



# Red Hat OpenStack Platform 16.1

## Service Telemetry Framework 1.1

Installing and deploying Service Telemetry Framework 1.1



# Red Hat OpenStack Platform 16.1 Service Telemetry Framework 1.1

---

Installing and deploying Service Telemetry Framework 1.1

OpenStack Team  
rhos-docs@redhat.com

## Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

This guide contains information about installing the core components and deploying Service Telemetry Framework 1.1.

## Table of Contents

<b>CHAPTER 1. INTRODUCTION TO SERVICE TELEMTRY FRAMEWORK</b> .....	<b>4</b>
1.1. SERVICE TELEMTRY FRAMEWORK ARCHITECTURE	4
1.2. INSTALLATION SIZE	6
<b>CHAPTER 2. INSTALLING THE CORE COMPONENTS OF SERVICE TELEMTRY FRAMEWORK</b> .....	<b>8</b>
2.1. THE CORE COMPONENTS OF STF	8
2.2. PREPARING YOUR OCP ENVIRONMENT FOR STF	8
2.2.1. Persistent volumes	9
2.2.1.1. Using ephemeral storage	9
2.2.2. Resource allocation	9
2.2.3. Node tuning operator	9
2.3. DEPLOYING STF TO THE OCP ENVIRONMENT	10
2.3.1. Deploying STF to the OCP environment with Elasticsearch	10
2.3.2. Deploying STF to the OCP environment without Elasticsearch	11
2.3.3. Creating a namespace	11
2.3.4. Creating an OperatorGroup	11
2.3.5. Enabling the OperatorHub.io Community Catalog Source	12
2.3.6. Enabling Red Hat STF Catalog Source	12
2.3.7. Subscribing to the AMQ Certificate Manager Operator	13
2.3.8. Subscribing to the Elastic Cloud on Kubernetes Operator	14
2.3.9. Subscribing to the Service Telemetry Operator	14
2.3.10. Overview of the ServiceTelemetry object	15
2.3.10.1. backends	16
2.3.10.1.1. Enabling Prometheus as a storage backend for metrics	16
2.3.10.1.2. Enabling Elasticsearch as a storage backend for events	16
2.3.10.2. clouds	16
2.3.10.3. alerting	17
2.3.10.4. graphing	17
2.3.10.5. highAvailability	17
2.3.10.6. transports	17
2.3.11. Creating a ServiceTelemetry object in OCP	18
2.4. REMOVING STF FROM THE OCP ENVIRONMENT	20
2.4.1. Deleting the namespace	20
2.4.2. Removing the CatalogSource	20
<b>CHAPTER 3. COMPLETING THE SERVICE TELEMTRY FRAMEWORK CONFIGURATION</b> .....	<b>22</b>
3.1. CONNECTING RED HAT OPENSTACK PLATFORM TO SERVICE TELEMTRY FRAMEWORK	22
3.2. DEPLOYING TO NON-STANDARD NETWORK TOPOLOGIES	22
3.3. CONFIGURING RED HAT OPENSTACK PLATFORM OVERCLOUD FOR SERVICE TELEMTRY FRAMEWORK	23
3.3.1. Retrieving the AMQ Interconnect route address	23
3.3.2. Configuring the STF connection for the overcloud	24
3.3.3. Validating client-side installation	25
<b>CHAPTER 4. ADVANCED FEATURES</b> .....	<b>30</b>
4.1. CUSTOMIZING THE DEPLOYMENT	30
4.1.1. Manifest override parameters	30
4.1.2. Overriding a managed manifest	31
4.2. ALERTS	33
4.2.1. Creating an alert rule in Prometheus	33
4.2.2. Configuring custom alerts	34
4.2.3. Creating an alert route in Alertmanager	35

4.3. CONFIGURING SNMP TRAPS	37
4.4. HIGH AVAILABILITY	38
4.4.1. Configuring high availability	38
4.5. DASHBOARDS	39
4.5.1. Setting up Grafana to host the dashboard	39
4.5.2. Importing dashboards	40
4.5.3. Viewing and editing queries	41
4.5.4. The Grafana infrastructure dashboard	42
4.5.4.1. Top panels	42
4.5.4.2. Networking panels	42
4.5.4.3. CPU panels	43
4.5.4.4. Memory panels	43
4.5.4.5. Disk/file system	44
4.6. MULTIPLE CLOUD CONFIGURATION	45
4.6.1. Planning AMQP address prefixes	46
4.6.2. Deploying Smart Gateways	47
4.6.3. Deleting the default Smart Gateways	48
4.6.4. Creating the OpenStack environment file	49
4.6.5. Querying metrics data from multiple clouds	51
4.7. EPHEMERAL STORAGE	52
4.7.1. Configuring ephemeral storage	52
4.8. MONITORING THE RESOURCE USAGE OF RED HAT OPENSTACK PLATFORM SERVICES	53
<b>APPENDIX A. COLLECTD PLUG-INS</b> .....	<b>54</b>



