Telemetry Framework Documentation

Release 1.0.0

Red Hat

Aug 20, 2019

Contents

1 Get The Code **3**

2 Contents **5**

2.1 Why Telemetry Framework? 5
2.2 Architecture 5
2.3 Platform Installation 6
2.4 Telemetry Framework Installation 12
2.5 Client Installation 13
2.6 Indices and tables 13

The telemetry framework is a system that allows for collection of metrics and events using collectd (or other collection methods that support an AMQP 1.x Proton client connection), sending data streams across an AMQP 1.x message bus back to the server side for storage (such as Prometheus, ElasticSearch, etc).

Once data is stored, metrics and events can be used as the sources for alerts, visualization, or the source of truth for orchestration frameworks.

Since the project is a framework, you can also add, remove, or replace components within the telemetry deployment. For example, you might add Grafana for visualization, or integrate third-party applications to leverage the data for automate decision making, or predicting long term trends.

About Documentation and Repository Structure

While the telemetry framework subscribes to an upstream-first mentality, it is also developed primarily against products within Red Hat. All of these products complimentary upstream projects. Within this documentation, we're going to be referencing to documentation sources that are possibly not applicable to you.

For example, the reference deployment currently in development uses the oVirt Hyperconverged system which is known as RHHI-V (Red Hat Hyperconverged Infrastructure for Virtualization). The telemetry framework has been tested against both the upstream and downstream versions of the application[1]. That goes also for the OKD/OpenShift (Kubernetes) platform, RHOSP (OpenStack) and the operating system RHEL (CentOS).

The documentation will be written from the perspective of installing from the downstream perspective, but we will do our best to provide sidebars pointing back to the equivalent upstream documentation.

Within the repository, we've also done our best to provide an upstream-centric deployment as well, but note that our primary focus is on support for the downstream products at this time. If you happen to find an area which is missing the upstream-first component (documentation, examples, scripts), please an issue and we'll get it resolved as quickly as possible.

---

[1] The telemetry framework is also known as a micro-service application, meaning that multiple containers are run and connected together using an orchestration platform.

CHAPTER 1

Get The Code

The source is available on GitHub.

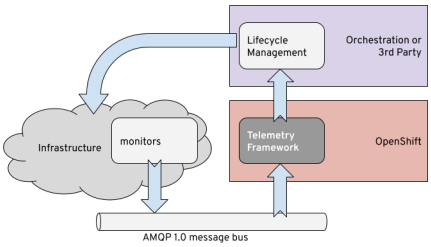
CHAPTER 2

Contents

2.1 Why Telemetry Framework?

In the modern datacenter, things are dynamic. Infrastructure needs to be monitored at a high polling interval, and actions taken quickly. Our environments are dynamic, constantly changing in the cloud era. An ability to monitor and react to these changing conditions with a strong degree of flexability is why the telemetry framework was created.

The telemetry framework is itself a dynamic application running atop OpenShift (Kubernetes) using several components such as Prometheus, the Smart Gateway, collectd and the Apache QPID Dispatch Router. With a strong desire for low latency in message delivery and high resolution information, the telemetry framework is an excellent core to building the information to automatically react to situations in your cloud infrastructure.

