Red Hat OpenStack Platform 16.1

Service Telemetry Framework 1.2

Installing and deploying Service Telemetry Framework 1.2
Installing and deploying Service Telemetry Framework 1.2

OpenStack Team
rhos-docs@redhat.com
Abstract

This guide contains information about installing the core components and deploying Service Telemetry Framework 1.2.
# Table of Contents

## CHAPTER 1. INTRODUCTION TO SERVICE TELEMETRY FRAMEWORK

1.1. SUPPORT FOR SERVICE TELEMETRY FRAMEWORK .............................................. 6
1.2. SERVICE TELEMETRY FRAMEWORK ARCHITECTURE ....................................... 6
1.3. INSTALLATION SIZE OF RED HAT OPENSHIFT CONTAINER PLATFORM .......... 9

## CHAPTER 2. PREPARING YOUR RED HAT OPENSHIFT CONTAINER PLATFORM ENVIRONMENT FOR SERVICE TELEMETRY FRAMEWORK ......................................................... 10

2.1. PERSISTENT VOLUMES .................................................................................. 10
  2.1.1. Ephemeral storage .................................................................................... 10
2.2. RESOURCE ALLOCATION ............................................................................ 11
2.3. METRICS RETENTION TIME PERIOD .............................................................. 11
2.4. NODE TUNING OPERATOR ......................................................................... 11

## CHAPTER 3. INSTALLING THE CORE COMPONENTS OF SERVICE TELEMETRY FRAMEWORK .............................................................. 13

3.1. DEPLOYING STF TO THE OCP ENVIRONMENT ................................................. 13
  3.1.1. Deploying STF to the OpenShift environment with ElasticSearch ................ 13
  3.1.2. Deploying STF to the OpenShift environment without ElasticSearch .......... 14
  3.1.3. Creating a namespace ............................................................................ 14
  3.1.4. Creating an OperatorGroup .................................................................... 14
  3.1.5. Enabling the OperatorHub.io Community Catalog Source ....................... 15
  3.1.6. Subscribing to the AMQ Certificate Manager Operator ............................ 15
  3.1.7. Subscribing to the Elastic Cloud on Kubernetes Operator ....................... 17
  3.1.8. Subscribing to the Service Telemetry Operator ....................................... 17
  3.2. OVERVIEW OF THE SERVICETELEMETRY OBJECT .................................... 18
    3.2.1. backends ............................................................................................... 18
      3.2.1.1. Enabling Prometheus as a storage backend for metrics .................... 19
      3.2.1.2. Enabling ElasticSearch as a storage backend for events ................ 19
    3.2.2. clouds ................................................................................................. 19
    3.2.3. alerting ................................................................................................. 20
    3.2.4. graphing ............................................................................................... 20
    3.2.5. highAvailability .................................................................................... 20
    3.2.6. transports ............................................................................................ 20
  3.3. CREATING A SERVICETELEMETRY OBJECT IN OCP .................................. 20
  3.4. REMOVING STF FROM THE OCP ENVIRONMENT ....................................... 23
    3.4.1. Deleting the namespace ....................................................................... 23
    3.4.2. Removing the CatalogSource ............................................................... 23

## CHAPTER 4. COMPLETING THE SERVICE TELEMETRY FRAMEWORK CONFIGURATION ................................................................. 25

4.1. DEPLOYING TO NON-STANDARD NETWORK TOPOLOGIES ................. 25
4.2. CONFIGURING RED HAT OPENSTACK PLATFORM OVERCLOUD FOR SERVICE TELEMETRY FRAMEWORK .............................................. 26
  4.2.1. Retrieving the AMQ Interconnect route address .................................... 26
  4.2.2. Configuring the STF connection for the overcloud ................................. 27
  4.2.3. Validating client-side installation ......................................................... 28

## CHAPTER 5. ADVANCED FEATURES ................................................................. 33

5.1. CUSTOMIZING THE DEPLOYMENT ................................................................. 33
  5.1.1. Manifest override parameters ............................................................... 34
  5.1.2. Overriding a managed manifest ............................................................. 34
5.2. ALERTS .................................................................................................... 36
  5.2.1. Creating an alert rule in Prometheus ..................................................... 36
  5.2.2. Configuring custom alerts ................................................................. 37
5.2.3. Creating an alert route in Alertmanager
5.3. CONFIGURING SNMP TRAPS
5.4. HIGH AVAILABILITY
  5.4.1. Configuring high availability
5.5. DASHBOARDS
  5.5.1. Setting up Grafana to host the dashboard
  5.5.2. Importing dashboards
  5.5.3. Viewing and editing queries
5.6. MULTIPLE CLOUD CONFIGURATION
  5.6.1. Planning AMQP address prefixes
  5.6.2. Deploying Smart Gateways
  5.6.3. Deleting the default Smart Gateways
  5.6.4. Creating the OpenStack environment file
  5.6.5. Querying metrics data from multiple clouds
5.7. EPHEMERAL STORAGE
  5.7.1. Configuring ephemeral storage
5.8. MONITORING THE RESOURCE USAGE OF RED HAT OPENSTACK PLATFORM SERVICES
5.9. MONITORING CONTAINER HEALTH AND API STATUS
5.10. CREATING A ROUTE IN RED HAT OPENSOURCE CONTAINER PLATFORM

CHAPTER 6. UPGRADING SERVICE TELEMETRY FRAMEWORK TO VERSION 1.2

6.1. REMOVING SERVICE TELEMETRY FRAMEWORK 1.1 OPERATORS
6.2. REMOVING THE SERVICE TELEMETRY FRAMEWORK V1.1 CATALOGSOURCE
6.3. SUBSCRIBING TO THE SERVICE TELEMETRY OPERATOR

CHAPTER 7. COLLECTD PLUGINS

collectd::plugin::aggregation
collectd::plugin::ampq
collectd::plugin::amqp1
collectd::plugin::apache
collectd::plugin::battery
collectd::plugin::bind
collectd::plugin::ceph
collectd::plugins::cgroups
collectd::plugin::connectivity
collectd::plugin::conntrack
collectd::plugin::contextswitch
collectd::plugin::cpu
collectd::plugin::cpufreq
collectd::plugin::cpusleep
collectd::plugin::csv
collectd::plugin::curl_json
collectd::plugin::curl
collectd::plugin::curl_xml
collectd::plugin::dbi
collectd::plugin::df
collectd::plugin::disk
collectd::plugin::dns
collectd::plugin::dpdk_telemetry
collectd::plugin::entropy
collectd::plugin::ethstat
collectd::plugin::exec
collectd::plugin::fhcount
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>collectd::plugin::filecount</td>
<td>68</td>
</tr>
<tr>
<td>collectd::plugin::fscache</td>
<td>68</td>
</tr>
<tr>
<td>collectd-hddtemp</td>
<td>68</td>
</tr>
<tr>
<td>collectd::plugin::hugepages</td>
<td>68</td>
</tr>
<tr>
<td>collectd::plugin::intel_rdt</td>
<td>69</td>
</tr>
<tr>
<td>collectd::plugin::interface</td>
<td>69</td>
</tr>
<tr>
<td>collectd::plugin::ipc</td>
<td>70</td>
</tr>
<tr>
<td>collectd::plugin::ipmi</td>
<td>70</td>
</tr>
<tr>
<td>collectd::plugin::iptables</td>
<td>70</td>
</tr>
<tr>
<td>collectd::plugin::irq</td>
<td>70</td>
</tr>
<tr>
<td>collectd::plugin::load</td>
<td>70</td>
</tr>
<tr>
<td>collectd::plugin::logfile</td>
<td>70</td>
</tr>
<tr>
<td>collectd::plugin::log_logstash</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::madwifi</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::match_empty_counter</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::match_hashed</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::match_regex</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::match_timediff</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::match_value</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::mbmon</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::mcelog</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::md</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::memcached</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::memcache</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::memory</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::multimeter</td>
<td>71</td>
</tr>
<tr>
<td>collectd::plugin::mysql</td>
<td>72</td>
</tr>
<tr>
<td>collectd::plugin::netlink</td>
<td>72</td>
</tr>
<tr>
<td>collectd::plugin::network</td>
<td>72</td>
</tr>
<tr>
<td>collectd::plugin::nfs</td>
<td>73</td>
</tr>
<tr>
<td>collectd::plugin::notify_nagios</td>
<td>73</td>
</tr>
<tr>
<td>collectd::plugin::ntpd</td>
<td>73</td>
</tr>
<tr>
<td>collectd::plugin::numa</td>
<td>73</td>
</tr>
<tr>
<td>collectd::plugin::olsrd</td>
<td>73</td>
</tr>
<tr>
<td>collectd::plugin::openldap</td>
<td>73</td>
</tr>
<tr>
<td>collectd::plugin::openvpn</td>
<td>73</td>
</tr>
<tr>
<td>collectd::plugin::ovs_stats</td>
<td>73</td>
</tr>
<tr>
<td>collectd::plugin::pcie_errors</td>
<td>74</td>
</tr>
<tr>
<td>collectd::plugin::ping</td>
<td>74</td>
</tr>
<tr>
<td>collectd::plugin::powerdns</td>
<td>75</td>
</tr>
<tr>
<td>collectd::plugin::processes</td>
<td>75</td>
</tr>
<tr>
<td>collectd::plugin::processes</td>
<td>75</td>
</tr>
<tr>
<td>collectd::plugin::protocols</td>
<td>76</td>
</tr>
<tr>
<td>collectd::plugin::python</td>
<td>76</td>
</tr>
<tr>
<td>collectd::plugin::sensors</td>
<td>76</td>
</tr>
<tr>
<td>collectd::plugin::serial</td>
<td>76</td>
</tr>
<tr>
<td>collectd::plugin::smart</td>
<td>76</td>
</tr>
<tr>
<td>collectd::plugin::snmp</td>
<td>76</td>
</tr>
<tr>
<td>collectd::plugin::snmp_agent</td>
<td>76</td>
</tr>
<tr>
<td>collectd::plugin::statsd</td>
<td>76</td>
</tr>
<tr>
<td>collectd::plugin::swap</td>
<td>77</td>
</tr>
<tr>
<td>collectd::plugin::sysevent</td>
<td>77</td>
</tr>
<tr>
<td>collectd::plugin::syslog</td>
<td>77</td>
</tr>
<tr>
<td>Component</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>collectd::plugin::table</td>
<td>77</td>
</tr>
<tr>
<td>collectd::plugin::tail</td>
<td>77</td>
</tr>
<tr>
<td>collectd::plugin::tail_csv</td>
<td>77</td>
</tr>
<tr>
<td>collectd::plugin::target_notification</td>
<td>77</td>
</tr>
<tr>
<td>collectd::plugin::target_replace</td>
<td>77</td>
</tr>
<tr>
<td>collectd::plugin::target_scale</td>
<td>77</td>
</tr>
<tr>
<td>collectd::plugin::target_set</td>
<td>77</td>
</tr>
<tr>
<td>collectd::plugin::target_v5upgrade</td>
<td>77</td>
</tr>
<tr>
<td>collectd::plugin::tcpconns</td>
<td>77</td>
</tr>
<tr>
<td>collectd::plugin::ted</td>
<td>78</td>
</tr>
<tr>
<td>collectd::plugin::thermal</td>
<td>78</td>
</tr>
<tr>
<td>collectd::plugin::threshold</td>
<td>78</td>
</tr>
<tr>
<td>collectd::plugin::turbostat</td>
<td>78</td>
</tr>
<tr>
<td>collectd::plugin::unixsock</td>
<td>78</td>
</tr>
<tr>
<td>collectd::plugin::uptime</td>
<td>78</td>
</tr>
<tr>
<td>collectd::plugin::users</td>
<td>78</td>
</tr>
<tr>
<td>collectd::plugin::uuid</td>
<td>78</td>
</tr>
<tr>
<td>collectd::plugin::virt</td>
<td>78</td>
</tr>
<tr>
<td>collectd::plugin::vmem</td>
<td>79</td>
</tr>
<tr>
<td>collectd::plugin::vserver</td>
<td>79</td>
</tr>
<tr>
<td>collectd::plugin::wireless</td>
<td>79</td>
</tr>
<tr>
<td>collectd::plugin::write_graphite</td>
<td>79</td>
</tr>
<tr>
<td>collectd::plugin::write_http</td>
<td>80</td>
</tr>
<tr>
<td>collectd::plugin::write_kafka</td>
<td>80</td>
</tr>
<tr>
<td>collectd::plugin::write_log</td>
<td>81</td>
</tr>
<tr>
<td>collectd::plugin::zfs_arc</td>
<td>81</td>
</tr>
</tbody>
</table>
CHAPTER 1. INTRODUCTION TO SERVICE TELEMETRY FRAMEWORK 1.2

IMPORTANT

Service Telemetry Framework (STF) is compatible with Red Hat OpenShift Container Platform versions 4.5 and 4.6.

Service Telemetry Framework (STF) receives monitoring data from Red Hat OpenStack Platform or third-party nodes for storage, viewing on dashboards, and alerting. The monitoring data can be either of two types:

**Metric**
- a numeric measurement of an application or system

**Event**
- irregular and discrete occurrences that happen in a system

The collection components that are required on the clients are lightweight. The multicast message bus that is shared by all clients and the deployment provides fast and reliable data transport. Other modular components for receiving and storing data are deployed in containers on OCP.

STF provides access to monitoring functions such as alert generation, visualization through dashboards, and single source of truth telemetry analysis to support orchestration.

1.1. SUPPORT FOR SERVICE TELEMETRY FRAMEWORK

Red Hat supports the two most recent versions of Service Telemetry Framework (STF). Earlier versions are not supported. For more information, see the Service Telemetry Framework Supported Version Matrix.

1.2. SERVICE TELEMETRY FRAMEWORK ARCHITECTURE

Service Telemetry Framework (STF) uses the components described in the following table:

**Table 1.1. STF components**

<table>
<thead>
<tr>
<th>Client</th>
<th>Component</th>
<th>Server (OCP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>An AMQP 1.x compatible messaging bus to shuttle the metrics to STF for storage in Prometheus</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>Smart Gateway to pick metrics and events from the AMQP 1.x bus and to deliver events to ElasticSearch or to provide metrics to Prometheus</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>Prometheus as time-series data storage</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>ElasticSearch as events data storage</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>collectd to collect infrastructure metrics and events</td>
<td>no</td>
</tr>
</tbody>
</table>
The Service Telemetry Framework data collection components, collectd and Ceilometer, and the transport components, AMQ Interconnect and Smart Gateway, are fully supported. The data storage components, Prometheus and ElasticSearch, including the Operator artifacts, and visualization component Grafana are community-supported, and are not officially supported.

For metrics, on the client side, collectd provides infrastructure metrics (without project data), and Ceilometer provides Red Hat OpenStack Platform platform data based on projects or user workload. Both Ceilometer and collectd deliver data to Prometheus by using the AMQ Interconnect transport, delivering the data through the message bus. On the server side, a Golang application called the Smart Gateway takes the data stream from the bus and exposes it as a local scrape endpoint for Prometheus.
If you plan to collect and store events, collectd or Ceilometer delivers event data to the server side by using the AMQ Interconnect transport, delivering the data through the message bus. Another Smart Gateway writes the data to the ElasticSearch datastore.

Server-side STF monitoring infrastructure consists of the following layers:

- Service Telemetry Framework 1.2 (STF)
- Red Hat OpenShift Container Platform 4.5 (OCP) or 4.6
- Infrastructure platform

Figure 1.2. Server-side STF monitoring infrastructure

**NOTE**

Do not install OCP on the same infrastructure that you want to monitor.