# CHAPTER 1. INTRODUCTION TO SERVICE TELEMETRY FRAMEWORK

Service Telemetry Framework (STF) receives monitoring data from Red Hat OpenStack Platform or third-party nodes for storage, dashboarding, 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. SERVICE TELEMETRY FRAMEWORK ARCHITECTURE

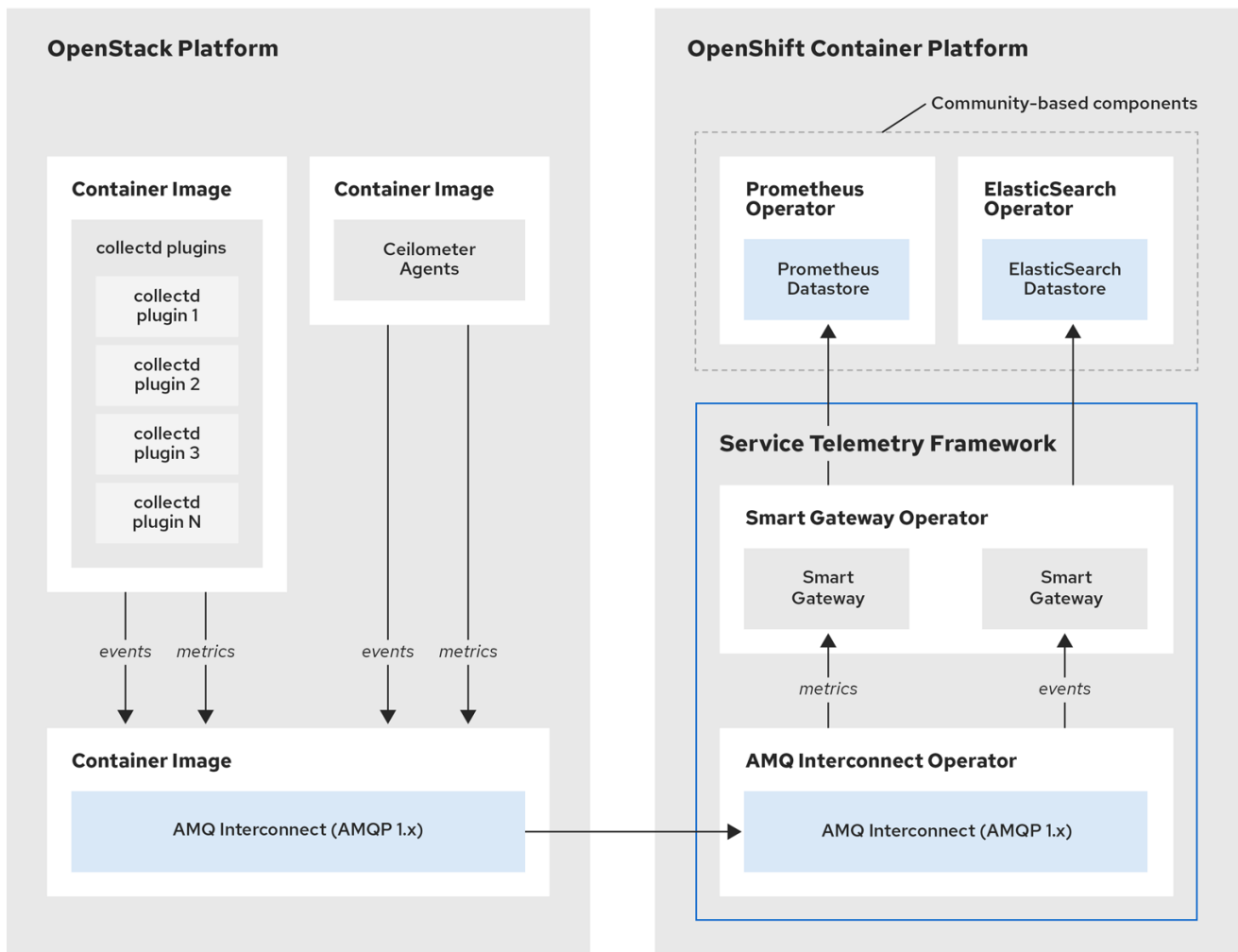
Service Telemetry Framework (STF) uses the components described in [Table 1.1, "STF components"](#):