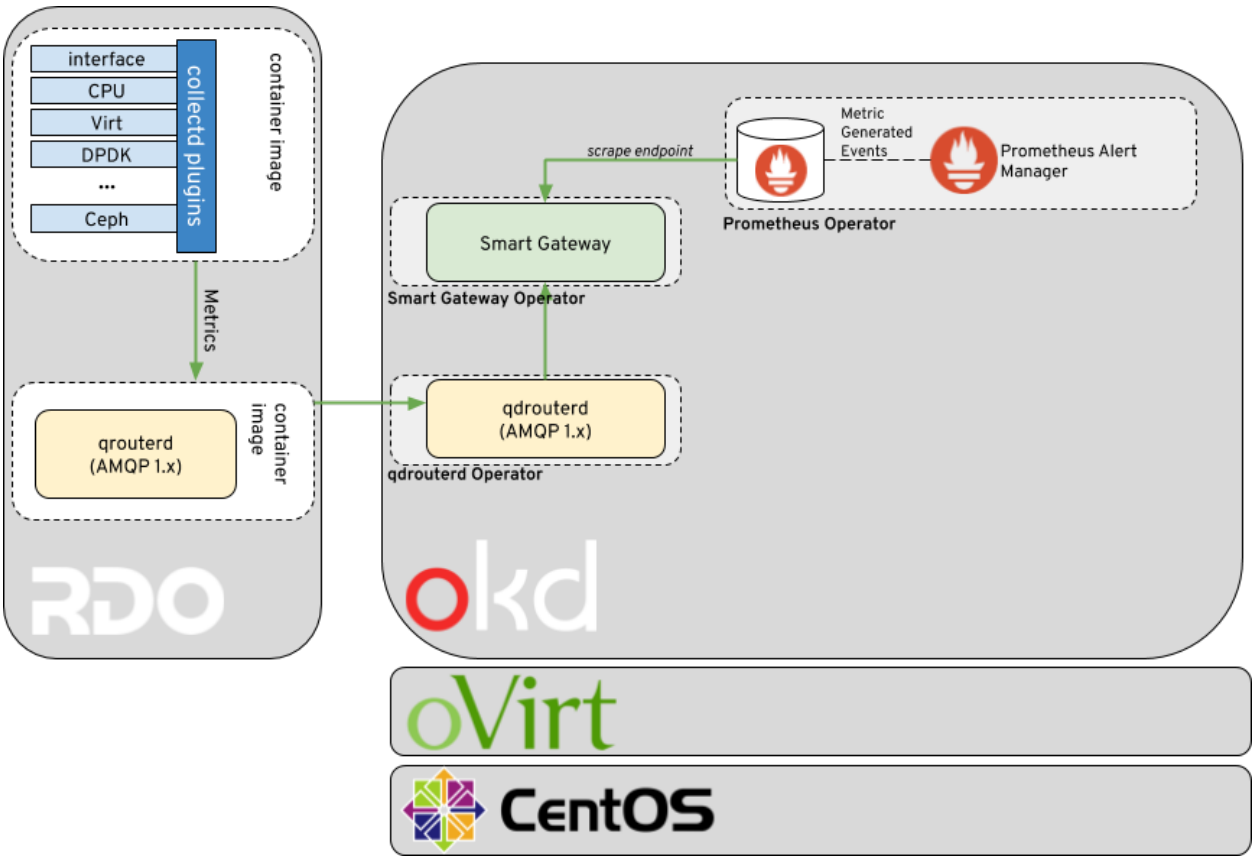


The telemetry framework gives you the core functionality to start gathering data, storing it, and then reacting to it. With Prometheus Alertmanager you can build rules that will create metrics-driven events that allow for notifications to various endpoints. The flexibility of the Prometheus time-series database can allow for predicting imminent failures before they happen.

You can add other systems as well to provide things like graphing, data correlation, traps, and automation.

2.2 Architecture

The telemetry framework is ultimately a metrics collection and storage tool leveraging Prometheus for the time-series data storage, and an AMQP 1.x compatible messaging bus for shuttling the metrics information back to Prometheus.



On the client side collectd is used for collecting the high resolution metrics. The metrics are then delivered to Prometheus using the AMQP1 plugin in order to place the data onto the message bus. On the server side, a small Golang application called the Smart Gateway takes the data stream from the bus, and exposes it as a local scrape endpoint for Prometheus.

In our reference architecture above, we are collecting data within an RDO OpenStack cloud from a containerized collectd and qdrouterd (Apache QPID Dispatch Router) that ships the metrics to a corresponding qdrouterd on the telemetry framework side.

The telemetry framework is a micro-service based application that runs within an OKD (OpenShift) environment. The deployment of OKD is typically run within a public or private cloud as virtual machines. Our setup is to leverage a hyperconverged deployment of oVirt in order to provide a minimal local environment for OKD while maintaining the high availability aspects that we desire. Our choice operating system is CentOS.

2.3 Platform Installation

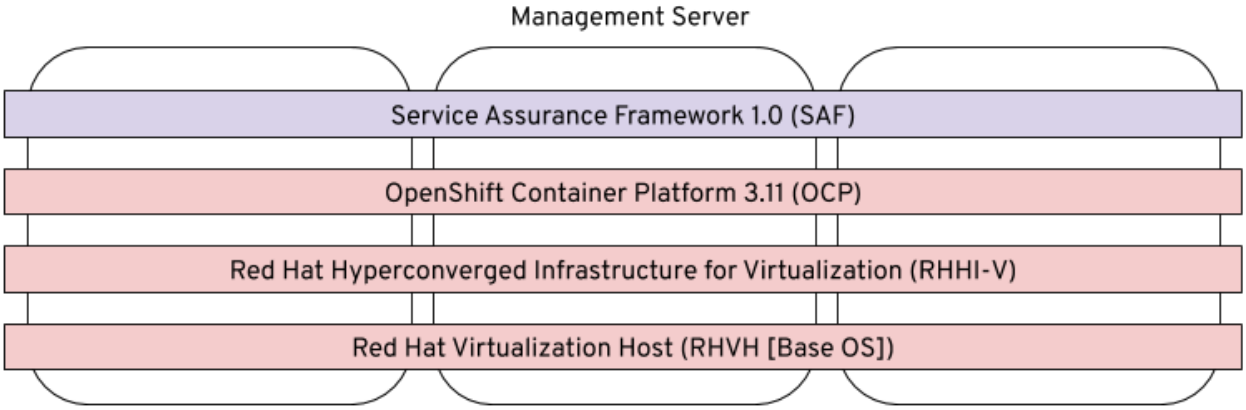
Why are you documentating the downstream version?

