only the SimpliVity monitoring part of this environment is shown in the logical diagram in Figure 2, additional CTC monitoring pieces are added at the time this whitepaper is written.

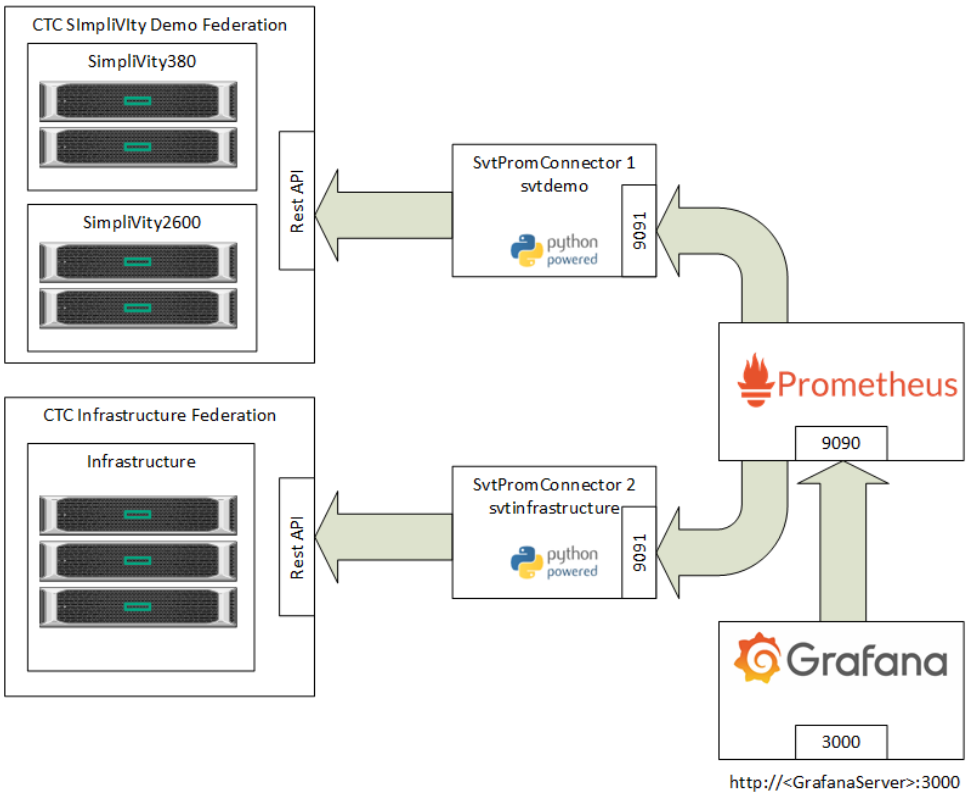

Figure 2 Logical diagram of the SimpliVity monitoring environment of the HPE Customer Technology Center (CTC), Böblingen

Design principles

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

The connector itself should have some adjustable input parameters, like 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.

The connector reads these input files after the necessary Python libraries are loaded and before the first connection to the