**Table 1.1. STF components**

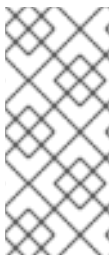
Client	Component	Server (OCP)
yes	An AMQP 1.x compatible messaging bus to shuttle the metrics to STF for storage in Prometheus	yes
no	Smart Gateway to pick metrics and events from the AMQP 1.x bus and to deliver events to Elasticsearch or to provide metrics to Prometheus	yes
no	Prometheus as time-series data storage	yes
no	ElasticSearch as events data storage	yes
yes	collectd to collect infrastructure metrics and events	no
yes	Ceilometer to collect Red Hat OpenStack Platform metrics and events	no



Figure 1.1. Service Telemetry Framework architecture overview



65\_OpenStack\_0620

**NOTE**

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.

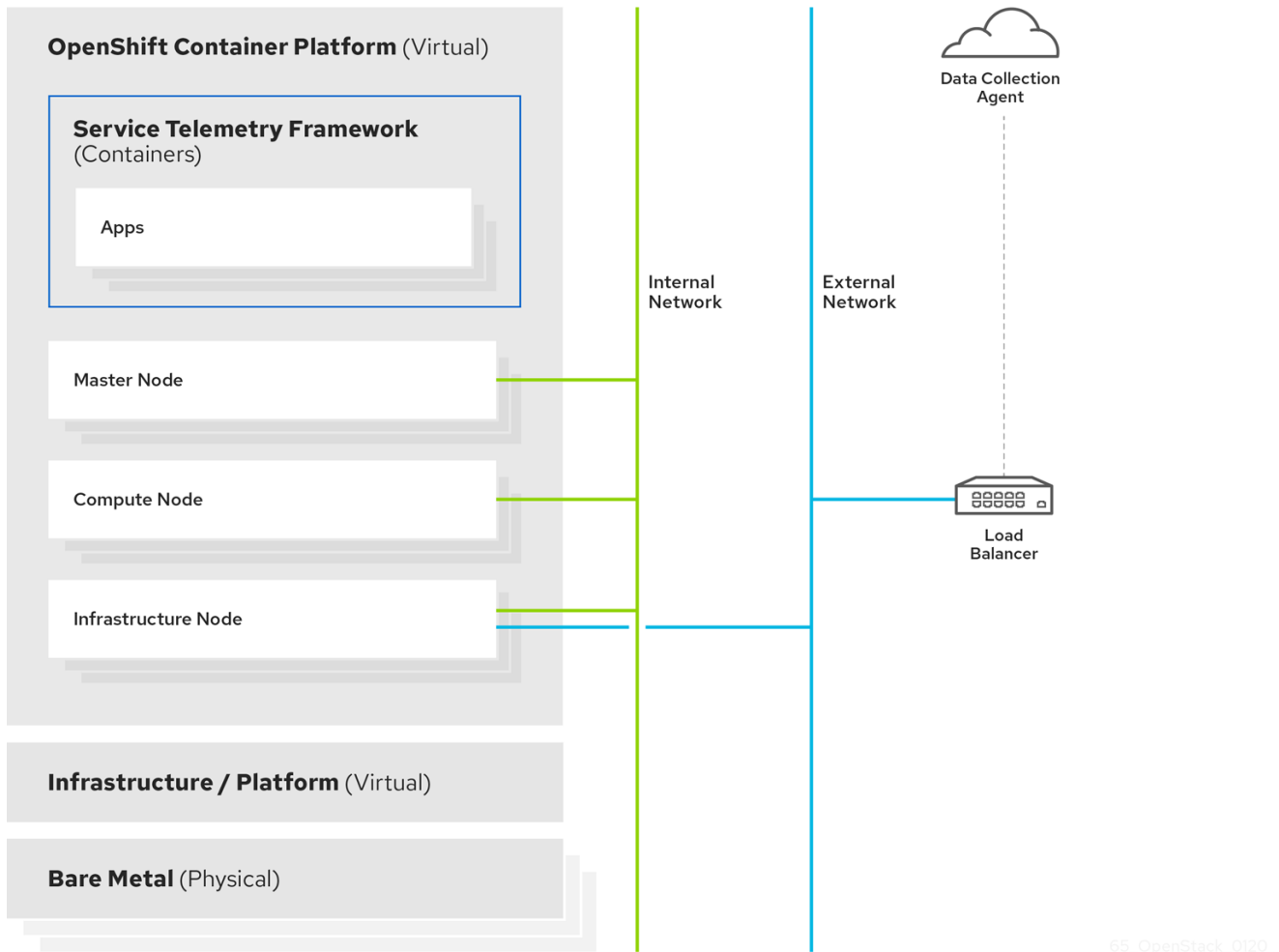
If you plan to collect and store events, collectd or Ceilometer delivers event data to the server side by using the AMQ Interconnect transport, delivering the data through the message bus. Another Smart Gateway writes the data to the ElasticSearch datastore.