As we noted in the *Project Overview* we are documenting the downstream components initially for the deployment of the telemetry framework. We will note the upstream equivalent where possible.

The installation of the telemetry framework simply requires the deployment of OpenShift 3.11 or later and a bastion node for executing the supplied *bash* script to load the components into the *sa-telemetry* namespace.

Our reference implementation leverages 3 physical nodes using RHHI-V platform which provides us with virtualization and distributed storage.

The layers are shown in the following diagram.



Installation of RHVH, RHHI-V, and OpenShift are beyond the scope of this documentation but we'll link to existing documentation in the following sections, and describe any example configurations, or other useful information to make installation more reproducible.

We'll build up the platform layer by installing a 3 node RHHI-V on top of RHVH and then creating the virtual machines in RHHI-V, setting up affinity, networking, etc prior to deploying OpenShift.

2.3.1 Installing Red Hat Virtualization Host (RHVH) Operating System

Upstream equivalent of RHVH

RHVH is the Red Hat Virtualization Host, a spin of RHEL with the specific repositories required to install RHHI-V. Installation of CentOS 7.6 or later should contain the required dependencies for an upstream installation.

First we need to install RHVH (Red Hat Virtualization Host). This needs to be done on 1 or 3 nodes (or groups of 3) so that we can do the installation of RHHI-V (Red Hat Hyperconverged Infrastructure for Virtualization) next.

Your node will need at least 2 physical disks; one for the installation of the operating system, and one blank disk for the GlusterFS system. If you have additional disks you can use these in either a RAID configuration for GlusterFS, or break the storage classes GlusterFS will setup across multiple disks.

See the official Red Hat Virtualization Hosts installation guide for obtaining RHVH and installing it.

The following procedure matches our reference architecture. Your procedure is likely to differ in order to match your environment, but the steps to building out the environment is likely to follow what we're documenting.

Prepare nodes

- Boot the RHVH 4.2.8 (or later) ISO for the node, or somehow automate the installation of RHVH with Satellite or other.

Now that you have three nodes with RHVH installed on them, let's continue.

Prerequisite setup from bastian host

The bastian host is our initial terminal. It is likely your laptop, or some other dedicated jump-host in your environment.

- Set a shell variable for the hostname you're going to work with

```
export SAFTLD=management.service-assurance.tld
```

The next steps will setup the SSH connections. We have a trusted environment and are not checking the fingerprints of the servers. Your environment may not allow for this trust.

- Get host fingerprints and copy local SSH key onto the nodes

```
for i in 1 2 3; do \
  ssh-keyscan -H virthost$i.${SAFTLD} >> ~/.ssh/known_hosts; \
  ssh-copy-id -i ~/.ssh/id_rsa.pub root@virthost$i.${SAFTLD}; \
done
```

Prerequisite setup from the control host

Next we need to setup SSH connections into the local server and satisfy the subscriptions for the servers. If you don't need to subscribe the servers then you can skip that step.

Login to one of the three servers you're going to designate as the *control* server. If you pick the first server in your list (e.g. `virthost1`) then make sure when you login to the web interface to complete the installation you use that same server.

We're going to create an SSH keypair on the *control* server and then copy that into the `authorized_keys` file on the remote machines. When we do our installation through the cockpit UI, Ansible will be executed against those nodes from our *control* server, and if we can't SSH into the machines passwordless the Ansible-controlled install will fail.

- Login to the control machine

```
ssh root@virthost1.${SAFTLD}
```

- Create an ssh key and make sure you can ssh into localhost without a password

```
ssh-keygen -N '' -t rsa -b 2048 -q -f ~/.ssh/id_rsa
ssh-copy-id -i ~/.ssh/id_rsa.pub root@$(hostname -f)
```

- Next validate that the control machine can login to all networks that might be utilized during the installation process

---

Tip:

```
# this one is for copying the key to all systems with various subdomains
export SAFTLD=service-assurance.tld
for i in 1 2 3; do ssh-keyscan -H virthost$i.${SAFTLD} >> ~/.ssh/known_hosts ;␣
↪ssh-copy-id -i ~/.ssh/id_rsa root@virthost$i.${SAFTLD}; done
```

Prerequisite setup across all RHHI-V nodes

The rest of the commands need to be run on all systems.