**Additional resources**

- For more information about how to deploy Red Hat OpenShift Container Platform, see the [OCP product documentation](https://access.redhat.com/).
- You can install OCP on cloud platforms or on bare metal. For more information about STF performance and scaling, see [https://access.redhat.com/articles/4907241](https://access.redhat.com/articles/4907241).
1.3. INSTALLATION SIZE OF RED HAT OPENSSHIFT CONTAINER PLATFORM

The size of your Red Hat OpenShift Container Platform (OCP) installation depends on the following factors:

- The number of nodes you want to monitor.
- The number of metrics you want to collect.
- The resolution of metrics.
- The length of time that you want to store the data.

Installation of Service Telemetry Framework (STF) depends on the existing Red Hat OpenShift Container Platform environment. Ensure that you install monitoring for Red Hat OpenStack Platform on a platform separate from your Red Hat OpenStack Platform environment. You can install Red Hat OpenShift Container Platform (OCP) on baremetal or other supported cloud platforms. For more information about installing OCP, see OpenShift Container Platform 4.5 Documentation.

The size of your OCP environment depends on the infrastructure you select. For more information about minimum resources requirements when installing OCP on baremetal, see Minimum resource requirements in the Installing a cluster on bare metal guide. For installation requirements of the various public and private cloud platforms which you can install, see the corresponding installation documentation for your cloud platform of choice.
CHAPTER 2. PREPARING YOUR RED HAT OPENShift CONTAINER PLATFORM ENVIRONMENT FOR SERVICE TELEMETRY FRAMEWORK

As you prepare your OCP environment for STF, you must plan for persistent storage, adequate resources, and event storage:

- Ensure that persistent storage is available in your Red Hat OpenShift Container Platform cluster to permit a production grade deployment. For more information, see Section 2.1, “Persistent volumes”.

- Ensure that enough resources are available to run the Operators and the application containers. For more information, see Section 2.2, “Resource allocation”.

- To install ElasticSearch, you must use a community catalog source. If you do not want to use a community catalog or if you do not want to store events, see Section 3.1, “Deploying STF to the OCP environment”.

- STF uses ElasticSearch to store events, which requires a larger than normal `vm.max_map_count`. The `vm.max_map_count` value is set by default in Red Hat OpenShift Container Platform. For more information about how to edit the value of `vm.max_map_count`, see Section 2.4, “Node tuning operator”.

2.1. PERSISTENT VOLUMES

STF uses persistent storage in OCP to instantiate the volumes dynamically so that Prometheus and ElasticSearch can store metrics and events.

When persistent storage is enabled through the Service Telemetry Operator, the Persistent Volume Claims requested in an STF deployment results in an access mode of RWO (ReadWriteOnce). If your environment contains pre-provisioned persistent volumes, ensure that volumes of RWO are available in the OCP default configured `storageClass`.

Additional resources

- For more information about configuring persistent storage for OCP, see Understanding persistent storage.

- For more information about recommended configurable storage technology in Red Hat OpenShift Container Platform, see Recommended configurable storage technology.

2.1.1. Ephemeral storage

You can use ephemeral storage to run Service Telemetry Framework (STF) without persistently storing data in your Red Hat OpenShift Container Platform (OCP) cluster.
WARNING
If you use ephemeral storage, you might experience data loss if a pod is restarted, updated, or rescheduled onto another node. Use ephemeral storage only for development or testing, and not production environments.

2.2. RESOURCE ALLOCATION

To enable the scheduling of pods within the OCP infrastructure, you need resources for the components that are running. If you do not allocate enough resources, pods remain in a Pending state because they cannot be scheduled.

The amount of resources that you require to run STF depends on your environment and the number of nodes and clouds that you want to monitor.

Additional resources

- For recommendations about sizing for metrics collection, see Service Telemetry Framework Performance and Scaling.
- For information about sizing requirements for ElasticSearch, see https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-managing-compute-resources.html.

2.3. METRICS RETENTION TIME PERIOD

The default retention time for metrics stored in STF is 24 hours, which provides enough data to allow for trends to develop for the purposes of alerting. To adjust STF for additional metrics retention time, set a new value in backends.metrics.prometheus.storage.retention, for example, 7d for seven days. If you use long retention periods, returning data from heavily populated Prometheus systems can result in queries returning slowly.

For long-term storage, use systems designed for long-term data retention, for example, Thanos.

Additional resources

- For recommendations about Prometheus data storage and estimating storage space, see https://prometheus.io/docs/prometheus/latest/storage/#operational-aspects

2.4. NODE TUNING OPERATOR

STF uses ElasticSearch to store events, which requires a larger than normal vm.max_map_count. The vm.max_map_count value is set by default in Red Hat OpenShift Container Platform.

TIP

If your host platform is a typical Red Hat OpenShift Container Platform 4 environment, do not make any adjustments. The default node tuning operator is configured to account for ElasticSearch workloads.
If you want to edit the value of `vm.max_map_count`, you cannot apply node tuning manually using the `sysctl` command because Red Hat OpenShift Container Platform manages nodes directly. To configure values and apply them to the infrastructure, you must use the node tuning operator. For more information, see Using the Node Tuning Operator.

In an OCP deployment, the default node tuning operator specification provides the required profiles for ElasticSearch workloads or pods scheduled on nodes. To view the default cluster node tuning specification, run the following command:

```
$ oc get Tuned/default -o yaml -n openshift-cluster-node-tuning-operator
```

The output of the default specification is documented at Default profiles set on a cluster. You can manage the assignment of profiles in the recommend section where profiles are applied to a node when certain conditions are met. When scheduling ElasticSearch to a node in STF, one of the following profiles is applied:

- `openshift-control-plane-es`
- `openshift-node-es`

When scheduling an ElasticSearch pod, there must be a label present that matches `tuned.openshift.io/elasticsearch`. If the label is present, one of the two profiles is assigned to the pod. No action is required by the administrator if you use the recommended Operator for ElasticSearch. If you use a custom-deployed ElasticSearch with STF, ensure that you add the `tuned.openshift.io/elasticsearch` label to all scheduled pods.

Additional resources

- For more information about virtual memory usage by ElasticSearch, see https://www.elastic.co/guide/en/elasticsearch/reference/current/vm-max-map-count.html
- For more information about how the profiles are applied to nodes, see Custom tuning specification.
CHAPTER 3. INSTALLING THE CORE COMPONENTS OF SERVICE TELEMETRY FRAMEWORK

You can use Operators to load the various application components and objects. Each of the following STF core components are managed by Operators:

- Prometheus and AlertManager
- ElasticSearch
- Smart Gateway
- AMQ Interconnect

Prerequisites

- Red Hat OpenShift Container Platform (OCP) version 4.5 or 4.6 is running.
- You have prepared your Red Hat OpenShift Container Platform (OCP) environment and ensured that there is persistent storage and enough resources to run the STF components on top of the OCP environment.

IMPORTANT

Service Telemetry Framework (STF) is compatible with Red Hat OpenShift Container Platform versions 4.5 and 4.6.

Additional resources

- For more information about Operators, see the Understanding Operators guide.

3.1. DEPLOYING STF TO THE OCP ENVIRONMENT

You can deploy STF to the OCP environment in one of two ways:

- Deploy STF and store events with ElasticSearch. For more information, see Section 3.1.1, “Deploying STF to the OpenShift environment with ElasticSearch”.
- Deploy STF without ElasticSearch and disable events support. For more information, see Section 3.1.2, “Deploying STF to the OpenShift environment without ElasticSearch”.

3.1.1. Deploying STF to the OpenShift environment with ElasticSearch

To deploy STF and store events with ElasticSearch, complete the following tasks:

Procedure

1. Section 3.1.3, “Creating a namespace”.
2. Section 3.1.4, “Creating an OperatorGroup”.
3. Section 3.1.5, “Enabling the OperatorHub.io Community Catalog Source”.
4. Section 3.1.6, “Subscribing to the AMQ Certificate Manager Operator”.
Section 3.1.7, “Subscribing to the Elastic Cloud on Kubernetes Operator”.

Section 3.1.8, “Subscribing to the Service Telemetry Operator”.

Section 3.3, “Creating a ServiceTelemetry object in OCP”.

### 3.1.2. Deploying STF to the Openshift environment without ElasticSearch

To deploy STF without ElasticSearch and disable events support, complete the following tasks:

**Procedure**

1. Section 3.1.3, “Creating a namespace”.

2. Section 3.1.4, “Creating an OperatorGroup”.

3. Section 3.1.6, “Subscribing to the AMQ Certificate Manager Operator”.

4. Section 3.1.8, “Subscribing to the Service Telemetry Operator”.

5. Section 3.3, “Creating a ServiceTelemetry object in OCP”.

### 3.1.3. Creating a namespace

Create a namespace to hold the STF components. The service-telemetry namespace is used throughout the documentation:

**Procedure**

- Enter the following command:

  ```bash
  $ oc new-project service-telemetry
  ```

### 3.1.4. Creating an OperatorGroup

Create an OperatorGroup in the namespace so that you can schedule the Operator pods.

**Procedure**

- Enter the following command:

  ```bash
  $ oc apply -f - <<EOF
  apiVersion: operators.coreos.com/v1
  kind: OperatorGroup
  metadata:
    name: service-telemetry-operator-group
  namespace: service-telemetry
  spec:
    targetNamespaces:
      - service-telemetry
  EOF
  ```

**Additional resources**
For more information, see OperatorGroups.

### 3.1.5. Enabling the OperatorHub.io Community Catalog Source

Before you install ElasticSearch, you must have access to the resources on the OperatorHub.io Community Catalog Source:

**Procedure**

- Enter the following command:

  ```
  $ oc apply -f - <<EOF
  apiVersion: operators.coreos.com/v1alpha1
  kind: CatalogSource
  metadata:
    name: operatorhubio-operators
    namespace: openshift-marketplace
  spec:
    sourceType: grpc
    image: quay.io/operatorhubio/catalog:latest
    displayName: OperatorHub.io Operators
    publisher: OperatorHub.io
  EOF
  ```

### 3.1.6. Subscribing to the AMQ Certificate Manager Operator

You must subscribe to the AMQ Certificate Manager Operator before you deploy the other STF components because the AMQ Certificate Manager Operator runs globally-scoped. The AMQ Certificate Manager Operator is not compatible with the dependency management of Operator Lifecycle Manager when you use it with other namespace-scoped operators.

**NOTE**

If you are using Red Hat OpenShift Container Platform (OCP) 4.5, go directly to step 3.

**Procedure**

1. If you are using OCP 4.6, enable the Red Hat STF Operators CatalogSource:

  ```
  $ oc apply -f - <<EOF
  apiVersion: operators.coreos.com/v1alpha1
  kind: CatalogSource
  metadata:
    name: redhat-operators-stf
    namespace: openshift-marketplace
  spec:
    displayName: Red Hat STF Operators
    image: quay.io/redhat-operators-stf/stf-catalog:v4.6
    publisher: Red Hat
    sourceType: grpc
    updateStrategy:
      registryPoll:
        interval: 30m
  EOF
  ```
2. If you are using OCP 4.6, subscribe to the AMQ Certificate Manager Operator via the `redhat-operators-stf` CatalogSource:

```bash
$ oc apply -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq7-cert-manager-operator
  namespace: openshift-operators
spec:
  channel: alpha
  installPlanApproval: Automatic
  name: amq7-cert-manager-operator
  source: redhat-operators-stf
  targetNamespaces: global
EOF
```

3. If you are using OCP 4.5, subscribe to the AMQ Certificate Manager Operator, create the subscription, and validate the AMQ Certificate Manager:

**NOTE**

The AMQ Certificate Manager is installed globally for all namespaces, so the `namespace` value provided is `openshift-operators`. You might not see your `amq7-cert-manager.v1.0.0` ClusterServiceVersion in the `service-telemetry` namespace for a few minutes until the processing executes against the namespace.

```bash
$ oc apply -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq7-cert-manager
  namespace: openshift-operators
spec:
  channel: alpha
  installPlanApproval: Automatic
  name: amq7-cert-manager
  source: redhat-operators
  targetNamespaces: openshift-marketplace
EOF
```

4. For OCP versions 4.5 and OCP 4.6, validate your `ClusterServiceVersion`:

```bash
$ oc get --namespace openshift-operators csv
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>DISPLAY</th>
<th>VERSION</th>
<th>REPLACES</th>
<th>PHASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>amq7-cert-manager.v1.0.0</td>
<td>Red Hat Integration - AMQ Certificate Manager</td>
<td>1.0.0</td>
<td>AMQ Certificate Manager</td>
<td>Succeeded</td>
</tr>
</tbody>
</table>

Ensure that `amq7-cert-manager.v1.0.0` has a phase **Succeeded**.
3.1.7. Subscribing to the Elastic Cloud on Kubernetes Operator

Before you install the Service Telemetry Operator and if you plan to store events in Elasticsearch, you must enable the Elastic Cloud Kubernetes Operator.

Procedure

1. To enable the Elastic Cloud on Kubernetes Operator, apply the following manifest to your OCP environment:

   ```
   $ oc apply -f - <<EOF
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: elastic-cloud-eck
     namespace: service-telemetry
   spec:
     channel: stable
     installPlanApproval: Automatic
     name: elastic-cloud-eck
     source: operatorhubio-operators
     sourceNamespace: openshift-marketplace
   EOF
   ```

2. Verify that the `ClusterServiceVersion` for Elasticsearch Cloud on Kubernetes **succeeded**:

   ```
   $ oc get csv
   NAME                       DISPLAY                                         VERSION   REPLACES   PHASE
   elastic-cloud-eck.v1.2.1   Elastic Cloud on Kubernetes                     1.2.1                Succeeded
   ```

3.1.8. Subscribing to the Service Telemetry Operator

You must subscribe to the Service Telemetry Operator, which manages the STF instances.

Procedure

1. To create the Service Telemetry Operator subscription, enter the `oc apply -f` command:

   ```
   $ oc apply -f - <<EOF
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: service-telemetry-operator
     namespace: service-telemetry
   spec:
     channel: stable-1.2
     installPlanApproval: Automatic
     name: service-telemetry-operator
     source: redhat-operators
     sourceNamespace: openshift-marketplace
   EOF
   ```
2. To validate the Service Telemetry Operator and the dependent operators, enter the following command:

```
$ oc get csv --namespace service-telemetry
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>DISPLAY</th>
<th>VERSION</th>
<th>REPLACES</th>
</tr>
</thead>
<tbody>
<tr>
<td>amq7-cert-manager.v1.0.0</td>
<td>Red Hat Integration - AMQ Certificate Manager</td>
<td>1.0.0</td>
<td></td>
</tr>
<tr>
<td>succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>amq7-interconnect-operator.v1.2.3</td>
<td>Red Hat Integration - AMQ Interconnect</td>
<td>1.2.3</td>
<td></td>
</tr>
<tr>
<td>succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>elastic-cloud-eck.v1.4.0</td>
<td>Elasticsearch (ECK) Operator</td>
<td>1.4.0</td>
<td></td>
</tr>
<tr>
<td>succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grafana-operator.v3.9.0</td>
<td>Grafana Operator</td>
<td>3.9.0</td>
<td></td>
</tr>
<tr>
<td>succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prometheusoperator.0.37.0</td>
<td>Prometheus Operator</td>
<td>0.37.0</td>
<td></td>
</tr>
<tr>
<td>succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>service-telemetry-operator.v1.2.1</td>
<td>Service Telemetry Operator</td>
<td>1.2.1</td>
<td></td>
</tr>
<tr>
<td>succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smart-gateway-operator.v2.2.1</td>
<td>Smart Gateway Operator</td>
<td>2.2.1</td>
<td></td>
</tr>
<tr>
<td>succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.2. OVERVIEW OF THE SERVICETELEMETRY OBJECT

To deploy the Service Telemetry Framework, you must create an instance of `ServiceTelemetry` in OCP. The `ServiceTelemetry` object is made up of the following major configuration parameters:

- alerting
- backends
- clouds
- graphing
- highAvailability
- transports

Each of these top-level configuration parameters provides various controls for a Service Telemetry Framework deployment.