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

- Service Telemetry Framework 1.1 (STF)

- Red Hat OpenShift Container Platform (OCP)
- Infrastructure platform

Figure 1.2. Server-side STF monitoring infrastructure



65\_OpenStack\_0120

For more information about how to deploy Red Hat OpenShift Container Platform, see the [OCP product documentation](#). 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>.

**NOTE**

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

## 1.2. INSTALLATION SIZE

The size of your Red Hat OpenShift Container Platform 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. INSTALLING THE CORE COMPONENTS OF SERVICE TELEMETRY FRAMEWORK

Before you install Service Telemetry Framework (STF), ensure that Red Hat OpenShift Container Platform (OCP) version 4.5 is running and that you understand the core components of the framework. As part of the OCP installation planning process, ensure that the administrator provides persistent storage and enough resources to run the STF components on top of the OCP environment.



### IMPORTANT

Red Hat OpenShift Container Platform version 4.5 is currently required for a successful installation of STF. To upgrade to later versions of STF, you must migrate installations of STF 1.0 that use an **OperatorSource** to **CatalogSource**. For more information about migrating, see [Migrating Service Telemetry Framework v1.0 from OperatorSource to CatalogSource](#).

### 2.1. THE CORE COMPONENTS OF STF

The following STF core components are managed by Operators:

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

Each component has a corresponding Operator that you can use to load the various application components and objects.

#### Additional resources

For more information about Operators, see the [Understanding Operators](#) guide.

### 2.2. PREPARING YOUR OCP ENVIRONMENT FOR STF

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 2.2.1, "Persistent volumes"](#).
- Ensure that enough resources are available to run the Operators and the application containers. For more information, see [Section 2.2.2, "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 2.3, "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 2.2.3, “Node tuning operator”](#).

## 2.2.1. 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**. For more information about recommended configurable storage technology in Red Hat OpenShift Container Platform, see [Recommended configurable storage technology](#).

### Additional resources

For more information about configuring persistent storage for OCP, see [Understanding persistent storage](#).

### 2.2.1.1. Using ephemeral storage



#### WARNING

You can use ephemeral storage with STF. However, 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.

### Additional resources

For more information about enabling ephemeral storage for STF, see [Section 4.7.1, “Configuring ephemeral storage”](#).

## 2.2.2. 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 <https://access.redhat.com/articles/4907241>.
- For information about sizing requirements for Elasticsearch, see <https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-managing-compute-resources.html>.

## 2.2.3. 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](#).

## 2.3. 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 2.3.1, “Deploying STF to the OCP environment with ElasticSearch”](#).
- Deploy STF without ElasticSearch and disable events support. For more information, see [Section 2.3.2, “Deploying STF to the OCP environment without ElasticSearch”](#).

### 2.3.1. Deploying STF to the OCP environment with ElasticSearch

Complete the following tasks:

1. [Section 2.3.3, "Creating a namespace"](#) .
2. [Section 2.3.4, "Creating an OperatorGroup"](#) .
3. [Section 2.3.5, "Enabling the OperatorHub.io Community Catalog Source"](#) .
4. [Section 2.3.6, "Enabling Red Hat STF Catalog Source"](#) .
5. [Section 2.3.7, "Subscribing to the AMQ Certificate Manager Operator"](#) .
6. [Section 2.3.8, "Subscribing to the Elastic Cloud on Kubernetes Operator"](#) .
7. [Section 2.3.9, "Subscribing to the Service Telemetry Operator"](#) .
8. [Section 2.3.11, "Creating a ServiceTelemetry object in OCP"](#) .