- (optional) Possibly setup the `/etc/resolv.conf` to point at your local DNS server if necessary and `chattr +i /etc/resolv.conf` to lock it (needed in my lab, no control of the DHCP server)

Note: In a proper environment this will not be necessary, but DNS is very important for proper operation of the environment as a whole

- Register the nodes

```
export SKU_NAME="SKU Name"
subscription-manager register --username='<email>' --password='<password>'
subscription-manager list --available --matches="${SKU_NAME}" --pool-only
subscription-manager attach --pool=<pool from list>

# alternatively do in a single command using the first item returned
SKU_NAME="SKU Name" ; subscription-manager register --username='<email>' --
↪password='<password>' subscription-manager attach --pool=$(subscription-manager␣
↪list --available --matches="${SKU_NAME}" --pool-only | head -n1)
```

- Enable subscriptions and update systems

```
subscription-manager repos --enable=rhel-7-server-rhvh-4-rpms
yum update -y
```

- Wipe all the disks being used for GlusterFS to make sure they are blank.

Warning: Of course update the list of disk paths below to match your own environment.

  - Wipe the disks

```
wipefs -f -a /dev/sdb
wipefs -f -a /dev/sdc
wipefs -f -a /dev/sdd

# this will wipe out any master boot record (MBR) data
dd if=/dev/zero of=/dev/sdb bs=512 count=1 conv=notrunc
```

- Note the size of the disks in GB, which you'll need for the next section

```
fdisk -s /dev/sdd | awk '{$1=$1/(1024^2); print $1,"GB";}'

# or...
lsblk
```

Load the web interface to start installation

Go to the web interface on your control host at `https://virthost1.management.service-assurance.tld:9090` to start installation of RHHI-V.

2.3.2 Installing Red Hat Hyperconverged Infrastructure for Virtualization (RHHI-V)

Upstream equivalent of RHHI-V

More information about deploying oVirt hyperconverged in a 1 or 3 node configuration is available at oVirt Gluster-Hyperconverged documentation.

Official documentation for installation of RHHI-V can be found at Chapter 6. Configure Red Hat Gluster Storage For Hosted Engine Using The Cockpit UI

In the previous section titled *Installing Red Hat Virtualization Host (RHVH) Operating System* we mentioned the creation of SSH keys and populating them among the various hosts. That is required during the installation of RHHI-V since it executes Ansible from the web interface to build the virtual machine for the engine, and to create the GlusterFS storage domains.

Download RHEL 7.6 KVM Image Onto Bastian Host

With RHHI-V now installed, we need to download a copy of the RHEL 7.6 KVM image which will be the source operating system during installation of OpenShift. You can download the image from https://access.redhat.com.

Get your download link (which is time sensitive and must be obtained each time you wish to download a new image) and download it onto your bastian host (or any other host where you can install `guestfish` which we'll discuss next).

```
cd /tmp
curl 'http://access.cdn.redhat.com/...' -o rhel-server-7.6-x86_64-kvm.qcow2
```

Tip: For an upstream-only deployment, download CentOS 7.6 qcow2 image for your base image.

Modify RHEL 7.6 KVM Image and Upload To RHV-M Engine

Before importing the template into the system, you'll need to run some `guestfish` commands to strip out the default `192.168.122.1` nameserver which can cause us issues as we'll be running the virtual machines with bridged network interfaces.

```
yum install guestfish -y
systemctl start libvirtd.service
virt-edit --expr 's/nameserver 192.168.122.1//g' \
    -a /tmp/rhel-server-7.6-x86_64-kvm.qcow2 /etc/resolv.conf
```

We can now upload our modified virtual machine image to the RHV-M engine with `scp` or another method. Our Ansible for creating the `rhel76_template` in RHHI-V will expect the virtual machine image file to be located in the `/root` directory on the RHV-M engine.

```
scp /tmp/rhel-server-7.6-x86_64-kvm.qcow2 root@engine.rhhi-v.tld:/root
```

2.3.3 Installing OpenShift

Upstream equivalent of OpenShift

The upstream OpenShift project is known as OKD and is the Red Hat distribution of Kubernetes. Documentation for installation of OKD 3.11 is available at docs.okd.io.

Installation of OpenShift within RHHI-V is done with Ansible playbooks and roles as created within the `rhhi-v/` subdirectory of the `telemetry-framework` repository.