**IMPORTANT**

Versions of Service Telemetry Operator prior to v1.1.0 used a flat API ([servicetelemetry.infra.watch/v1alpha1](http://servicetelemetry.infra.watch/v1alpha1)) interface for creating the `ServiceTelemetry` object. In Service Telemetry Operator v1.1.0, there is a dictionary-based API interface ([servicetelemetry.infra.watch/v1beta1](http://servicetelemetry.infra.watch/v1beta1)) to allow for better control of STF deployments, including managing multi-cluster deployments natively, and allowing the management of additional storage backends in the future. Ensure that any previously created `ServiceTelemetry` objects are updated to the new interface.

Support for [servicetelemetry.infra.watch/v1alpha1](http://servicetelemetry.infra.watch/v1alpha1) will be removed in STF 1.3.

### 3.2.1. backends
Use the `backends` parameter to control which storage backends are available for storage of metrics and events, and to control the enablement of Smart Gateways, as defined by the `clouds` parameter. For more information, see Section 3.2.2, “clouds”.

Currently, you can use Prometheus as the metrics backend, and ElasticSearch as the events backend.

### 3.2.1.1. Enabling Prometheus as a storage backend for metrics

**Procedure**

- To enable Prometheus as a storage backend for metrics, configure the `ServiceTelemetry` object:

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
spec:
  backends:
    metrics:
      prometheus:
        enabled: true
```

### 3.2.1.2. Enabling ElasticSearch as a storage backend for events

To enable events support in STF, you must enable the Elastic Cloud for Kubernetes Operator. For more information, see Section 3.1.7, “Subscribing to the Elastic Cloud on Kubernetes Operator”.

By default, ElasticSearch storage of events is disabled. For more information, see Section 3.1.2, “Deploying STF to the OpenShift environment without ElasticSearch”.

### 3.2.2. clouds

Use the `clouds` parameter to control which Smart Gateway objects are deployed, thereby providing the interface for multiple monitored cloud environments to connect to an instance of STF. If a supporting backend is available, then metrics and events Smart Gateways for the default cloud configuration are created. By default, the Service Telemetry Operator creates Smart Gateways for `cloud1`.

You can create a list of cloud objects to control which Smart Gateways are created for each cloud defined. Each cloud is made up of data types and collectors. Data types are `metrics` or `events`. Each data type is made up of a list of collectors and the message bus subscription address. Available collectors are `collectd` and `ceilometer`. Ensure that the subscription address for each of these collectors is unique for every cloud, data type, and collector combination.

The default `cloud1` configuration is represented by the following `ServiceTelemetry` object, providing subscriptions and data storage of metrics and events for both collectd and Ceilometer data collectors for a particular cloud instance:

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: stf-default
  namespace: service-telemetry
```
spec:
cLOUDS:
  - name: cloud1
metrics:
  collectors:
    - collectorType: collectd
      subscriptionAddress: collectd/telemetry
    - collectorType: ceilometer
      subscriptionAddress: anycast/ceilometer/metering.sample
events:
  collectors:
    - collectorType: collectd
      subscriptionAddress: collectd/notify
    - collectorType: ceilometer
      subscriptionAddress: anycast/ceilometer/event.sample

Each item of the Clouds parameter represents a cloud instance. The cloud instances are made up of 3 top-level parameters: name, metrics, and events. The metrics and events parameters represent the corresponding backend for storage of that data type. The collectors parameter then specifies a list of objects made up of two parameters, collectorType and subscriptionAddress, and these represent an instance of the Smart Gateway. The collectorType specifies data collected by either collectd or Ceilometer. The subscriptionAddress parameter provides the AMQ Interconnect address that a Smart Gateway instance should subscribe to.

3.2.3. alerting

Use the alerting parameter to control creation of an Alertmanager instance and the configuration of the storage backend. By default, alerting is enabled. For more information, see Section 5.2, “Alerts”.

3.2.4. graphing

Use the graphing parameter to control the creation of a Grafana instance. By default, graphing is disabled. For more information, see Section 5.5, “Dashboards”.

3.2.5. highAvailability

Use The highAvailability parameter to control the instantiation of multiple copies of STF components to reduce recovery time of components that fail or are rescheduled. By default, highAvailability is disabled. For more information, see Section 5.4, “High availability”.

3.2.6. transports

Use the transports parameter to control the enablement of the message bus for a STF deployment. The only transport currently supported is AMQ Interconnect. Ensure that it is enabled for proper operation of STF. By default, the qdr transport is enabled.

3.3. CREATING A SERVICETELEMETRY OBJECT IN OCP

Create a ServiceTelemetry object in OCP to result in the creation of supporting components for a Service Telemetry Framework deployment. For more information, see Section 3.2, “Overview of the ServiceTelemetry object”.

Procedure
1. To create a `ServiceTelemetry` object that results in a default STF deployment, create a `ServiceTelemetry` object with an empty `spec` object:

   ```
   $ oc apply -f <<-EOF
   apiVersion: infra.watch/v1beta1
   kind: ServiceTelemetry
   metadata:
       name: default
       namespace: service-telemetry
   spec: {}
   EOF
   ```

Creating a default `ServiceTelemetry` object results in a STF deployment with the following defaults:

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
    name: default
spec:
    alerting:
        enabled: true
    alertmanager:
        storage:
            strategy: persistent
            persistent:
                storageSelector: {}
            pvcStorageRequest: 20G
    backends:
        metrics:
            prometheus:
                enabled: true
            scrapeInterval: 10s
            storage:
                strategy: persistent
                retention: 24h
                persistent:
                    storageSelector: {}
                pvcStorageRequest: 20G
    events:
        elasticsearch:
            enabled: false
            storage:
                strategy: persistent
                persistent:
                    pvcStorageRequest: 20Gi
    graphing:
        enabled: false
    grafana:
        ingressEnabled: false
        adminPassword: secret
        adminUser: root
        disableSignoutMenu: false
    transports:
        qdr:
            enabled: true
```
2. Optional: To create a **ServiceTelemetry** object that results in collection and storage of events for the default cloud, enable the ElasticSearch backend:

```
$ oc apply -f - <<EOF
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
spec:
  backends:
    events:
      elasticsearch:
        enabled: true
EOF
```

3. To view the STF deployment logs in the Service Telemetry Operator, use the **oc logs** command:

```
$ oc logs --selector name=service-telemetry-operator -c ansible
PLAY RECAP ***
localhost : ok=55  changed=0   unreachable=0   failed=0   skipped=16
rescued=0   ignored=0
```

4. View the pods and the status of each pod to determine that all workloads are operating nominally:

**NOTE**

If you set `backends.events.elasticsearch.enabled: true`, the notification Smart Gateways reports **Error** and **CrashLoopBackOff** error messages for a period of time before ElasticSearch starts.
3.4. REMOVING STF FROM THE OCP ENVIRONMENT

Remove STF from an OCP environment if you no longer require the STF functionality.

Complete the following tasks:

1. Section 3.4.1, “Deleting the namespace”.

2. Section 3.4.2, “Removing the CatalogSource”.

3.4.1. Deleting the namespace

To remove the operational resources for STF from OCP, delete the namespace.

Procedure

1. Run the `oc delete` command:

   $ oc delete project service-telemetry

2. Verify that the resources have been deleted from the namespace:

   $ oc get all
   No resources found.

3.4.2. Removing the CatalogSource

If you do not expect to install Service Telemetry Framework again, delete the CatalogSource. When you remove the CatalogSource, PackageManifests related to STF are removed from the Operator Lifecycle Manager catalog.

Procedure

1. If you enabled the OperatorHub.io Community Catalog Source during the installation process and you no longer need this catalog source, delete it:
$ oc delete --namespace=openshift-marketplace catalogsource operatorhubio-operators
catalogsource.operators.coreos.com "operatorhubio-operators" deleted

Additional resources

For more information about the OperatorHub.io Community Catalog Source, see Section 3.1, “Deploying STF to the OCP environment”.
CHAPTER 4. COMPLETING THE SERVICE TELEMETRY FRAMEWORK CONFIGURATION

To collect metrics, events, or both, and to send them to the Service Telemetry Framework (STF) storage domain, you must configure the Red Hat OpenStack Platform overcloud to enable data collection and transport.

To deploy data collection and transport to STF on Red Hat OpenStack Platform cloud nodes that employ routed L3 domains, such as distributed compute node (DCN) or spine-leaf, see Section 4.1, “Deploying to non-standard network topologies”.

4.1. DEPLOYING TO NON-STANDARD NETWORK TOPOLOGIES

If your nodes are on a separate network from the default InternalApi network, you must make configuration adjustments so that AMQ Interconnect can transport data to the Service Telemetry Framework (STF) server instance. This scenario is typical in a spine-leaf or a DCN topology. For more information about DCN configuration, see the Spine Leaf Networking guide.

If you use STF with Red Hat OpenStack Platform 16.1 and plan to monitor your Ceph, Block, or Object storage nodes, you must make configuration changes that are similar to the configuration changes that you make to the spine-leaf and DCN network configuration. To monitor Ceph nodes, use the CephStorageExtraConfig parameter to define which network interface to load into the AMQ Interconnect and collectd configuration files.

Similarly, you must specify BlockStorageExtraConfig and ObjectStorageExtraConfig parameters if your environment uses Block and Object storage roles.

The deployment of a spine-leaf topology involves creating roles and networks, then assigning those networks to the available roles. When you configure data collection and transport for STF for an Red Hat OpenStack Platform deployment, the default network for roles is InternalApi. For Ceph, Block and Object storage roles, the default network is Storage. Because a spine-leaf configuration can result in different networks being assigned to different Leaf groupings and those names are typically unique, additional configuration is required in the parameter_defaults section of the Red Hat OpenStack Platform environment files.

Procedure

1. Document which networks are available for each of the Leaf roles. For examples of network name definitions, see Creating a network data file in the Spine Leaf Networking guide. For more information about the creation of the Leaf groupings (roles) and assignment of the networks to those groupings, see Creating a roles data file in the Spine Leaf Networking guide.

2. Add the following configuration example to the ExtraConfig section for each of the leaf roles. In this example, internal_api_subnet is the value defined in the name_lower parameter of your network definition (with _subnet appended to the name for Leaf 0), and
is the network to which the ComputeLeaf0 leaf role is connected. In this case, the network identification of 0 corresponds to the Compute role for leaf 0, and represents a value that is different from the default internal API network name.

For the ComputeLeaf0 leaf role, specify extra configuration to perform a hiera lookup to determine which network interface for a particular network to assign to the collectd AMQP host parameter. Perform the same configuration for the AMQ Interconnect listener address parameter.

```
ComputeLeaf0ExtraConfig:
   ☛ tripleo::profile::base::metrics::collectd::amqp_host: "%(hiera('internal_api_subnet'))"
   ☛ tripleo::profile::base::metrics::qdr::listener_addr: "%(hiera('internal_api_subnet'))"
```

Additional leaf roles typically replace _subnet with _leafN where N represents a unique identifier for the leaf.

```
ComputeLeaf1ExtraConfig:
   ☛ tripleo::profile::base::metrics::collectd::amqp_host: "%(hiera('internal_api_leaf1'))"
   ☛ tripleo::profile::base::metrics::qdr::listener_addr: "%(hiera('internal_api_leaf1'))"
```

This example configuration is on a CephStorage leaf role:

```
CephStorageLeaf0ExtraConfig:
   ☛ tripleo::profile::base::metrics::collectd::amqp_host: "%(hiera('storage_subnet'))"
   ☛ tripleo::profile::base::metrics::qdr::listener_addr: "%(hiera('storage_subnet'))"
```

## 4.2. CONFIGURING RED HAT OPENSTACK PLATFORM OVERCLOUD FOR SERVICE TELEMETRY FRAMEWORK

To configure the Red Hat OpenStack Platform overcloud, you must configure the data collection applications and the data transport to STF, and deploy the overcloud.

To configure the Red Hat OpenStack Platform overcloud, complete the following tasks:

1. Section 4.2.1, “Retrieving the AMQ Interconnect route address”
2. Section 4.2.2, “Configuring the STF connection for the overcloud”
3. Section 4.2.3, “Validating client-side installation”

Additional resources

- To collect data through AMQ Interconnect, see The amqp1 plug-in in the Monitoring Tools Configuration guide.

### 4.2.1. Retrieving the AMQ Interconnect route address

When you configure the Red Hat OpenStack Platform overcloud for STF, you must provide the AMQ Interconnect route address in the STF connection file.

**Procedure**

1. Log in to your Red Hat OpenShift Container Platform (OCP) environment.
2. In the service-telemetry project, retrieve the AMQ Interconnect route address:

```bash
$ oc get routes -o-template='{{ range .items }}{{printf "%s\n" .spec.host}}{{ end }}' | grep ^\-5671
default-interconnect-5671-service-telemetry.apps.infra.watch
```

**NOTE**

If your STF installation differs from the documentation, ensure that you retrieve the correct AMQ Interconnect route address.

### 4.2.2. Configuring the STF connection for the overcloud

To configure the STF connection, you must create a file that contains the connection configuration of the AMQ Interconnect for the overcloud to the STF deployment. Enable the collection of events and storage of the events in STF and deploy the overcloud.

#### Procedure

1. Log in to the Red Hat OpenStack Platform undercloud as the *stack* user.

2. Create a configuration file called `stf-connectors.yaml` in the `/home/stack` directory.

**IMPORTANT**

The Service Telemetry Operator simplifies the deployment of all data ingestion and data storage components for single cloud deployments. To share the data storage domain with multiple clouds, see Section 5.6, “Multiple cloud configuration”.

Additionally, setting `EventPipelinePublishers` and `PipelinePublishers` to empty lists results in no metric or event data passing to Red Hat OpenStack Platform legacy telemetry components, such as Gnocchi or Panko. If you need to send data to additional pipelines, the Ceilometer polling interval of 5 seconds as specified in `ExtraConfig` might overwhelm the legacy components. If you configure a longer polling interval, you must also modify STF to avoid stale metrics, resulting in what appears to be missing data in Prometheus.

If an adjustment needs to be made to the polling interval, then modify the `ServiceTelemetry` object `backends.metrics.prometheus.scrapeInterval` parameter from the default value of `10s` to double the polling interval of the data collectors. For example, if `CollectdAmqpInterval` and `ceilometer::agent::polling::polling_interval` are adjusted to `30` then set the `backends.metrics.prometheus.scrapeInterval` to a value of `60s`.

3. In the `stf-connectors.yaml` file, configure the `MetricsQdrConnectors` address to connect the AMQ Interconnect on the overcloud to the STF deployment.

- Add the `CeilometerQdrPublishMetrics: true` parameter to enable collection and transport of Ceilometer metrics to STF.

- Add the `CeilometerQdrPublishEvents: true` parameter to enable collection and transport of Ceilometer events to STF.
- Add the `EventPipelinePublishers: []` and `PipelinePublishers: []` to avoid writing data to Gnocchi and Panko.

- Add the `ManagePolling: true` and `ManagePipeline: true` parameters to allow full control of Ceilometer polling and pipeline configuration.

- Add the `ExtraConfig` parameter `ceilometer::agent::polling::polling_interval` to set the polling interval of Ceilometer to be compatible with the default STF scrape interval.

- Replace the `host` parameter with the value of HOST/PORT that you retrieved in Section 4.2.1, “Retrieving the AMQ Interconnect route address”:

  ```yaml
  parameter_defaults:
  EventPipelinePublishers: []
  PipelinePublishers: []
  CeilometerQdrPublishEvents: true
  CeilometerQdrPublishMetrics: true
  MetricsQdrConnectors:
  - host: default-interconnect-5671-service-telemetry.apps.infra.watch
    port: 443
    role: edge
    sslProfile: sslProfile
    verifyHostname: false
  ExtraConfig:
    ceilometer::agent::polling::polling_interval: 5
  ```

4. Add the following files to your Red Hat OpenStack Platform director deployment to setup collectd and AMQ Interconnect:

- the `stf-connectors.yaml` environment file
- the `enable-stf.yaml` file that ensures that the environment is being used during the overcloud deployment
- the `ceilometer-write-qdr.yaml` file that ensures that Ceilometer telemetry is sent to STF

```
openstack overcloud deploy <other arguments>
  --templates /usr/share/openstack-tripleo-heat-templates \
  --environment-file <...other-environment-files...> \
  --environment-file /usr/share/openstack-tripleo-heat-templates/environments/metrics/ceilometer-write-qdr.yaml \
  --environment-file /usr/share/openstack-tripleo-heat-templates/environments/enable-stf.yaml \
  --environment-file /home/stack/stf-connectors.yaml
```

5. Deploy the Red Hat OpenStack Platform overcloud.

4.2.3. Validating client-side installation

To validate data collection from the STF storage domain, query the data sources for delivered data. To validate individual nodes in the Red Hat OpenStack Platform deployment, connect to the console using SSH.
TIP

Some telemetry data is only available when Red Hat OpenStack Platform has active workloads.

Procedure

1. Log in to an overcloud node, for example, controller-0.

2. Ensure that metrics_qdr container is running on the node:

   ```
   $ sudo podman container inspect --format '{{.State.Status}}' metrics_qdr
   running
   ```

3. Return the internal network address on which AMQ Interconnect is running, for example, 172.17.1.44 listening on port 5666:

   ```
   $ sudo podman exec -it metrics_qdr cat /etc/qpid-dispatch/qdrouterd.conf
   
   listener {
       host: 172.17.1.44
       port: 5666
       authenticatePeer: no
       saslMechanisms: ANONYMOUS
   }
   ```

4. Return a list of connections to the local AMQ Interconnect:

   ```
   $ sudo podman exec -it metrics_qdr qdstat --bus=172.17.1.44:5666 --connections
   
   Connections
<table>
<thead>
<tr>
<th>id</th>
<th>host</th>
<th>container</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>dir</td>
<td>security</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>1</td>
<td>default-interconnect-5671-service-telemetry.apps.infra.watch:443</td>
<td>default-interconnect-7458fd4d69-bgzfbc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLSv1.2(DHE-RSA-AES256-GCM-SHA384) anonymous-user</td>
</tr>
<tr>
<td>12</td>
<td>172.17.1.44:60290 openstack.org/om/container/controller-0/ceilometer-agent-notification/25/5c02cee550f143ec9ea030db5cccba14</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no-auth</td>
</tr>
<tr>
<td>16</td>
<td>172.17.1.44:36408 metrics</td>
<td>normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no-auth</td>
</tr>
<tr>
<td>899</td>
<td>172.17.1.44:39500 10a2e99d-1b8a-4329-b48c-4335e5f75c84</td>
<td>anonymous-user normal</td>
</tr>
</tbody>
</table>
   ```

There are four connections:

- Outbound connection to STF
• Inbound connection from ceilometer

• Inbound connection from collectd

• Inbound connection from our qdstat client
  The outbound STF connection is provided to the MetricsQdrConnectors host parameter and is the route for the STF storage domain. The other hosts are internal network addresses of the client connections to this AMQ Interconnect.

5. To ensure that messages are being delivered, list the links, and view the _edge address in the deliv column for delivery of messages:

$ sudo podman exec -it metrics_qdr qdstat --bus=172.17.1.44:5666 --links
Router Links
  type    dir  conn id  id    peer  class   addr                  phs cap pri undel  unsettling deliv
  preset  psdeliver  acc  rej  rel     mod  delay  rate
===========================================================================
-------------------------------------------------------------------------------
=====
endpoint out 1  5 local _edge                      250  0  0  0       2979926 0
0 0 0 2979926 0 0 0
endpoint in 1  6
250  0  0  0  0  0  0
0 0 0 0 0 0 0
endpoint in 1  7
250  0  0  0  0  0  0
0 0 0 0 0 0 0
endpoint out 1  8
250  0  0  0  0  0  0
0 0 0 0 0 0 0
endpoint in 1  9
250  0  0  0  0  0  0
0 0 0 0 0 0 0
endpoint out 1 10
250  0  0  0  911  911  0
0 0 0 911 0 0 0
endpoint in 1 11
250  0  0  0  911  0  0
0 0 0 0 0 0 0
endpoint out 12 32 local temp.ISY6Mcicol4J2Kp 250  0  0  0
0 0 0 0 0 0 0
endpoint in 16 41
250  0  0  0  2979924 0  0
0 0 2979924 0 0 0 0
endpoint in 912 1834 mobile $management
0 250  0  0  1
0 0 1 0 0 0 0
endpoint out 912 1835 local temp.9Ok2resl9tmt+CT 250  0  0  0
0 0 0 0 0 0 0
6. To list the addresses from Red Hat OpenStack Platform nodes to STF, connect to OCP to get the AMQ Interconnect pod name and list the connections. List the available AMQ Interconnect pods:

$ oc get pods -l application=default-interconnect

NAME                                    READY   STATUS    RESTARTS   AGE
default-interconnect-7458fd4d69-bgzfb   1/1     Running   0          6d21h

7. Connect to the pod and run the qdstat --connections command to list the known connections:
In this example, there are three edge connections from the Red Hat OpenStack Platform nodes with connection id 22, 23, and 24.

8. To view the number of messages delivered by the network, use each address with the oc exec command:

```bash
$ oc exec -it default-interconnect-7458fd4d69-bgzfb -- qdstat --connections
```

```
2020-04-21 18:25:47.243852 UTC
default-interconnect-7458fd4d69-bgzfb

Connections
id  host               container                                                      role    dir  security  authentication  tenant  last dlv    uptime

5   10.129.0.110:48498 bridge-3f5 edge in no-security anonymous-user 000:00:00:02 000:17:36:29
6   10.129.0.111:43254 rcv[default-cloud1-ceil-meter-smartgateway-58f885c76d-xmxwn] edge in no-security anonymous-user 000:00:00:02 000:17:36:20
7   10.130.0.109:50518 rcv[default-cloud1-coll-event-smartgateway-58fbbd4485-rl9bd] normal in no-security anonymous-user - 000:17:36:11
8   10.130.0.110:33802 rcv[default-cloud1-ceil-event-smartgateway-6cfb65478c-g5q82] normal in no-security anonymous-user 000:01:26:18 000:17:36:05
22  10.128.0.1:51948 Router.ceph-0.redhat.local edge in TLSv1/SSLv3(DHE-RSA-AES256-GCM-SHA384) anonymous-user 000:00:00:03 000:22:08:43
23  10.128.0.1:51950 Router.compute-0.redhat.local edge in TLSv1/SSLv3(DHE-RSA-AES256-GCM-SHA384) anonymous-user 000:00:00:03 000:22:08:43
24  10.128.0.1:52082 Router.controller-0.redhat.local edge in TLSv1/SSLv3(DHE-RSA-AES256-GCM-SHA384) anonymous-user 000:00:00:00 000:22:08:34
27  127.0.0.1:42202 c2f541c1-4c97-4b37-a189-a396c08fb079 normal in no-security no-auth 000:00:00:00 000:00:00:00
```

In this example, there are three edge connections from the Red Hat OpenStack Platform nodes with connection id 22, 23, and 24.

8. To view the number of messages delivered by the network, use each address with the oc exec command:
<table>
<thead>
<tr>
<th>Service</th>
<th>Count</th>
<th>Type</th>
<th>Length</th>
<th>Status</th>
<th>Hex</th>
<th>Total</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>mobile collectd/notify</td>
<td>0</td>
<td>multicast</td>
<td>1</td>
<td>0</td>
<td>70</td>
<td>70</td>
<td>0</td>
</tr>
<tr>
<td>mobile collectd/telemetry</td>
<td>0</td>
<td>multicast</td>
<td>1</td>
<td>0</td>
<td>216,128,890</td>
<td>216,128,890</td>
<td>0</td>
</tr>
</tbody>
</table>
CHAPTER 5. ADVANCED FEATURES

The following optional features can provide additional functionality to the Service Telemetry Framework (STF):

- Section 5.1, “Customizing the deployment”
- Section 5.2, “Alerts”
- Section 5.3, “Configuring SNMP Traps”
- Section 5.4, “High availability”
- Section 5.5, “Dashboards”
- Section 5.6, “Multiple cloud configuration”
- Section 5.7, “Ephemeral storage”
- Section 5.8, “Monitoring the resource usage of Red Hat OpenStack Platform services”
- Section 5.10, “Creating a route in Red Hat OpenShift Container Platform”

5.1. CUSTOMIZING THE DEPLOYMENT

The Service Telemetry Operator watches for a ServiceTelemetry manifest to load into Red Hat OpenShift Container Platform (OCP). The Operator then creates other objects in memory, which results in the dependent Operators creating the workloads they are responsible for managing.

WARNING

When you override the manifest, you must provide the entire manifest contents, including object names or namespaces. There is no dynamic parameter substitution when you override a manifest.

Use manifest overrides only as a last resort short circuit.

To override a manifest successfully with Service Telemetry Framework (STF), deploy a default environment using the core options only. For more information about the core options, see Section 3.3, “Creating a ServiceTelemetry object in OCP”. When you deploy STF, use the `oc get` command to retrieve the default deployed manifest. When you use a manifest that was originally generated by Service Telemetry Operator, the manifest is compatible with the other objects that are managed by the Operators.

For example, when the `backends.metrics.prometheus.enabled: true` parameter is configured in the ServiceTelemetry manifest, the Service Telemetry Operator requests components for metrics retrieval and storage using the default manifests. In some cases, you might want to override the default manifest. For more information, see Section 5.1.1, “Manifest override parameters”.
5.1.1. Manifest override parameters

This table describes the available parameters that you can use to override a manifest, along with the corresponding retrieval commands.

<table>
<thead>
<tr>
<th>Override parameter</th>
<th>Description</th>
<th>Retrieval command</th>
</tr>
</thead>
<tbody>
<tr>
<td>alertmanagerManifest</td>
<td>Override the <code>Alertmanager</code> object creation. The Prometheus Operator watches for <code>Alertmanager</code> objects.</td>
<td><code>oc get alertmanager default -oyaml</code></td>
</tr>
<tr>
<td>alertmanagerConfigManifest</td>
<td>Override the <code>Secret</code> that contains the Alertmanager configuration. The Prometheus Operator uses a secret named <code>alertmanager-{{ alertmanager-name }}</code>, for example, <code>default</code>, to provide the <code>alertmanager.yaml</code> configuration to Alertmanager.</td>
<td><code>oc get secret alertmanager-default -oyaml</code></td>
</tr>
<tr>
<td>elasticsearchManifest</td>
<td>Override the <code>ElasticSearch</code> object creation. The Elastic Cloud on Kubernetes Operator watches for <code>ElasticSearch</code> objects.</td>
<td><code>oc get elasticsearch elasticsearch -oyaml</code></td>
</tr>
<tr>
<td>interconnectManifest</td>
<td>Override the <code>Interconnect</code> object creation. The AMQ Interconnect Operator watches for <code>Interconnect</code> objects.</td>
<td><code>oc get interconnect default-interconnect -oyaml</code></td>
</tr>
<tr>
<td>prometheusManifest</td>
<td>Override the <code>Prometheus</code> object creation. The Prometheus Operator watches for <code>Prometheus</code> objects.</td>
<td><code>oc get prometheus default -oyaml</code></td>
</tr>
<tr>
<td>servicemonitorManifest</td>
<td>Override the <code>ServiceMonitor</code> object creation. The Prometheus Operator watches for <code>ServiceMonitor</code> objects.</td>
<td><code>oc get servicemonitor default -oyaml</code></td>
</tr>
</tbody>
</table>

5.1.2. Overriding a managed manifest

Edit the `ServiceTelemetry` object and provide a parameter and manifest. For a list of available manifest override parameters, see Section 5.1, “Customizing the deployment”. The default `ServiceTelemetry` object is `default`. Use `oc get servicetelemetry` to list the available STF deployments.
TIP

The `oc edit` command loads the default system editor. To override the default editor, pass or set the environment variable `EDITOR` to the preferred editor. For example, `EDITOR=nano oc edit servicetelemetry default`.

**Procedure**

1. Log in to Red Hat OpenShift Container Platform.

2. Change to the `service-telemetry` namespace:

   ```bash
   $ oc project service-telemetry
   ```

3. Load the `ServiceTelemetry` object into an editor:

   ```bash
   $ oc edit servicetelemetry default
   ```

4. To modify the `ServiceTelemetry` object, provide a manifest override parameter and the contents of the manifest to write to OCP instead of the defaults provided by STF.

   ```bash
   $ oc edit stf default
   ```

   ```yaml
   apiVersion: infra.watch/v1beta1
   kind: ServiceTelemetry
   metadata:
     ...  
   spec:
     alertmanagerConfigManifest: |
     
     apiVersion: v1
     kind: Secret
     metadata:
       name: 'alertmanager-default'
       namespace: 'service-telemetry'
     type: Opaque
     stringData:
       alertmanager.yaml: |
       
       global:
         resolve_timeout: 10m
       route:
         group_by: ['job']
         group_wait: 30s
         group_interval: 5m
         repeat_interval: 12h
         receiver: 'null'
       receivers:
         - name: 'null'
     status:
     ...  
   ```
Manifest override parameter is defined in the spec of the ServiceTelemetry object.

End of the manifest override content.

5. Save and close.

5.2. ALERTS

You create alert rules in Prometheus and alert routes in Alertmanager. Alert rules in Prometheus servers send alerts to an Alertmanager, which manages the alerts. Alertmanager can silence, inhibit, or aggregate alerts, and send notifications using email, on-call notification systems, or chat platforms.

To create an alert, complete the following tasks:

1. Create an alert rule in Prometheus. For more information, see Section 5.2.1, “Creating an alert rule in Prometheus”.

2. Create an alert route in Alertmanager. For more information, see Section 5.2.3, “Creating an alert route in Alertmanager”.

Additional resources

For more information about alerts or notifications with Prometheus and Alertmanager, see https://prometheus.io/docs/alerting/overview/

To view an example set of alerts that you can use with Service Telemetry Framework (STF), see https://github.com/infrawatch/service-telemetry-operator/tree/master/deploy/alerts

5.2.1. Creating an alert rule in Prometheus

Prometheus evaluates alert rules to trigger notifications. If the rule condition returns an empty result set, the condition is false. Otherwise, the rule is true and it triggers an alert.

Procedure

1. Log in to Red Hat OpenShift Container Platform.

2. Change to the service-telemetry namespace:

   ```
   $ oc project service-telemetry
   ```

3. Create a PrometheusRule object that contains the alert rule. The Prometheus Operator loads the rule into Prometheus:

   ```
   $ oc apply -f - <<EOF
   apiVersion: monitoring.coreos.com/v1
   kind: PrometheusRule
   metadata:
     creationTimestamp: null
   labels:
     prometheus: default
     role: alert-rules
   name: prometheus-alarm-rules
   EOF
   ```
4. To verify that the rules have been loaded into Prometheus by the Operator, create a pod with access to curl:

   $ oc run curl --generator=run-pod/v1 --image=radial/busyboxplus:curl -i --tty

5. Run curl to access the prometheus-operated service to return the rules loaded into memory:

   [ root@curl:/ ]$ curl prometheus-operated:9090/api/v1/rules
   {
     "status":"success",
     "data":{
       "groups": [
         {
           "name": "/openstack.rules",
           "file": "/etc/prometheus/rules/prometheus-default-rulefiles-0/service-telemetry-prometheus-alarm-rules.yaml",
           "rules": [
             {
               "name": "Metric Listener down",
               "query": "collectd_qpid_router_status \u003c 1",
               "duration": 0,
               "labels": {},
               "annotations": {},
               "alerts": [],
               "health": "ok",
               "type": "alerting",
               "interval": 30
             }
           ]
         }
       ]
     }
   }

6. To verify that the output shows the rules loaded into the PrometheusRule object, for example the output contains the defined ./openstack.rules, exit from the pod:

   [ root@curl:/ ]$ exit

7. Clean up the environment by deleting the curl pod:

   $ oc delete pod curl
   pod "curl" deleted

Additional resources

For more information on alerting, see https://github.com/coreos/prometheus-operator/blob/master/Documentation/user-guides/alerting.md

5.2.2. Configuring custom alerts

You can add custom alerts to the PrometheusRule object that you created in Section 5.2.1, “Creating an alert rule in Prometheus”.

Procedure

1. Use the oc edit command:

   $ oc edit prometheusrules prometheus-alarm-rules

2. Edit the PrometheusRules manifest.
3. Save and close.

Additional resources

- For more information about configuring alerting rules, see https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/.
- For more information about PrometheusRules objects, see https://github.com/coreos/prometheus-operator/blob/master/Documentation/user-guides/alerting.md

5.2.3. Creating an alert route in Alertmanager

Use Alertmanager to deliver alerts to an external system, such as email, IRC, or other notification channel. The Prometheus Operator manages the Alertmanager configuration as an Red Hat OpenShift Container Platform (OCP) secret. STF by default deploys a basic configuration that results in no receivers:

```
alertmanager.yaml: |-
  global:
    resolve_timeout: 5m
  route:
    group_by: ['job']
    group_wait: 30s
    group_interval: 5m
    repeat_interval: 12h
    receiver: 'null'
  receivers:
    - name: 'null'
```

To deploy a custom Alertmanager route with STF, an `alertmanagerConfigManifest` parameter must be passed to the Service Telemetry Operator that results in an updated secret, managed by the Prometheus Operator.

Procedure

1. Log in to Red Hat OpenShift Container Platform.

2. Change to the `service-telemetry` namespace:

   ```
   $ oc project service-telemetry
   ```

3. Edit the `ServiceTelemetry` object for your STF deployment:

   ```
   $ oc edit stf default
   ```

4. Add a new parameter, `alertmanagerConfigManifest`, and the `Secret` object contents to define the `alertmanager.yaml` configuration for Alertmanager:
NOTE

This step loads the default template that is already managed by Service Telemetry Operator. To verify that the changes are populating correctly, change a value, return the alertmanager-default secret, and verify that the new value is loaded into memory. For example, change the value global.resolve_timeout from 5m to 10m.

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
spec:
  backends:
    metrics:
      prometheus:
        enabled: true
  alertmanagerConfigManifest: |
    apiVersion: v1
    kind: Secret
    metadata:
      name: 'alertmanager-default'
      namespace: 'service-telemetry'
      type: Opaque
    stringData:
      alertmanager.yaml: |-
        global:
          resolve_timeout: 10m
        route:
          group_by: ['job']
          group_wait: 30s
          group_interval: 5m
          repeat_interval: 12h
          receiver: 'null'
        receivers:
          - name: 'null'
```

5. Verify that the configuration was applied to the secret:

```
$ oc get secret alertmanager-default -o go-template='{{index .data
"alertmanager.yaml" | base64decode }}'
```

```
global:
  resolve_timeout: 10m
route:
  group_by: ['job']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
receivers:
  - name: 'null'
```
To verify the configuration has been loaded into Alertmanager, create a pod with access to `curl`:

```
$ oc run curl --generator=run-pod/v1 --image=radial/busyboxplus:curl -i --tty
```

Run `curl` against the `alertmanager-operated` service to retrieve the status and `configYAML` contents and review the supplied configuration matches the configuration loaded into Alertmanager:

```
[root@curl:/] $ curl alertmanager-operated:9093/api/v1/status
{
  "status": "success",
  "data": {
    "configYAML": "global:
      resolve_timeout: 10m
      http_config: {}
      smtp_hello: localhost
      smtp_require_tls: true
      pagerduty_url: https://events.pagerduty.com/v2/enqueue
      hipchat_api_url: https://api.hipchat.com
      wechat_api_url: https://api.weixin.qq.com/cgi-bin
      victorops_api_url: https://alert.victorops.com/integrations/generic/20131114/alert
      route:
        receiver: "null"
        group_by:
          - job
        group_wait: 30s
        group_interval: 5m
        repeat_interval: 12h
        receivers: 
          name: "null"
          templates: [
            "null"
          ]
    }
  }
```

8. Verify that the `configYAML` field contains the expected changes. Exit from the pod:

```
[root@curl:/] $ exit
```

9. To clean up the environment, delete the `curl` pod:

```
$ oc delete pod curl
pod "curl" deleted
```

Additional resources

- For more information about the Red Hat OpenShift Container Platform secret and the Prometheus operator, see Alerting.

## 5.3. CONFIGURING SNMP TRAPS

You can integrate Service Telemetry Framework (STF) with an existing infrastructure monitoring platform that receives notifications via SNMP traps. To enable SNMP traps, modify the `ServiceTelemetry` object and configure the `snmpTraps` parameters.

For more information about configuring alerts, see Section 5.2, “Alerts”.

### Prerequisites

- Know the IP address or hostname of the SNMP trap receiver where you want to send the alerts

### Procedure

1. To enable SNMP traps, modify the `ServiceTelemetry` object:

```
$ oc edit stf default
```
2. Set the `alerting.alertmanager.receivers.snmpTraps` parameters:

   ```yaml
   apiVersion: infra.watch/v1beta1
   kind: ServiceTelemetry
   ...
   spec:
   ...
   alerting:
      alertmanager:
         receivers:
            snmpTraps:
               enabled: true
               target: 10.10.10.10
   ```

3. Ensure that you set the value of `target` to the IP address or hostname of the SNMP trap receiver.

### 5.4. HIGH AVAILABILITY

High availability is the ability of Service Telemetry Framework (STF) to rapidly recover from failures in its component services. Although Red Hat OpenShift Container Platform (OCP) restarts a failed pod if nodes are available to schedule the workload, this recovery process might take more than one minute, during which time events and metrics are lost. A high availability configuration includes multiple copies of STF components, reducing recovery time to approximately 2 seconds. To protect against failure of an OCP node, deploy STF to an OCP cluster with three or more nodes.

**NOTE**

STF is not yet a fully fault tolerant system. Delivery of metrics and events during the recovery period is not guaranteed.

Enabling high availability has the following effects:

- Three ElasticSearch pods run instead of the default one.
- The following components run two pods instead of the default one:
  - AMQ Interconnect
  - Alertmanager
  - Prometheus
  - Events Smart Gateway
  - Collectd Metrics Smart Gateway
- Recovery time from a lost pod in any of these services reduces to approximately 2 seconds.

**NOTE**

The Ceilometer Metrics Smart Gateway is not yet HA.

### 5.4.1. Configuring high availability
To configure STF for high availability, add `highAvailability.enabled: true` to the ServiceTelemetry object in OCP. You can either set this parameter at installation time or, if you already deployed STF, complete the following steps:

**Procedure**

1. Log in to Red Hat OpenShift Container Platform.

2. Change to the `service-telemetry` namespace:
   
   ```
   $ oc project service-telemetry
   ```

3. Use the `oc` command to edit the ServiceTelemetry object:
   
   ```
   $ oc edit stf default
   ```

4. Add `highAvailability.enabled: true` to the `spec` section:
   
   ```yaml
   apiVersion: infra.watch/v1beta1
   kind: ServiceTelemetry
   ...
   spec:
   ...
   highAvailability:
   enabled: true
   ```

5. Save your changes and close the object.

### 5.5. DASHBOARDS

Use third-party application Grafana to visualize system-level metrics gathered by collectd for each individual host node.

For more information about configuring collectd, see Section 4.2, “Configuring Red Hat OpenStack Platform overcloud for Service Telemetry Framework”.

You can use two dashboards to monitor a cloud:

- **Infrastructure dashboard** Use the infrastructure dashboard to view metrics for a single node at a time. Select a node from the upper left corner of the dashboard.

- **Cloud view dashboard** Use the cloud view dashboard to view panels for monitoring service resource usage, API stats, and cloud events. You must enable API health monitoring and service monitoring to provide the data for this dashboard.
  
  - To enable API health monitoring, see Section 5.9, “Monitoring container health and API status”.
  
  - To enable service monitoring, see Section 5.8, “Monitoring the resource usage of Red Hat OpenStack Platform services”.

#### 5.5.1. Setting up Grafana to host the dashboard

Grafana is not included in the default Service Telemetry Framework (STF) deployment so you must
deploy the Grafana Operator from OperatorHub.io. Using the Service Telemetry Operator to
deploy Grafana results in a Grafana instance and the configuration of the default data sources for
the local STF deployment.

Prerequisites

Enable OperatorHub.io catalog source for the Grafana Operator. For more information, see
Section 3.1.5, “Enabling the OperatorHub.io Community Catalog Source”.

Procedure

1. Log in to Red Hat OpenShift Container Platform.

2. Change to the service-telemetry namespace:

   $ oc project service-telemetry

3. Deploy the Grafana operator:

   $ oc apply -f - <<EOF
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: grafana-operator
     namespace: service-telemetry
   spec:
     channel: alpha
     installPlanApproval: Automatic
     name: grafana-operator
     source: operatorhubio-operators
     sourceNamespace: openshift-marketplace
   EOF

4. To verify that the operator launched successfully, run the oc get csv command. If the value
   of the PHASE column is Succeeded, the operator launched successfully:

   $ oc get csv

   NAME                                DISPLAY                                         VERSION   REPLACES
   PHASE
   grafana-operator.v3.2.0             Grafana Operator                                3.2.0
   Succeeded
   ...

5. To launch a Grafana instance, create or modify the ServiceTelemetry object. Set
   graphing.enabled to true.

   $ oc edit stf default

   apiVersion: infra.watch/v1beta1
   kind: ServiceTelemetry
   ...
   spec:
6. Verify that the Grafana instance deployed:

   $ oc get pod -l app=grafana

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>grafana-deployment-7fc7848b56-sbkhv</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>1m</td>
</tr>
</tbody>
</table>

5.5.2. Importing dashboards

The Grafana Operator can import and manage dashboards by creating GrafanaDashboard objects. You can view example dashboards at https://github.com/infrawatch/dashboards.

Procedure

1. Import a dashboard:


   grafanadashboard.integreatly.org/rhos-dashboard created

2. Verify that the resources installed correctly:

   $ oc get grafanadashboards

<table>
<thead>
<tr>
<th>NAME</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>rhos-dashboard</td>
<td>7d21h</td>
</tr>
</tbody>
</table>

   $ oc get grafanadatasources

<table>
<thead>
<tr>
<th>NAME</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>default-ds-prometheus</td>
<td>20h</td>
</tr>
</tbody>
</table>

3. Expose the grafana service as a route:

   $ oc create route edge dashboards --service=grafana-service --insecure-policy="Redirect" --port=3000

4. Retrieve the Grafana route address:

   $ oc get route dashboards

<table>
<thead>
<tr>
<th>NAME</th>
<th>HOST/PORT</th>
<th>PATH</th>
<th>SERVICES</th>
</tr>
</thead>
<tbody>
<tr>
<td>dashboards</td>
<td>dashboards-service-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The HOST/PORT value is the Grafana route address.

5. Navigate to https://<GRAFANA-ROUTE-ADDRESS> in a web browser. Replace <GRAFANA-ROUTE-ADDRESS> with the HOST/PORT value that you retrieved in the previous step.

6. To view the dashboard, click Dashboards and Manage.

5.5.3. Viewing and editing queries

Procedure

1. Log in to Red Hat OpenShift Container Platform. To view and edit queries, log in as the admin user.

2. Change to the service-telemetry namespace:

   $ oc project service-telemetry

3. To retrieve the default username and password, describe the Grafana object using the oc describe command:

   $ oc describe grafana default

   TIP

   To set the admin username and password through the ServiceTelemetry object, use the graphing.grafana.adminUser and graphing.grafana.adminPassword parameters.

5.6. MULTIPLE CLOUD CONFIGURATION

You can configure multiple Red Hat OpenStack Platform clouds to target a single instance of Service Telemetry Framework (STF):

1. Plan the AMQP address prefixes that you want to use for each cloud. For more information, see Section 5.6.1, “Planning AMQP address prefixes”.

2. Deploy metrics and events consumer Smart Gateways for each cloud to listen on the corresponding address prefixes. For more information, see Section 5.6.2, “Deploying Smart Gateways”.

3. Configure each cloud to send its metrics and events to STF on the correct address. For more information, see Section 5.6.4, “Creating the OpenStack environment file”. 
5.6.1. Planning AMQP address prefixes

By default, Red Hat OpenStack Platform nodes get data through two data collectors; collectd and Ceilometer. These components send telemetry data or notifications to the respective AMQP addresses, for example, collectd/telemetry, where STF Smart Gateways listen on those addresses for monitoring data. To support multiple clouds and to identify which cloud generated the monitoring data, configure each cloud to send data to a unique address. Prefix a cloud identifier to the second part of the address. The following list shows some example addresses and identifiers:

- collectd/cloud1-telemetry
- collectd/cloud1-notify
- anycast/ceilometer/cloud1-metering.sample
- anycast/ceilometer/cloud1-event.sample
- collectd/cloud2-telemetry
- collectd/cloud2-notify
5.6.2. Deploying Smart Gateways

You must deploy a Smart Gateway for each of the data collection types for each cloud; one for collectd metrics, one for collectd events, one for Ceilometer metrics, and one for Ceilometer events. Configure each of the Smart Gateways to listen on the AMQP address that you define for the corresponding cloud. Smart Gateways are defined via the `clouds` parameter in the `ServiceTelemetry` manifest.

When you deploy STF for the first time, Smart Gateway manifests are created that define the initial Smart Gateways for a single cloud. When deploying Smart Gateways for multiple cloud support, you deploy multiple Smart Gateways for each of the data collection types that handle the metrics and the events data for each cloud. The initial Smart Gateways are defined under `cloud1` with the following subscription addresses:

<table>
<thead>
<tr>
<th>collector</th>
<th>type</th>
<th>default subscription address</th>
</tr>
</thead>
<tbody>
<tr>
<td>collectd</td>
<td>metrics</td>
<td>collectd/telemetry</td>
</tr>
<tr>
<td>collectd</td>
<td>events</td>
<td>collectd/notify</td>
</tr>
<tr>
<td>Ceilometer</td>
<td>metrics</td>
<td>anycast/ceilometer/metering.sample</td>
</tr>
<tr>
<td>Ceilometer</td>
<td>events</td>
<td>anycast/ceilometer/event.sample</td>
</tr>
</tbody>
</table>

Prerequisites

You have determined your naming scheme and have created your list of clouds objects. For more information about determining your naming scheme, see [ ]. For more information about creating the content for the `clouds` parameter, see xref:clouds_assembly-installing-the-core-components-of-stf[.  