## 2.3.2. Deploying STF to the OCP environment without Elasticsearch

Complete the following tasks:

1. [Section 2.3.3, "Creating a namespace"](#) .
2. [Section 2.3.4, "Creating an OperatorGroup"](#) .
3. [Section 2.3.6, "Enabling Red Hat STF Catalog Source"](#) .
4. [Section 2.3.7, "Subscribing to the AMQ Certificate Manager Operator"](#) .
5. [Section 2.3.9, "Subscribing to the Service Telemetry Operator"](#) .
6. [Section 2.3.11, "Creating a ServiceTelemetry object in OCP"](#) .

## 2.3.3. 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:

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

## 2.3.4. Creating an OperatorGroup

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

### Procedure

- Enter the following command:

```
$ 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](#).

## 2.3.5. 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/operator-framework/upstream-community-operators:latest
  displayName: OperatorHub.io Operators
  publisher: OperatorHub.io
EOF

```

## 2.3.6. Enabling Red Hat STF Catalog Source

Before you deploy STF on Red Hat OpenShift Container Platform, you must enable the catalog source.

### Procedure

1. Install a CatalogSource that contains the Service Telemetry Operator and the Smart Gateway Operator:

```

$ 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:latest
  publisher: Red Hat

```



```
sourceType: grpc
updateStrategy:
  registryPoll:
    interval: 30m
EOF
```

- To validate the creation of your CatalogSource, use the **oc get catalogsources** command:

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

NAME	DISPLAY	TYPE	PUBLISHER	AGE
redhat-operators-stf	Red Hat STF Operators	grpc	Red Hat	62m

- To validate that the Operators are available from the catalog, use the **oc get packagemanifest** command:

```
$ oc get packagemanifests | grep "Red Hat STF"
```

smart-gateway-operator	Red Hat STF Operators	63m
service-telemetry-operator	Red Hat STF Operators	63m

### 2.3.7. 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.

#### Procedure

- Subscribe to the AMQ Certificate Manager Operator, create the subscription, and validate the AMQ7 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.

```
$ 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
sourceNamespace: openshift-marketplace
EOF
```

- To validate your **ClusterServiceVersion**, use the **oc get csv** command:

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

NAME                                DISPLAY                                VERSION  REPLACES  PHASE
amq7-cert-manager.v1.0.0  Red Hat Integration - AMQ Certificate Manager  1.0.0
Succeeded
```

Ensure that `amq7-cert-manager.v1.0.0` has a phase **Succeeded**.

### 2.3.8. 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

- Apply the following manifest to your OCP environment to enable the Elastic Cloud on Kubernetes Operator:

```
$ 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
```

- To verify that the **ClusterServiceVersion** for ElasticSearch Cloud on Kubernetes **succeeded**, enter the **oc get csv** command:

```
$ oc get csv

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

### 2.3.9. Subscribing to the Service Telemetry Operator

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

#### Procedure

- 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
  installPlanApproval: Automatic
  name: service-telemetry-operator
  source: redhat-operators-stf
  sourceNamespace: openshift-marketplace
EOF
```

- To validate the Service Telemetry Operator and the dependent operators, enter the following command:

```
$ oc get csv --namespace service-telemetry
NAME                                DISPLAY                                VERSION  REPLACES
PHASE
amq7-cert-manager.v1.0.0            Red Hat Integration - AMQ Certificate Manager  1.0.0
Succeeded
amq7-interconnect-operator.v1.2.1    Red Hat Integration - AMQ Interconnect        1.2.1
Succeeded
elastic-cloud-eck.v1.2.1            Elastic Cloud on Kubernetes                  1.2.1
Succeeded
prometheusoperator.0.37.0           Prometheus Operator                          0.37.0
Succeeded
service-telemetry-operator.v1.1.0    Service Telemetry Operator                  1.1.0
Succeeded
smart-gateway-operator.v2.1.0       Smart Gateway Operator                      2.1.0
Succeeded
```

### 2.3.10. Overview of the ServiceTelemetry object



#### IMPORTANT

Versions of Service Telemetry Operator prior to v1.1.0 used a flat API (**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**) 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.

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.

### 2.3.10.1. 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 2.3.10.2, “clouds”](#).

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

#### 2.3.10.1.1. 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
```

#### 2.3.10.1.2. 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 2.3.8, “Subscribing to the Elastic Cloud on Kubernetes Operator”](#).