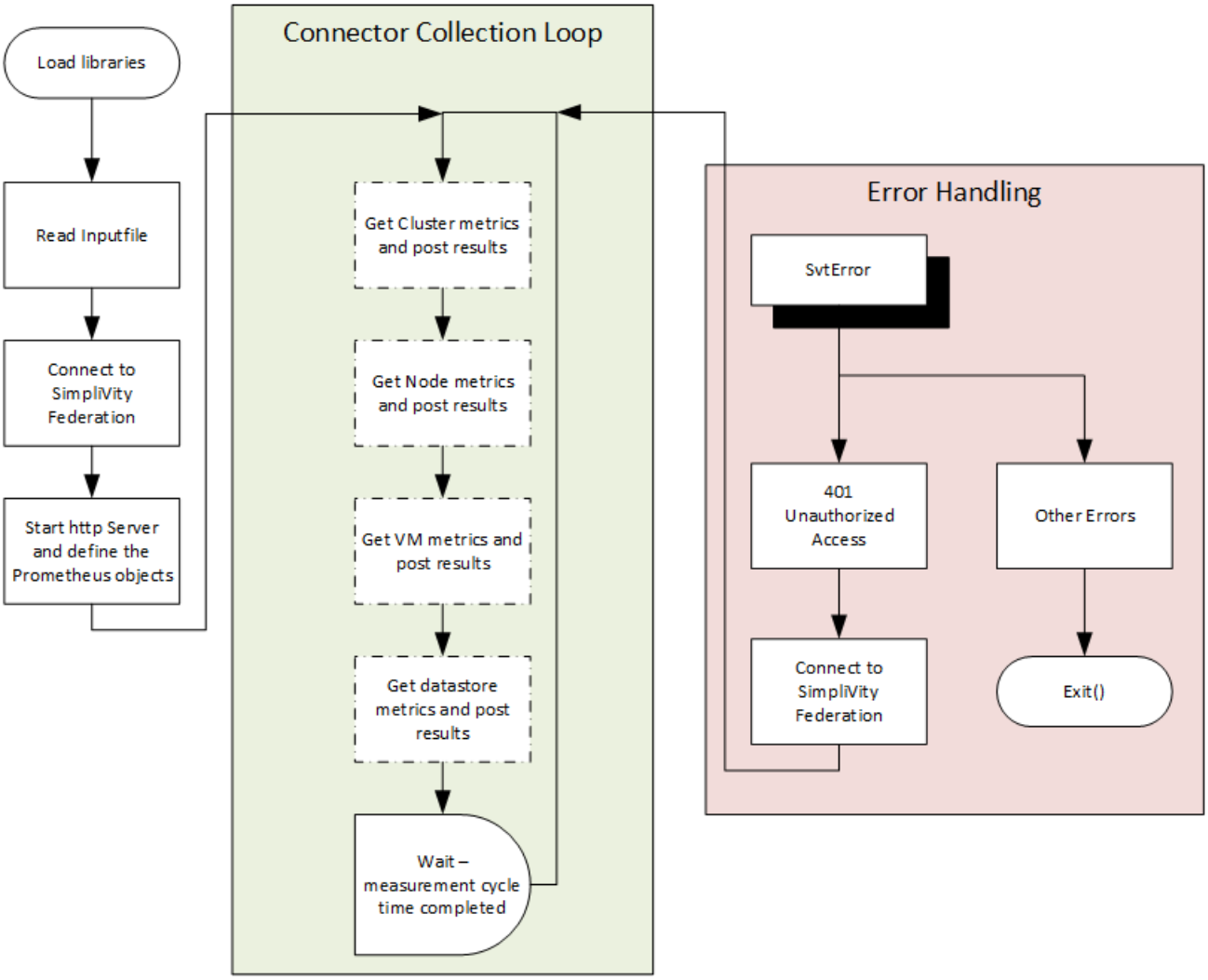

Figure 3 SimpliVity Connector Process Flow

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

The total run time of a single collection cycle obviously depends a lot on the number of entities (cluster, node, VMs, etc.) 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 spended 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

An adjustment of the different metrics, that should be collected, can be done by experienced Python developers by editing the Python script, i.e. commenting and uncommenting the different metric collection.

Solution components

Hardware

Testing was performed on two SimpliVity Federation, one with two two-node-cluster and one with a three-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: 10 GbE Management: 1 GbE
Storage	5 x 1920 GB SSD – Small
OmniStack Version	3.7.9

HPE SIMPLIVITY 2600	CONFIGURATION
Server	2 x XL170R
Memory	384 GB
CPU	2 x Intel Xeon Gold 5118 CPU (2.3 GHz, 12-core)
Networking	Storage and Federation: 10 GbE Management: 1 GbE
Storage	6 x 1920 GB SSD – Small
OmniStack Version	3.7.9

Software

The following software was used to build the solution:

SOFTWARE COMPONENT	VERSION
Docker Community Edition	18.09.2
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 build on a separate physical server or virtual machine or as a fully containerized solution. The Customer Technology Center in Böblingen, decided to deploy the monitoring environment as container. And the rest of the document will describe the this implementation approach. If you decide to deploy it as a virtual machine or

physical server, then please consult the corresponding Prometheus und Grafana documentation. The following components are required to build the containerized solution:

- A Docker container run time environment
- Prometheus container image
- Grafana container image
- SimpliVity Connector and CreateCredentials scripts

We will assume that you do have a Docker container run time environment available and will describe the steps necessary to implement the SimpliVity monitoring environment as it is highlighted in

Figure 2 Logical diagram of the SimpliVity monitoring environment of the HPE Customer Technology Center (CTC), Böblingen. Each SimpliVity connector, Prometheus and Grafana will run as a separate container.

Application Network
The 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.

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

```
# python CreateCredentials.py
```

The script will ask for the necessary input parameter:

- username and password credentials for connecting into the 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 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 13), contains the input parameter (username and password encrypted) for the SimpliVity connector. The other one, <filename>.key, provides the key that is needed to decrypt the encrypted username 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 SimpliVity connector docker image is based on an Ubuntu image where Python with the necessary packages is installed. Additionally, the SimpliVity Connector script together with the input files are copied into the image and the script itself is started, once the SimpliVity connector container is booted.