Procedure

1. Log in to Red Hat OpenShift Container Platform.
2. Change to the `service-telemetry` namespace:

   $ oc project service-telemetry

3. Edit the `default` ServiceTelemetry object and add the `clouds` parameter with your configuration:

   $ oc edit stf default
   ```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
...
spec:
...
clouds:
- name: cloud1
  events:
    collectors:
    - collectorType: collectd
      subscriptionAddress: collectd/cloud1-notify
    - collectorType: ceilometer
      subscriptionAddress: anycast/ceilometer/cloud1-event.sample
metrics:
    collectors:
    - collectorType: collectd
      subscriptionAddress: collectd/cloud1-telemetry
    - collectorType: ceilometer
      subscriptionAddress: anycast/ceilometer/cloud1-metering.sample
- name: cloud2
  events:
  ...

4. Save the ServiceTelemetry object.

5. Verify that each Smart Gateway is running. This can take several minutes depending on the number of Smart Gateways:

   $ oc get po -l app=smart-gateway

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>default-cloud1-ceil-event-smartgateway-6cfb65478c-g5q82</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>13h</td>
</tr>
<tr>
<td>default-cloud1-ceil-meter-smartgateway-58f885c76d-xmxwn</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>13h</td>
</tr>
<tr>
<td>default-cloud1-coll-event-smartgateway-58fbbd4485-rl9bd</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>13h</td>
</tr>
<tr>
<td>default-cloud1-coll-meter-smartgateway-7c6fc495c4-jn728</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>13h</td>
</tr>
</tbody>
</table>

5.6.3. Deleting the default Smart Gateways

After you configure STF for multiple clouds, you can delete the default Smart Gateways if they are no longer in use. The Service Telemetry Operator can remove SmartGateway objects that have been created but are no longer listed in the ServiceTelemetry clouds list of objects. You can enable the removal of SmartGateway objects that are not defined by the clouds parameter by setting cloudsRemoveOnMissing: true in the ServiceTelemetry manifest.

TIP

If you do not want any Smart Gateways deployed, define an empty clouds object using the clouds: {} parameter.
WARNING

The `cloudsRemoveOnMissing` parameter is disabled by default. If you enable the `cloudsRemoveOnMissing` parameter, you remove any manually created SmartGateway objects in the current namespace without any possibility to restore.

Procedure

1. Define your `clouds` parameter with the list of cloud objects to be managed by the Service Telemetry Operator. For more information, see Section 3.2.2, “clouds”.

2. Edit the ServiceTelemetry object and add the `cloudsRemoveOnMissing` parameter:

```yaml
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  ...
spec:
  ...
  cloudsRemoveOnMissing: true
  clouds:
    ...
```

3. Save the modifications.

4. Verify that the Operator deleted the Smart Gateways. This can take several minutes while the Operators reconcile the changes:

```
$ oc get smartgateways
```

5.6.4. Creating the OpenStack environment file

To label traffic according to the cloud of origin, you must create a configuration with cloud-specific instance names. Create an `stf-connectors.yaml` file and adjust the values of `CeilometerQdrEventsConfig`, `CeilometerQdrMetricsConfig` and `CollectdAmqpInstances` to match the AMQP address prefix scheme.

NOTE

If you enabled container health and API status monitoring, you must also modify the `CollectdSensubilityResultsChannel` parameter. For more information, see Section 5.9, “Monitoring container health and API status”.
WARNING

Remove enable-stf.yaml and ceilometer-write-qdr.yaml environment file references from your overcloud deployment. This configuration is redundant and results in duplicate information being sent from each cloud node.

Procedure

1. Create the stf-connectors.yaml file and modify it to match the AMQP address that you want for this cloud deployment:

```
resource_registry:
  OS::TripleO::Services::Collectd: /usr/share/openstack-tripleo-heat-templates/deployment/metrics/collectd-container-puppet.yaml
  OS::TripleO::Services::MetricsQdr: /usr/share/openstack-tripleo-heat-templates/deployment/metrics/qdr-container-puppet.yaml
  OS::TripleO::Services::CeilometerAgentCentral: /usr/share/openstack-tripleo-heat-templates/deployment/ceilometer/ceilometer-agent-central-container-puppet.yaml
  OS::TripleO::Services::CeilometerAgentNotification: /usr/share/openstack-tripleo-heat-templates/deployment/ceilometer/ceilometer-agent-notification-container-puppet.yaml
  OS::TripleO::Services::CeilometerAgentIpmi: /usr/share/openstack-tripleo-heat-templates/deployment/ceilometer/ceilometer-agent-ipmi-container-puppet.yaml
  OS::TripleO::Services::ComputeCeilometerAgent: /usr/share/openstack-tripleo-heat-templates/deployment/ceilometer/ceilometer-agent-compute-container-puppet.yaml
  OS::TripleO::Services::Redis: /usr/share/openstack-tripleo-heat-templates/deployment/database/redis-pacemaker-puppet.yaml

parameter_defaults:
  EnableSTF: true
  EventPipelinePublishers: []
  PipelinePublishers: []
  CeilometerEnablePanko: false
  CeilometerQdrPublishEvents: true
  CeilometerQdrEventsConfig:
    driver: amqp
    topic: cloud1-event
  CeilometerQdrMetricsConfig:
    driver: amqp
    topic: cloud1-metering
  CollectdConnectionType: amqp1
  CollectdAmqpInterval: 5
  CollectdDefaultPollingInterval: 5
  CollectdAmqpInstances:
    cloud1-notify:
```
Define the topic for Ceilometer events. This value is the address format of anycast/ceilometer/cloud1-event.sample.

Define the topic for Ceilometer metrics. This value is the address format of anycast/ceilometer/cloud1-metering.sample.

Define the topic for collectd events. This value is the format of collectd/cloud1-notify.

Define the topic for collectd metrics. This value is the format of collectd/cloud1-telemetry.

Define the topic for collectd-sensubility events. This should be the exact string format of collectd/cloud1-notify.

Adjust the **MetricsQdrConnectors** host to the address of the STF route.

Enable monitoring of health and API status.

2. Ensure that the naming convention in the stf-connectors.yaml file aligns with the spec.amqpUrl field in the Smart Gateway configuration. For example, configure the **CeilometerQdrEventsConfig.topic** field to a value of cloud1-event.

3. Save the file in a directory for custom environment files, for example /home/stack/customTemplates/.

4. Source the authentication file:

   ```bash
   [stack@undercloud-0 ~]$ source stackrc
   (undercloud) [stack@undercloud-0 ~]$ 
   ```
5. Include the `stf-connectors.yaml` file in the `overcloud deployment` command, along with any other environment files relevant to your environment:

```
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud deploy \
--templates /usr/share/openstack-tripleo-heat-templates \
... \
-e /home/stack/custom_templates/stf-connectors.yaml \
... 
```

Additional resources

- For information about validating the deployment, see Section 4.2.3, “Validating client-side installation”.

5.6.5. Querying metrics data from multiple clouds

Data stored in Prometheus has a service label attached according to the Smart Gateway it was scraped from. You can use this label to query data from a specific cloud.

To query data from a specific cloud, use a Prometheus `promql` query that matches the associated service label; for example:

```
collectd_uptime{service="default-cloud1-coll-meter-smartgateway"}
```

5.7. EPHEMERAL STORAGE

You can use ephemeral storage to run Service Telemetry Framework (STF) without persistently storing data in your Red Hat OpenShift Container Platform (OCP) cluster.

**WARNING**

If you use ephemeral storage, you might experience data loss if a pod is restarted, updated, or rescheduled onto another node. Use ephemeral storage only for development or testing, and not production environments.

5.7.1. Configuring ephemeral storage

To configure STF components for ephemeral storage, add `...storage.strategy: ephemeral` to the corresponding parameter. For example, to enable ephemeral storage for the Prometheus backend, set `backends.metrics.prometheus.storage.strategy: ephemeral`. Components that support configuration of ephemeral storage include `alerting.alertmanager`, `backends.metrics.prometheus`, and `backends.events.elasticsearch`. You can add ephemeral storage configuration at installation time or, if you already deployed STF, complete the following steps:

Procedure

1. Log in to Red Hat OpenShift Container Platform.
2. Change to the `service-telemetry` namespace:

```
$ oc project service-telemetry
```
3. Edit the ServiceTelemetry object:

   $ oc edit stf default

4. Add the `...storage.strategy: ephemeral` parameter to the `spec` section of the relevant component:

   ```yaml
   apiVersion: infra.watch/v1beta1
   kind: ServiceTelemetry
   metadata:
     name: stf-default
     namespace: service-telemetry
   spec:
     alerting:
       enabled: true
     alertmanager:
       storage:
         strategy: ephemeral
     backends:
       metrics:
         prometheus:
           enabled: true
           storage:
             strategy: ephemeral
       events:
         elasticsearch:
           enabled: true
           storage:
             strategy: ephemeral
   ```

5. Save your changes and close the object.

5.8. MONITORING THE RESOURCE USAGE OF RED HAT OPENSTACK PLATFORM SERVICES

Monitor the resource usage of the Red Hat OpenStack Platform services, such as the APIs and other infrastructure processes, to identify bottlenecks in the overcloud by showing services running out of compute power. Enable the `collectd-libpod-stats` plugin to gather CPU and memory usage metrics for every container running in the overcloud.

Prerequisites

- You have created the `stf-connectors.yaml` file. For more information, see Section 4.2, “Configuring Red Hat OpenStack Platform overcloud for Service Telemetry Framework”.
- You are using the most current version of Red Hat OpenStack Platform: 16.1.

Procedure

1. Open the `stf-connectors.yaml` file.

2. Add the following configuration to `parameter_defaults`:

   ```yaml
   CollectdEnableLibpodstats: true
   ```
3. Continue with the overcloud deployment procedure.

5.9. MONITORING CONTAINER HEALTH AND API STATUS

Container health assesses the status of each of the Red Hat OpenStack Platform service containers by periodically running a health check script using the OCI (Open Container Initiative) standard. Most Red Hat OpenStack Platform services implement a health check that logs issues and returns a binary status. For the Red Hat OpenStack Platform APIs, the health checks query the root endpoint and determine the health based on the response time.

To monitor healthchecks in Service Telemetry Framework (STF), you must enable and configure the collectd-sensubility plugin to work with the amqp1 protocol. The STF architecture considers healthcheck results to be events and they are stored in ElasticSearch.

Prerequisites

- You have created the stf-connectors.yaml file in your templates directory. For more information, see Section 4.2, “Configuring Red Hat OpenStack Platform overcloud for Service Telemetry Framework”.

- You are using the most current version of Red Hat OpenStack Platform 16.1.

Procedure

1. Open the stf-connectors.yaml and edit the collectd-sensubility parameter:

   ```yaml
   CollectdEnableSensubility: true
   CollectdSensubilityTransport: amqp1
   ```

2. If your environment has multiple clouds, configure the collectd-sensubility events channel with the new collectd events address. Edit the stf-connectors.yaml file:

   ```yaml
   CollectdSensubilityResultsChannel: collectd/cloudprefix-notify
   ```

Additional resources

- For more information about multiple cloud addresses, see Section 5.6, “Multiple cloud configuration”.

5.10. CREATING A ROUTE IN RED HAT OPENSSHIFT CONTAINER PLATFORM

In Red Hat OpenShift Container Platform, you can expose applications to the external network via a route. For more information, see Configuring ingress cluster traffic.

In Service Telemetry Framework (STF), routes are not exposed by default to limit the attack surface of STF deployments. To access some services deployed in STF, you must expose the services in OCP for access.

A common service to expose in STF is Prometheus, as shown in the following example:

Procedure
1. Log in to Red Hat OpenShift Container Platform.

2. Change to the `service-telemetry` namespace:

   ```bash
   $ oc project service-telemetry
   ```

3. List the available services in the `service-telemetry` project:

   ```bash
   $ oc get services
   ```

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alertmanager-operated</td>
<td>ClusterIP</td>
<td>None</td>
<td>&lt;none&gt;</td>
<td>9093/TCP,9094/TCP,9094/UDP</td>
</tr>
<tr>
<td>default-cloud1-ceil-meter-smartgateway</td>
<td>ClusterIP</td>
<td>172.30.114.195</td>
<td>&lt;none&gt;</td>
<td>8081/TCP</td>
</tr>
<tr>
<td>default-cloud1-coll-meter-smartgateway</td>
<td>ClusterIP</td>
<td>172.30.133.180</td>
<td>&lt;none&gt;</td>
<td>8081/TCP</td>
</tr>
<tr>
<td>default-interconnect</td>
<td>ClusterIP</td>
<td>172.30.3.241</td>
<td>&lt;none&gt;</td>
<td>5672/TCP,55671/TCP,5671/TCP,5673/TCP,5671/TCP,5673/TCP</td>
</tr>
<tr>
<td>ibm-auditlogging-operator-metrics</td>
<td>ClusterIP</td>
<td>172.30.216.249</td>
<td>&lt;none&gt;</td>
<td>8383/TCP,8686/TCP</td>
</tr>
<tr>
<td>prometheus-operated</td>
<td>ClusterIP</td>
<td>None</td>
<td>&lt;none&gt;</td>
<td>9090/TCP</td>
</tr>
<tr>
<td>service-telemetry-operator-metrics</td>
<td>ClusterIP</td>
<td>172.30.11.66</td>
<td>&lt;none&gt;</td>
<td>8383/TCP,8686/TCP</td>
</tr>
<tr>
<td>smart-gateway-operator-metrics</td>
<td>ClusterIP</td>
<td>172.30.145.199</td>
<td>&lt;none&gt;</td>
<td>8383/TCP,8686/TCP</td>
</tr>
</tbody>
</table>

4. Take note of the port and service name to expose as a route, for example, service `prometheus-operated` and port 9090.

5. Expose the `prometheus-operated` service as an edge route and redirect insecure traffic to the secure endpoint of port 9090:

   ```bash
   $ oc create route edge metrics-store --service=prometheus-operated --insecure-policy="Redirect" --port=9090
   ```

   `route.route.openshift.io/metrics-store created`

6. To verify and find the exposed external DNS for the route, use the `oc get route` command:

   ```bash
   $ oc get route metrics-store -o go-template='{{.spec.host}}'
   ```

   `metrics-store-service-telemetry.apps.infra.watch`

   The `prometheus-operated` service is now available at the exposed DNS address, for example, `https://metrics-store-service-telemetry.apps.infra.watch`

   **NOTE**

   The address of the route must be resolvable and configuration is environment specific.
Additional resources

- For more information about Red Hat OpenShift Container Platform networking, see *Understanding networking*

- For more information about route configuration, see *Route configuration*

- For more information about ingress cluster traffic, see *Configuring ingress cluster traffic overview*
CHAPTER 6. UPGRADING SERVICE TELEMETRY FRAMEWORK TO VERSION 1.2

To migrate from Service Telemetry Framework (STF) v1.1 to STF v1.2, you must replace the `ClusterServiceVersion` and `Subscription` objects in the `service-telemetry` namespace on your Red Hat OpenShift Container Platform (OCP) v4.5 environment.

Prerequisites

- When you upgrade the OCP environment from v4.5 to v4.6, ensure that you add the AMQ Certificate Manager Operator from the `redhat-operators-stf` CatalogSource. For more information, see [Subscribing to the AMQ Certificate Manager Operator](#).
- Ensure that you backup data prior to any upgrade of the environment. Upgrading STF v1.1 to v1.2 results in a brief outage while the Smart Gateways are upgraded. Additionally, changes to the `ServiceTelemetry` and `SmartGateway` objects will not have any effect while the Operators are being replaced.

To upgrade from STF v1.1 to v1.2, complete the following procedures:

Procedure

1. **Remove the STF 1.1 Operators**

2. **Remove the STF 1.1 CatalogSource**

3. **Subscribe to the Service Telemetry Operator**

### 6.1. REMOVING SERVICE TELEMETRY FRAMEWORK 1.1 OPERATORS

Remove the Operators from STF v1.1, Smart Gateway Operator, and Service Telemetry Operator.

Procedure

1. **Retrieve the Subscription name of the Smart Gateway Operator:**

   ```bash
   $ oc get sub smart-gateway-operator-stable-redhat-operators-stf-openshift-marketplace | grep smart-gateway
   smart-gateway-operator-stable-redhat-operators-stf-openshift-marketplace   smart-gateway-operator   redhat-operators-stf   stable
   ```

2. **Delete the Smart Gateway Operator subscription:**

   ```bash
   $ oc delete sub smart-gateway-operator-stable-redhat-operators-stf-openshift-marketplace
   subscription.operators.coreos.com "smart-gateway-operator-stable-redhat-operators-stf-openshift-marketplace" deleted
   ```

3. **Delete the Service Telemetry Operator subscription:**
4. Get the Smart Gateway Operator and Service Telemetry Operator ClusterServiceVersion:

```
$ oc get csv -o name | grep -E service-telemetry|smart-gateway
```

```
clusterserviceversion.operators.coreos.com/service-telemetry-operator.v1.1.3
clusterserviceversion.operators.coreos.com/smart-gateway-operator.v2.1.3
```

5. Delete the Service Telemetry Operator ClusterServiceVersion:

```
$ oc delete clusterserviceversion.operators.coreos.com/service-telemetry-operator.v1.1.3
```

```
clusterserviceversion.operators.coreos.com "service-telemetry-operator.v1.1.3" deleted
```

6. Delete the Smart Gateway Operator ClusterServiceVersion:

```
$ oc delete clusterserviceversion.operators.coreos.com/smart-gateway-operator.v2.1.3
```

```
clusterserviceversion.operators.coreos.com "smart-gateway-operator.v2.1.3" deleted
```

### 6.2. REMOVING THE SERVICE TELEMETRY FRAMEWORK V1.1 CATALOGSOURCE

Remove the STF v1.1 CatalogSource. STF v1.2 is shipped in the default-enabled redhat-operators CatalogSource.

Procedure

- Delete the redhat-operators-stf CatalogSource:

```
$ oc delete catalogsource -n openshift-marketplace redhat-operators-stf
```

```
catalogsource.operators.coreos.com "redhat-operators-stf" deleted
```

### 6.3. SUBSCRIBING TO THE SERVICE TELEMETRY OPERATOR

You must subscribe to the Service Telemetry Operator, which manages the STF instances.

Procedure

1. To create the Service Telemetry Operator subscription, enter the `oc apply -f` command:

```
$ oc apply -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: service-telemetry-operator
EOF
```
To validate the Service Telemetry Operator and the dependent operators, enter the following command:

```
$ oc get csv --namespace service-telemetry
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>DISPLAY</th>
<th>VERSION</th>
<th>REPLACES</th>
</tr>
</thead>
<tbody>
<tr>
<td>amq7-cert-manager.v1.0.0</td>
<td>Red Hat Integration - AMQ Certificate Manager</td>
<td>1.0.0</td>
<td></td>
</tr>
<tr>
<td>Succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>amq7-interconnect-operator.v1.2.3</td>
<td>Red Hat Integration - AMQ Interconnect</td>
<td>1.2.3</td>
<td></td>
</tr>
<tr>
<td>Succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>elastic-cloud-eck.v1.4.0</td>
<td>Elasticsearch (ECK) Operator</td>
<td>1.4.0</td>
<td></td>
</tr>
<tr>
<td>Succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grafana-operator.v3.9.0</td>
<td>Grafana Operator</td>
<td>3.9.0</td>
<td></td>
</tr>
<tr>
<td>Succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prometheusoperator.0.37.0</td>
<td>Prometheus Operator</td>
<td>0.37.0</td>
<td></td>
</tr>
<tr>
<td>Succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>service-telemetry-operator.v1.2.1</td>
<td>Service Telemetry Operator</td>
<td>1.2.1</td>
<td></td>
</tr>
<tr>
<td>Succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smart-gateway-operator.v2.2.1</td>
<td>Smart Gateway Operator</td>
<td>2.2.1</td>
<td></td>
</tr>
<tr>
<td>Succeeded</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When the new Operators start, they reconcile the existing `ServiceTelemetry` and `SmartGateway` objects, resulting in the restart of Smart Gateway containers.

- To check the state of the Smart Gateway containers, use the `oc get pods` command:

```
oc get pods
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>default-cloud1-ceil-meter-smartgateway-5849c4cdb5-xgl42</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>35s</td>
</tr>
<tr>
<td>default-cloud1-coll-meter-smartgateway-749674f75c-k7pm7</td>
<td>2/2</td>
<td>Terminating</td>
<td>0</td>
<td>56m</td>
</tr>
<tr>
<td>default-cloud1-coll-meter-smartgateway-868476456b-ksh9b</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>26s</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 7. COLLECTD PLUGINS

IMPORTANT

Red Hat is currently updating the plugin information in this guide for this release.

You can configure multiple collectd plugins to suit your Red Hat OpenStack Platform 16.1 environment. Reference the tables of available parameters for specific plugins, such as in the following example:

```
ExtraConfig:
  collectd::plugin::example_plugin::<parameter>: <value>
```

Reference the metrics tables of specific plugins for Prometheus or Grafana queries.

**collectd::plugin::aggregation**

You can aggregate several values into one with the aggregation plugin. Use the aggregation functions such as sum, average, min, and max to calculate metrics, for example average and total CPU statistics.

- `collectd::plugin::aggregation::aggregators`
- `collectd::plugin::aggregation::interval`

**collectd::plugin::amqp**

Use the amqp1 plugin to write values to an amqp1 message bus, for example, AMQ Interconnect.

Table 7.1. amqp1 parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>manage_package</td>
<td>Boolean</td>
</tr>
<tr>
<td>transport</td>
<td>String</td>
</tr>
<tr>
<td>host</td>
<td>string</td>
</tr>
<tr>
<td>port</td>
<td>integer</td>
</tr>
<tr>
<td>user</td>
<td>String</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
</tr>
<tr>
<td>address</td>
<td>String</td>
</tr>
<tr>
<td>instances</td>
<td>Hash</td>
</tr>
<tr>
<td>retry_delay</td>
<td>Integer</td>
</tr>
</tbody>
</table>
### send_queue_limit

**Parameter**

send_queue_limit

**Type**

Integer

### interval

**Parameter**

interval

**Type**

Integer

---

### Example configuration

Parameter Defaults:

`CollectdExtraPlugins:
- amqp1`

ExtraConfig:

`collectd::plugin::amqp1::send_queue_limit: 50`

---

### collectd::plugin::apache

Use the **apache** plugin to collect Apache data.

**Table 7.2. apache parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>instances</td>
<td>Hash</td>
</tr>
<tr>
<td>interval</td>
<td>Integer</td>
</tr>
<tr>
<td>manage-package</td>
<td>Boolean</td>
</tr>
<tr>
<td>package_install_options</td>
<td>List</td>
</tr>
</tbody>
</table>

---

### Example configuration

Parameter Defaults:

ExtraConfig:

`collectd::plugin::apache:
localhost:
url: "http://10.0.0.111/status?auto"`

---

### Additional resources

For more information about configuring the **apache** plugin, see **apache**.

---

### collectd::plugin::battery

Use the **battery** plugin to report the remaining capacity, power, or voltage of laptop batteries.

**Table 7.3. battery parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>values_percentage</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
Additional resources

For more information about configuring the battery plugin, see battery.

collectd::plugin::bind
Use the bind plugin to retrieve encoded statistics about queries and responses from a DNS server. The plugin submits the values to collectd.

collectd::plugin::ceph
Use the ceph plugin to gather data from ceph daemons.

Table 7.4. ceph parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>daemons</td>
<td>Array</td>
</tr>
<tr>
<td>longrunavglatency</td>
<td>Boolean</td>
</tr>
<tr>
<td>convertspecialmetricypes</td>
<td>Boolean</td>
</tr>
<tr>
<td>manage_package</td>
<td>Boolean</td>
</tr>
<tr>
<td>package_name</td>
<td>String</td>
</tr>
</tbody>
</table>

Example configuration

```yaml
parameter_defaults:
  ExtraConfig:
    collectd::plugin::ceph::daemons:
    - ceph-osd.0
    - ceph-osd.1
    - ceph-osd.2
    - ceph-osd.3
    - ceph-osd.4
```

NOTE

If an Object Storage Daemon (OSD) is not on every node, you must list the OSDs.
NOTE
When you deploy collectd, the ceph plugin is added to the ceph nodes. Do not add the ceph plugin on ceph nodes to CollectdExtraPlugins, because this results in a deployment failure.

Additional resources
For more information about configuring the ceph plugin, see ceph.

collectd::plugins::cgroups
Use the cgroups plugin to collect information for processes in a cgroup.

Table 7.5. cgroups parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ignore_selected</td>
<td>Boolean</td>
</tr>
<tr>
<td>interval</td>
<td>Integer</td>
</tr>
<tr>
<td>cgroups</td>
<td>List</td>
</tr>
</tbody>
</table>

Additional resources
For more information about configuring the cgroups plugin, see cgroups.

collectd::plugin::connectivity
Use the connectivity plugin to monitor the state of network interfaces.

NOTE
If no interfaces are listed, all interfaces are monitored by default.

Table 7.6. connectivity parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>interfaces</td>
<td>Array</td>
</tr>
</tbody>
</table>

Example configuration

```
    parameter_defaults:
      ExtraConfig:
        collectd::plugin::connectivity::interfaces:
          - eth0
          - eth1
```

Additional resources
For more information about configuring the connectivity plugin, see connectivity.
collectd::plugin::conntrack
Use the `conntrack` plugin to track the number of entries in the Linux connection-tracking table. There are no parameters for this plugin.

collectd::plugin::contextswitch
Use the `ContextSwitch` plugin to collect the number of context switches handled by the system.

Additional resources
For more information about configuring the `contextswitch` plugin, see `contextswitch`.

collectd::plugin::cpu
Use the `cpu` plugin to monitor the time the CPU spends in various states, for example, idle, executing user code, executing system code, waiting for IO-operations, and other states.

The `cpu` plugin collects _jiffies_, not percentage values. The value of a jiffy depends on the clock frequency of your hardware platform, and therefore is not an absolute time interval unit.

To report a percentage value, set the Boolean parameters `reportbycpu` and `reportbystate` to true, and then set the Boolean parameter `valuespercentage` to true.

Table 7.7. cpu metrics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>Amount of idle time</td>
<td><code>collectd_cpu_total{...type_instance=idle}</code></td>
</tr>
<tr>
<td>interrupt</td>
<td>CPU blocked by interrupts</td>
<td><code>collectd_cpu_total{...type_instance=interrupt}</code></td>
</tr>
<tr>
<td>nice</td>
<td>Amount of time running low priority processes</td>
<td><code>collectd_cpu_total{...type_instance=nice}</code></td>
</tr>
<tr>
<td>softirq</td>
<td>Amount of cycles spent in servicing interrupt requests</td>
<td><code>collectd_cpu_total{...type_instance=softirq}</code></td>
</tr>
<tr>
<td>steal</td>
<td>The percentage of time a virtual CPU waits for a real CPU while the hypervisor is servicing another virtual processor</td>
<td><code>collectd_cpu_total{...type_instance=steal}</code></td>
</tr>
<tr>
<td>system</td>
<td>Amount of time spent on system level (kernel)</td>
<td><code>collectd_cpu_total{...type_instance=system}</code></td>
</tr>
<tr>
<td>user</td>
<td>Jiffies used by user processes</td>
<td><code>collectd_cpu_total{...type_instance=user}</code></td>
</tr>
<tr>
<td>wait</td>
<td>CPU waiting on outstanding I/O request</td>
<td><code>collectd_cpu_total{...type_instance=wait}</code></td>
</tr>
</tbody>
</table>

Table 7.8. cpu parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportbystate</td>
<td>Boolean</td>
</tr>
<tr>
<td>valuespercentage</td>
<td>Boolean</td>
</tr>
<tr>
<td>reportbycpu</td>
<td>Boolean</td>
</tr>
<tr>
<td>reportnumcpu</td>
<td>Boolean</td>
</tr>
<tr>
<td>reportgueststate</td>
<td>Boolean</td>
</tr>
<tr>
<td>subtractgueststate</td>
<td>Boolean</td>
</tr>
<tr>
<td>interval</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**Example configuration**

```yaml
parameter_defaults:
  CollectdExtraPlugins:
  - cpu
  ExtraConfig:
    collectd::plugin::cpu::reportbystate: true
```

**Additional resources**

For more information about configuring the `cpu` plugin, see `cpu`.

- `collectd::plugin::cpufreq`
  - None

- `collectd::plugin::cpusleep`

- `collectd::plugin::csv`
  - `collectd::plugin::csv::datadir`
  - `collectd::plugin::csv::storerates`
  - `collectd::plugin::csv::interval`

- `collectd::plugin::curl_json`
- `collectd::plugin::curl`
- `collectd::plugin::curl_xml`
- `collectd::plugin::dbi`
- `collectd::plugin::df`

Use the `df` plugin to collect disk space usage information for file systems.

**Table 7.9. df metrics**
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>free</td>
<td>Amount of free disk space</td>
<td><code>collectd_df_df_complex[... , type_instance=&quot;free&quot;]</code></td>
</tr>
<tr>
<td>reserved</td>
<td>Amount of reserved disk space</td>
<td><code>collectd_df_df_complex[... , type_instance=&quot;reserved&quot;]</code></td>
</tr>
<tr>
<td>used</td>
<td>Amount of used disk space</td>
<td><code>collectd_df_df_complex[... , type_instance=&quot;used&quot;]</code></td>
</tr>
</tbody>
</table>

Table 7.10. df parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>devices</td>
<td>Array</td>
</tr>
<tr>
<td>fstypes</td>
<td>Array</td>
</tr>
<tr>
<td>ignoreselected</td>
<td>Boolean</td>
</tr>
<tr>
<td>mountpoints</td>
<td>Array</td>
</tr>
<tr>
<td>reportbydevice</td>
<td>Boolean</td>
</tr>
<tr>
<td>reportinodes</td>
<td>Boolean</td>
</tr>
<tr>
<td>reportreserved</td>
<td>Boolean</td>
</tr>
<tr>
<td>valuesabsolute</td>
<td>Boolean</td>
</tr>
<tr>
<td>valuespercentage</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

Example configuration

```yaml
parameter_defaults:
  CollectdExtraPlugins:
    - df
ExtraConfig:
  collectd::plugin::df::FStype: "ext4"
```

Additional resources

For more information about configuring the `df` plugin, see [df](#).

**collectd::plugin::disk**

Use the `disk` plugin to collect performance statistics of hard-disks and, if supported, partitions. This plugin is enabled by default.

Table 7.11. disk parameters
### Table 7.12. disk metrics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>merged</td>
<td>The number of operations that can be merged together, already queued operations, for example, one physical disk access served two or more logical operations.</td>
</tr>
<tr>
<td>time</td>
<td>The average time an I/O-operation takes to complete. Since this is a little messy to calculate take the actual values with a grain of salt.</td>
</tr>
<tr>
<td>io_time</td>
<td>Time spent doing I/Os (ms). You can make use of this metric as a device load percentage. A value of 1 second matches 100% of load.</td>
</tr>
<tr>
<td>weighted_io_time</td>
<td>Measure of both I/O completion time and the backlog that might be accumulating.</td>
</tr>
<tr>
<td>pending_operations</td>
<td>Shows queue size of pending I/O operations.</td>
</tr>
</tbody>
</table>

### Example configuration