The first step is creating an inventory file that will result in the creation of the virtual machines within the RHHI-V environment and then subsequently execute `openshift-ansible` to install the OpenShift platform within the virtual machines.

Both upstream and downstream deployment methods use the same automation and the primary difference is that when installing the downstream version of OpenShift an extra variables file will be required that contains the information for registration of OpenShift and pulling the components from another repository.

Creating Virtual Machine and OpenShift Inventory Files

A few example inventory files for our lab configuration exist within the `rhhi-v/inventory/` directory. We'll create a new inventory configuration based on a working example.

On your bastian host clone the telemetry-framework repository and create a new directory in the `rhhi-v/inventory/` directory.

```
mkdir -p ~/src/github.com/redhat-service-assurance
cd ~/src/github.com/redhat-service-assurance
git clone https://github.com/redhat-service-assurance/telemetry-framework
cd telemetry-framework/rhhi-v
mkdir inventory/my_lab
```

With our new directory created, we need two inventory files to deploy our infrastructure on top of RHHI-V; `hosts.yml` and `openshift.yml`.

The `hosts.yml` file will contain the information required to instantiate and configure the virtual machines in RHHI-V in preparation for our OpenShift installation.

Virtual Machines Inventory File

The `hosts.yml` file is relatively long, but most of it is boilerplate that you can use to build out your own hosts file. An example inventory file is available in `rhhi-v/inventory/nfvha-lab/hosts.yml`.

OpenShift Inventory File

The `openshift.inventory` file is used by `openshift-ansible` to configure your OpenShift cluster. An example `openshift.inventory` file that works with the `hosts.yml` file for creating the virtual machines, is available in the `rhhi-v/inventory/nfvha-lab/openshift.inventory` file.

Create Variables Files

We need a file that contains our login information for registering our RHEL virtual machines.

Importing RHEL 7.6 Template Into RHHI-V

```
ansible-galaxy install -r requirements.yml
ansible-playbook -i inventory/nfvha-lab/ \
     --ask-vault-pass playbooks/rhel-template.yml
```

Instantiate The OpenShift Cluster on RHHI-V

```
# make sure you edit your inventory files first
cd telemetry-framework/rhhi-v/
ansible-playbook -i inventory/nfvha-lab/ \
    --ask-vault-pass -e "@./vars/rhsub.vars" playbooks/vm-infra.yml
```

2.4 Telemetry Framework Installation

Once you have your OpenShift environment setup you can install the core components of the telemetry framework.

Installation of telemetry framework is handled through the use of Kubernetes spec objects that you load into OpenShift via the *oc* console (or web interface should you so desire). These spec objects set the desired state for the object, for example, creating a Deployment. The spec objects for the telemetry framework can be found in the *deploy/* directory of the *telemetry-framework* repository hosted on GitHub.

These spec objects then load the various Operators into memory. We can then load some additional spec objects into memory where the Operators will start to deploy our various application components for us, and managing their lifecycle, application configurations, etc.

2.4.1 Installation

There are 3 core components to the telemetry framework:

- Prometheus (and the AlertManager)
- Smart Gateway
- QPID Dispatch Router

Each of these components has a corresponding Operator that we'll use to spin up the various application components and objects.

Using the provided script

You can use the provided script in the *deploy/* directory of the *telemetry-framework* repository. Simply run the *deploy.sh* script with no arguments (or the *CREATE* argument) to instantiate the various components in your OpenShift deployment. If you want to remove the components you can supply the *DELETE* argument to the script.

Prior to running the *deploy.sh* script you must already be logged into OpenShift as an administrator, and have the *oc* application readily available in your *$PATH*. The script will do some basic checks to make sure this is true.

Additionally, the script will switch to the *sa-telemetry* namespace prior to deploying, and if it can't find that namespace, will attempt to create it.

To deploy telemetry framework from the script, simply run the following command after cloning the *telemetry-framework* repo into the following directory.

```
cd ~/src/github.com/redhat-service-assurance/telemetry-framework/deploy/
./deploy.sh CREATE
```

Deploying manually

If you want to deploy the components manually you can review the order-of-operations within the *deploy.sh* script for the recommended object creation order.

The high level order would be:

- deploy the Operators first (order does not matter)
- deploy the application components

Each of the Operators have a similar directory layout, and you should install the objects in this order:

- service account
- role
- role binding
- operator

For the applications they also have a similar layout and the recommended order is (with some objects not part of all application components):

- service account
- role
- role binding
- config map
- secret
- deployment
- route

2.5 Client Installation

2.6 Indices and tables

- genindex
- modindex
- search