We decided to have separate container for each monitored SimpliVity federation, since we do use different credentials on each federation. Each container was built using a dockerfile (see Appendix A: SimpliVity Connector - Dockerfile) were the only difference are the provided input files (<filename>.xml and <filename>.key). The following commands were used to create the SimpliVity connector images for the CTC SimpliVity demo federation 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 inputfiles (*.key and *.xml) for the two federation. 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 SimpliVity federation that should be monitored. The entry for the 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>; i.e. in the above example, the connector container for the CTC Simplivity demo environment has the name simplivity and provides the collected metrics on port 9091. The Prometheus job for the CTC infrastructure SimpliVity federation looked similar with a different target. Each Prometheus scrape job is defined by the job name, scrape interval and timeout, the metrics path and the targets. The above example is defining that the metrics of the SimpliVity system should be collected every 20s from the address http://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

Appendix B: Prometheus Configuration File for the complete Prometheus configuration file used to monitor the SimpliVity federations in the CTC Böblingen.

Persistent Storage
Prometheus stores the metrics in its timeseries database.In order to have the data persistent 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 plugin or one can use a NFS fileserver to provide persistent storage to a docker container.

We used an NFS fileshare (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 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 SimpliVity connector container for the CTC demo federation

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

2. Start the 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 are 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 now 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 SimpliVity Prometheus Connector Github location. Nevertheless, it is still possible either to customize the example dashboards or to create new ones. Please take a look at the Grafana documentation on the details on how to build Grafana dashboards.