```
parameter_defaults:
  ExtraConfig:
    collectd::plugin::disk::disk: "sda"
    collectd::plugin::disk::ignoreselected: false
```

### Additional resources

For more information about configuring the `disk` plugin, see `disk`.

- `collectd::plugin::dns`
- `collectd::plugin::dpdk_telemetry`
- `collectd::plugin::entropy`
  - `collectd::plugin::entropy::interval`
- `collectd::plugin::ethstat`
  - `collectd::plugin::ethstat::interfaces`
  - `collectd::plugin::ethstat::maps`
collectd::plugin::ethstat::mappedonly
collectd::plugin::ethstat::interval

collectd::plugin::exec
collectd::plugin::exec::commands
collectd::plugin::exec::commands_defaults
collectd::plugin::exec::globals
collectd::plugin::exec::interval

collectd::plugin::fhcount
collectd::plugin::fhcount::valuesabsolute
collectd::plugin::fhcount::valuespercentage
collectd::plugin::fhcount::interval

collectd::plugin::filecount
collectd::plugin::filecount::directories
collectd::plugin::filecount::interval

collectd::plugin::fscache
None

collectd-hddtemp
collectd::plugin::hddtemp::host
collectd::plugin::hddtemp::port
collectd::plugin::hddtemp::interval

collectd::plugin::hugepages
Use the hugepages plugin to collect hugepages information. This plugin is enabled by default.

Table 7.13. hugepages parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defaults</td>
<td>report_per_node_hp</td>
</tr>
<tr>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>report_root_hp</td>
<td>Boolean</td>
</tr>
<tr>
<td>true</td>
<td>values_pages</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>values_bytes</td>
<td>Boolean</td>
</tr>
<tr>
<td>false</td>
<td>values_percentage</td>
</tr>
<tr>
<td>Boolean</td>
<td>false</td>
</tr>
</tbody>
</table>

Example configuration

```
parameter_defaults:
  ExtraConfig:
    collectd::plugin::hugepages::values_percentage: true
```

Additional resources

- For more information about configuring the `hugepages` plugin, see [hugepages](#).

`collectd::plugin::intel_rdt`
`collectd::plugin::interface`

Use the `interface` plugin to measure interface traffic in octets, packets per second, and error rate per second. This plugin is enabled by default.

Table 7.14. interface parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>interfaces</td>
</tr>
<tr>
<td>Array</td>
<td>[]</td>
</tr>
<tr>
<td>ignoreselected</td>
<td>Boolean</td>
</tr>
<tr>
<td>false</td>
<td>reportinactive</td>
</tr>
<tr>
<td>Boolean</td>
<td>true</td>
</tr>
</tbody>
</table>

Example configuration

```
parameter_defaults:
  ExtraConfig:
    collectd::plugin::interface::interfaces: - lo
    collectd::plugin::interface::ignoreselected: true
```

Additional resources
For more information about configuring the `interfaces` plugin, see `interfaces`.

`collectd::plugin::ipc`

- None

`collectd::plugin::ipmi`

- `collectd::plugin::ipmi::ignore_selected`
- `collectd::plugin::ipmi::notify_sensor_add`
- `collectd::plugin::ipmi::notify_sensor_remove`
- `collectd::plugin::ipmi::notify_sensor_not_present`
- `collectd::plugin::ipmi::sensors`
- `collectd::plugin::ipmi::interval`

`collectd::plugin::iptables`

`collectd::plugin::irq`

- `collectd::plugin::irq::irqs`
- `collectd::plugin::irq::ignoreselected`
- `collectd::plugin::irq::interval`

`collectd::plugin::load`

Use the `load` plugin to collect the system load and to get overview on system use. This plugin is enabled by default.

Table 7.15. plugin parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>report_relative</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

Example configuration

```
parameter_defaults:
    ExtraConfig:
        collectd::plugin::load::report_relative: false
```

Additional resources

- For more information about configuring the `load` plugin, see `load`.

`collectd::plugin::logfile`

- `collectd::plugin::logfile::log_level`
- `collectd::plugin::logfile::log_file`
Use the `mcelog` plugin to send notifications and statistics relevant to Machine Check Exceptions when they occur. Configure `mcelog` to run in daemon mode and ensure that logging capabilities are enabled.

Table 7.16. `mcelog` parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mcelogfile</td>
<td>String</td>
</tr>
<tr>
<td>Memory</td>
<td>Hash { mcelogclientsocket[string], persistentnotification[boolean] }</td>
</tr>
</tbody>
</table>

Example configuration

```yaml
parameter_defaults:
  CollectdExtraPlugins: mcelog
  CollectdEnableMcelog: true
```

Additional resources

- For more information about configuring the `mcelog` plugin, see `mcelog`.

The `memory` plugin provides information about the memory of the system. This plugin is enabled by default.

Table 7.17. `memory` parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>valuesabsolute</td>
<td>Boolean</td>
</tr>
<tr>
<td>valuespercentage</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

**Example configuration**

```yaml
parameter_defaults:
  ExtraConfig:
    collectd::plugin::memory::valuesabsolute: true
    collectd::plugin::memory::valuespercentage: false
```

**Additional resources**

- For more information about configuring the `memory` plugin, see `memory`.

---

**collectd::plugin::multimeter**

**collectd::plugin::mysql**

- `collectd::plugin::mysql::interval`

**collectd::plugin::netlink**

- `collectd::plugin::netlink::interfaces`
- `collectd::plugin::netlink::verboseinterfaces`
- `collectd::plugin::netlink::qdiscs`
- `collectd::plugin::netlink::classes`
- `collectd::plugin::netlink::filters`
- `collectd::plugin::netlink::ignoreselected`
- `collectd::plugin::netlink::interval`

**collectd::plugin::network**

- `collectd::plugin::network::timetolive`
- `collectd::plugin::network::maxpacketsize`
- `collectd::plugin::network::forward`
- `collectd::plugin::network::reportstats`
- `collectd::plugin::network::listeners`
- `collectd::plugin::network::servers`
- `collectd::plugin::network::interval`
collectd::plugin::nfs
  • collectd::plugin::nfs::interval

collectd::plugin::notify_nagios

collectd::plugin::ntpd
  • collectd::plugin::ntpd::host
  • collectd::plugin::ntpd::port
  • collectd::plugin::ntpd::reverselookups
  • collectd::plugin::ntpd::includeunitid
  • collectd::plugin::ntpd::interval

collectd::plugin::numa
  • None

collectd::plugin::olsrd

collectd::plugin::openldap

collectd::plugin::openvpn
  • collectd::plugin::openvpn::statusfile
  • collectd::plugin::openvpn::improvednamingschema
  • collectd::plugin::openvpn::collectcompression
  • collectd::plugin::openvpn::collectindividualusers
  • collectd::plugin::openvpn::collectusercount
  • collectd::plugin::openvpn::interval

collectd::plugin::ovs_stats
Use the ovs_stats plugin to collect statistics of OVS-connected interfaces. The ovs_stats plugin uses the OVSDB management protocol (RFC7047) monitor mechanism to get statistics from OVSDB.

Table 7.18. ovs_stats parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>String</td>
</tr>
<tr>
<td>bridges</td>
<td>List</td>
</tr>
<tr>
<td>port</td>
<td>Integer</td>
</tr>
<tr>
<td>socket</td>
<td>String</td>
</tr>
</tbody>
</table>
Example configuration

The following example shows how to enable the `ovs_stats` plugin. If you deploy your overcloud with OVS, you do not need to enable the `ovs_stats` plugin.

```yaml
parameter_defaults:
  CollectdExtraPlugins:
    - ovs_stats
ExtraConfig:
  collectd::plugin::ovs_stats::socket: '/run/openvswitch/db.sock'
```

Additional resources

- For more information about configuring the `ovs_stats` plugin, see `ovs_stats`.

`collectd::plugin::pcie_errors`

Use the `pcie_errors` plugin to poll PCI config space for baseline and Advanced Error Reporting (AER) errors, and to parse syslog for AER events. Errors are reported through notifications.

**Table 7.19. pcie_errors parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Enum (sysfs, proc)</td>
</tr>
<tr>
<td>access</td>
<td>String</td>
</tr>
<tr>
<td>reportmasked</td>
<td>Boolean</td>
</tr>
<tr>
<td>persistent_notifications</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

Example configuration

```yaml
parameter_defaults:
  CollectdExtraPlugins:
    - pcie_errors
```

Additional resources

- For more information about configuring the `pcie_errors` plugin, see `pcie_errors`.

`collectd::plugin::ping`

- `collectd::plugin::ping::hosts`
- `collectd::plugin::ping::timeout`
- `collectd::plugin::ping::ttl`
- `collectd::plugin::ping::source_address`
- `collectd::plugin::ping::device`
The `collectd::plugin::processes` plugin provides information about processes of the system. This plugin is enabled by default.

Table 7.20. plugin parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>processes</td>
<td>Array</td>
</tr>
<tr>
<td>process_matches</td>
<td>Array</td>
</tr>
<tr>
<td>collect_context_switch</td>
<td>Boolean</td>
</tr>
<tr>
<td>collect_file_descriptor</td>
<td>Boolean</td>
</tr>
<tr>
<td>collect_memory_maps</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

Additional resources

- For more information about configuring the `processes` plugin, see `processes`.  

```sql

```
collectd::plugin::protocols
  • collectd::plugin::protocols::ignoreselected
  • collectd::plugin::protocols::values

collectd::plugin::python
collectd::plugin::sensors
collectd::plugin::serial
collectd::plugin::smart
  • collectd::plugin::smart::disks
  • collectd::plugin::smart::ignoreselected
  • collectd::plugin::smart::interval

collectd::plugin::snmp
collectd::plugin::snmp_agent
Use the snmp_agent plugin as an SNMP subagent to map collectd metrics to relevant OIDs. The snmp agent also requires a running snmpd service.

Example configuration:

```
parameter_defaults:
  CollectdExtraPlugins:
    snmp_agent
resource_registry:
  OS::TripleO::Services::Snmp: /usr/share/openstack-tripleo-heat-templates/deployment/snmp/snmp-baremetal-puppet.yaml
```

Additional resources:
For more information about how to configure snmp_agent, see snmp_agent.

collectd::plugin::statsd
  • collectd::plugin::statsd::host
  • collectd::plugin::statsd::port
  • collectd::plugin::statsd::deletecounters
  • collectd::plugin::statsd::deletetimers
  • collectd::plugin::statsd::delegauges
  • collectd::plugin::statsd::deletesets
  • collectd::plugin::statsd::countersum
  • collectd::plugin::statsd::timerpercentile
  • collectd::plugin::statsd::timelower
  • collectd::plugin::statsd::timerupper
collectd::plugin::statsd::timersum
collectd::plugin::statsd::timercount
collectd::plugin::statsd::interval

collectd::plugin::swap

collectd::plugin::swap::reportbydevice
collectd::plugin::swap::reportbytes
collectd::plugin::swap::valuesabsolute
collectd::plugin::swap::valuespercentage
collectd::plugin::swap::reportio
collectd::plugin::swap::interval

collectd::plugin::sysevent
collectd::plugin::syslog

collectd::plugin::syslog::log_level
collectd::plugin::syslog::notify_level
collectd::plugin::syslog::interval

collectd::plugin::table

collectd::plugin::table::tables
collectd::plugin::table::interval

collectd::plugin::tail

collectd::plugin::tail::files
collectd::plugin::tail::interval

collectd::plugin::tail_csv

collectd::plugin::tail_csv::metrics
collectd::plugin::tail_csv::files

collectd::plugin::target_notification
collectd::plugin::target_replace
collectd::plugin::target_scale
collectd::plugin::target_set
collectd::plugin::target_v5upgrade
collectd::plugin::tcpconns

collectd::plugin::tcpconns::localports
collectd::plugin::tcpconns::remoteports
• collectd::plugin::tcpconns::listening
• collectd::plugin::tcpconns::allportssummary
• collectd::plugin::tcpconns::interval

collectd::plugin::ted
collectd::plugin::thermal
• collectd::plugin::thermal::devices
• collectd::plugin::thermal::ignoreselected
• collectd::plugin::thermal::interval

collectd::plugin::threshold
• collectd::plugin::threshold::types
• collectd::plugin::threshold::plugins
• collectd::plugin::threshold::hosts
• collectd::plugin::threshold::interval

collectd::plugin::turbostat
• collectd::plugin::turbostat::core_c_states
• collectd::plugin::turbostat::package_c_states
• collectd::plugin::turbostat::system_management_interrupt
• collectd::plugin::turbostat::digital_temperature_sensor
• collectd::plugin::turbostat::tcc_activation_temp
• collectd::plugin::turbostat::running_average_power_limit
• collectd::plugin::turbostat::logical_core_names

collectd::plugin::unixsock
collectd::plugin::uptime
• collectd::plugin::uptime::interval

collectd::plugin::users
• collectd::plugin::users::interval

collectd::plugin::uuid
• collectd::plugin::uuid::uuid_file
• collectd::plugin::uuid::interval

collectd::plugin::virt

Use the virt plugin to collect CPU, disk, network load, and other metrics through the libvirt API for Red Hat OpenStack Platform 16.1 Service Telemetry Framework 1.2.
Use the **virt** plugin to collect CPU, disk, network load, and other metrics through the **libvirt** API for virtual machines on the host.

Table 7.21. virt parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection</td>
<td>String</td>
</tr>
<tr>
<td>refresh_interval</td>
<td>Hash</td>
</tr>
<tr>
<td>domain</td>
<td>String</td>
</tr>
<tr>
<td>block_device</td>
<td>String</td>
</tr>
<tr>
<td>interface_device</td>
<td>String</td>
</tr>
<tr>
<td>ignore_selected</td>
<td>Boolean</td>
</tr>
<tr>
<td>plugin_instance_format</td>
<td>String</td>
</tr>
<tr>
<td>hostname_format</td>
<td>String</td>
</tr>
<tr>
<td>interface_format</td>
<td>String</td>
</tr>
<tr>
<td>extra_stats</td>
<td>String</td>
</tr>
</tbody>
</table>

**Example configuration**

```
ExtraConfig:
  collectd::plugin::virt::plugin_instance_format: name
```

**Additional resources**

For more information about configuring the **virt** plugin, see **virt**.

**collectd::plugin::vmem**
- collectd::plugin::vmem::verbose
- collectd::plugin::vmem::interval

**collectd::plugin::vserver**

**collectd::plugin::wireless**

**collectd::plugin::write_graphite**
- collectd::plugin::write_graphite::carbons
- collectd::plugin::write_graphite::carbon_defaults
- collectd::plugin::write_graphite::globals
**collectd::plugin::write_http**

Use the write_http output plugin to submit values to an HTTP server by using POST requests and encoding metrics with JSON, or by using the PUTVAL command.

Table 7.22. write_http parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ensure</td>
<td>Enum[present, absent]</td>
</tr>
<tr>
<td>nodes</td>
<td>Hash[String, Hash[String, Scalar]]</td>
</tr>
<tr>
<td>urls</td>
<td>Hash[String, Hash[String, Scalar]]</td>
</tr>
<tr>
<td>manage_package</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

**Example configuration**

```yaml
parameter_defaults:
  CollectdExtraPlugins:
    - write_http
ExtraConfig:
  collectd::plugin::write_http::nodes:
    collectd:
      url: "http://collectd.tld.org/collectd"
      metrics: true
      header: "X-Custom-Header: custom_value"
```

**Additional resources**

- For more information about configuring the write_http plugin, see write_http.

**collectd::plugin::write_kafka**

Use the write_kafka plugin to send values to a Kafka topic. Configure the write_kafka plugin with one or more topic blocks. For each topic block, you must specify a unique name and one Kafka producer. You can use the following per-topic parameters inside the topic block:

Table 7.23. write_kafka parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>kafka_hosts</td>
<td>Array[String]</td>
</tr>
<tr>
<td>kafka_port</td>
<td>Integer</td>
</tr>
<tr>
<td>topics</td>
<td>Hash</td>
</tr>
<tr>
<td>properties</td>
<td>Hash</td>
</tr>
<tr>
<td>meta</td>
<td>Hash</td>
</tr>
</tbody>
</table>
Example configuration:

```
parameter_defaults:
  CollectdExtraPlugins:
    - write_kafka
ExtraConfig:
  collectd::plugin::write_kafka::kafka_hosts:
    - nodeA
    - nodeB
  collectd::plugin::write_kafka::topics:
    some_events:
      format: JSON
```

Additional resources:

For more information about how to configure the `write_kafka` plugin, see `write_kafka`.

```
collectd::plugin::write_log
  - collectd::plugin::write_log::format

collectd::plugin::zfs_arc
  - None
```