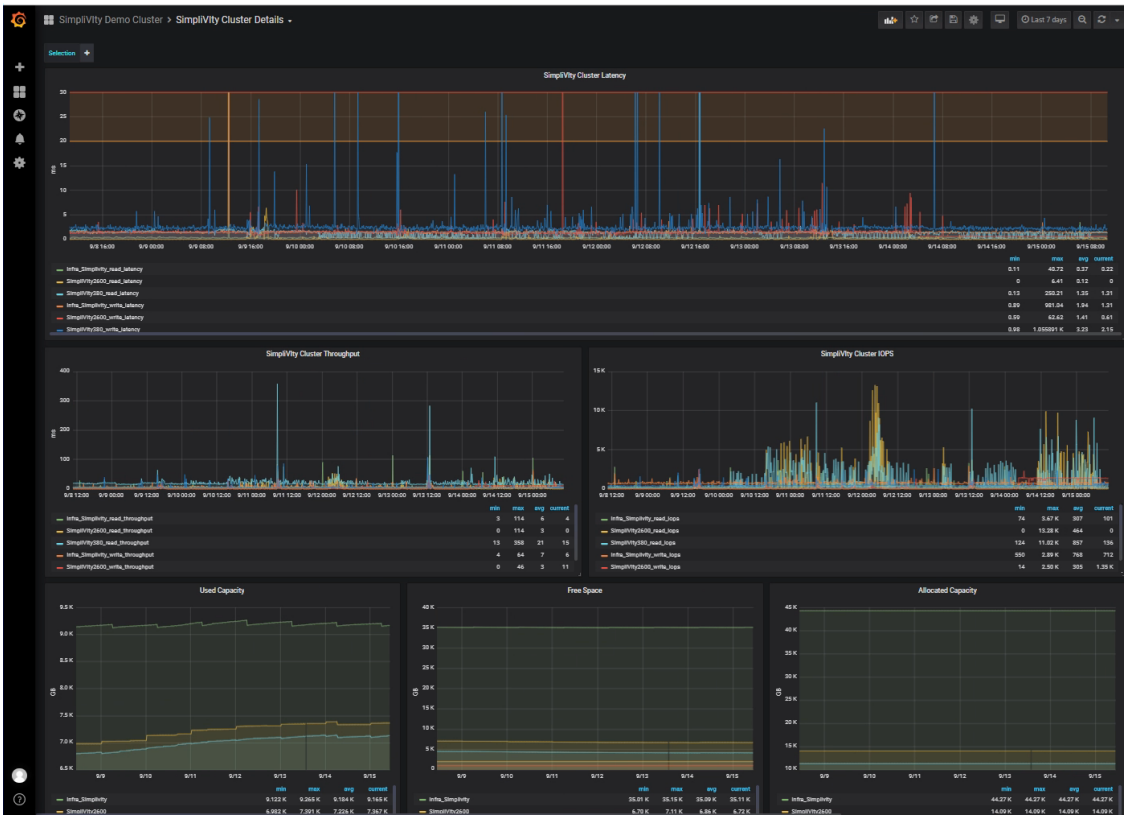

Figure 5 SimpliVity Cluster Details Dashboard

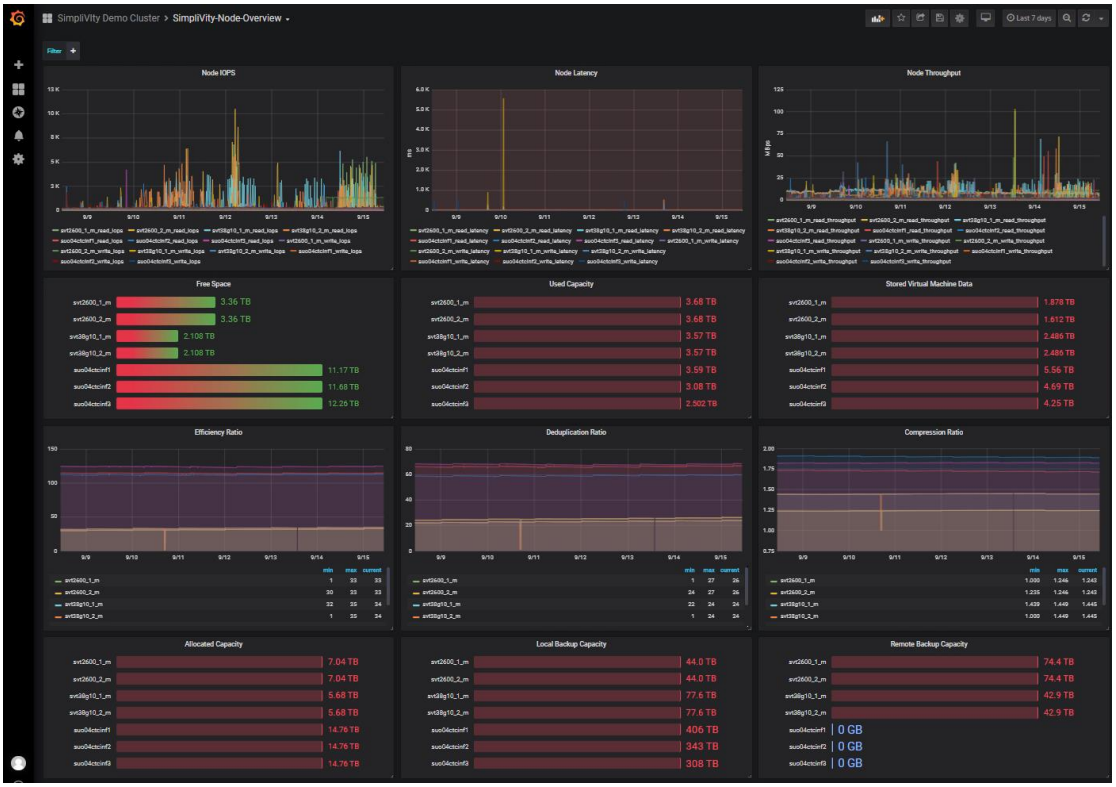

Figure 6 SimpliVity Node Overview Dashboard

Metrics and Data Collected

The connector collects data for Simplivity Cluster, SimpliVity Node(Host) and SimpliVity Virtual Machine from SimpliVity APIs and process the data into metric information to prometheus, which inturn 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 timerange given in inputfile. 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 SimpliVity demo federation with 2 clusters, 4 nodes and approx. 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 12 seconds. If there is a need to collect the VM metrics, then we either 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 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 whitepaper closed the existing gap for monitoring the HPE SimpliVity cluster with Prometheus and Grafana, the defacto standard open source monitoring tools for containerized solutions. It presented a flexible Prometheus connector, based on the 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: SimpliVity Connector - Dockerfile

```
# Dockerfile to build a SimpliVity Connector container
# docker build -t svtconnector -f svtPrometheusConnector.Dockerfile .
FROM ubuntu
#
LABEL maintainer="Thomas Beha"
LABEL version="2.0"
LABEL copyright="Thomas Beha, 2019"
LABEL license="GNU General Public License v3"
LABEL DESCRIPTION="CTC SimpliVity Pythone 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.com/info/simplivity

HPE SimpliVity Rest API
https://developer.hpe.com/api/simplivity/

HPE Technology Consulting Services
hpe.com/us/en/services/consulting.html

Prometheus
https://prometheus.io/

Grafana
https://grafana.com/

SimpliVity Python Library
https://github.com/tbeha/SimpliVity-Python

SimpliVity Prometheus Connector
https://github.com/tbeha/SimpliVity-Prometheus

HPE Reference Configuration for RedHat OpenShift Container Platform (OCP) on HPE SimpliVity
https://github.com/HewlettPackard/Openshift-on-SimpliVity

HPE Express Containers with Docker Enterprise Edition on HPE SimpliVity
https://hewlettpackard.github.io/Docker-SimpliVity/

Learn more at
hpe.com/info/simplivity

December 2019, Rev. 1.0

Hewlett Packard Enterprise