By default, ElasticSearch storage of events is disabled. For more information, see [Section 2.3.2, “Deploying STF to the OCP environment without ElasticSearch”](#).

### 2.3.10.2. 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.

### 2.3.10.3. 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 4.2, "Alerts"](#).

### 2.3.10.4. graphing

Use the **graphing** parameter to control the creation of a [Grafana](#) instance. By default, **graphing** is disabled. For more information, see [Section 4.5, "Dashboards"](#).

### 2.3.10.5. 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 4.4, "High availability"](#).

### 2.3.10.6. 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.

### 2.3.11. 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 2.3.10, "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
          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
highAvailability:
  enabled: false
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

```

- 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

```

- 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

```

- 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.

```
$ oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
alertmanager-default-0	2/2	Running	0	38s
default-cloud1-ceil-meter-smartgateway-58d8876857-lbf9d	1/1	Running	0	159m
default-cloud1-coll-meter-smartgateway-8645d64f5f-rxfpb	2/2	Running	0	159m
default-interconnect-79d9994b5-xnfvv	1/1	Running	0	167m
elastic-operator-746f86c956-jkvcq	1/1	Running	0	6h23m
interconnect-operator-5b474bddd-csztsj	1/1	Running	0	6h19m
prometheus-default-0	3/3	Running	1	5m39s
prometheus-operator-7dfb478c8b-bfd4j	1/1	Running	0	6h19m
service-telemetry-operator-656fc8ccb6-4w8x4	2/2	Running	0	98m
smart-gateway-operator-7f49676d5d-nqzmp	2/2	Running	0	6h21m

## 2.4. 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 2.4.1, "Deleting the namespace"](#).
2. [Section 2.4.2, "Removing the CatalogSource"](#).

### 2.4.1. 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.
```

### 2.4.2. 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. Delete the CatalogSource:

```
$ oc delete --namespace=openshift-marketplace catalogsource redhat-operators-stf  
catalogsource.operators.coreos.com "redhat-operators-stf" deleted
```

2. Verify that the STF PackageManifests are removed from the platform. If successful, the following command returns no result:

```
$ oc get packagemanifests | grep "Red Hat STF"
```

3. 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 2.3, “Deploying STF to the OCP environment”](#).

## CHAPTER 3. COMPLETING THE SERVICE TELEMETRY FRAMEWORK CONFIGURATION

### 3.1. CONNECTING RED HAT OPENSTACK PLATFORM TO SERVICE TELEMETRY FRAMEWORK

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 3.2, “Deploying to non-standard network topologies”](#).

### 3.2. 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.

```
CephStorageExtraConfig:
  tripleo::profile::base::metrics::collectd::amqp_host: "%{hiera('storage')}}"
  tripleo::profile::base::metrics::qdr::listener_addr: "%{hiera('storage')}}"
  tripleo::profile::base::ceilometer::agent::notification::notifier_host_addr: "%{hiera('storage')}}"
```

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')}}"
```

### 3.3. 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 3.3.1, "Retrieving the AMQ Interconnect route address"](#)
2. [Section 3.3.2, "Configuring the STF connection for the overcloud"](#)
3. [Section 3.3.3, "Validating client-side installation"](#)

#### Additional resources

- To collect data through AMQ Interconnect, see [The amqp1 plug-in](#) in the *Monitoring Tools Configuration* guide.

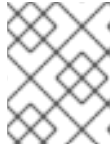
#### 3.3.1. Retrieving the AMQ Interconnect route address

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:

```
$ oc get routes -ogo-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.

## 3.3.2. 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 4.6, "Multiple cloud configuration"](#).

Additionally, setting `EventPipelinePublishers` and `MetricPipelinePublishers` 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 **MetricPipelinePublishers: []** 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 3.3.1, “Retrieving the AMQ Interconnect route address”](#)

```
parameter_defaults:
  EventPipelinePublishers: []
  MetricPipelinePublishers: []
  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.

### 3.3.3. 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
  id host                               container
  role dir security                     authentication tenant
=====
=====
=====
=====
  1 default-interconnect-5671-service-telemetry.apps.infra.watch:443 default-
interconnect-7458fd4d69-bgzfb                               edge out
  TLSv1.2(DHE-RSA-AES256-GCM-SHA384) anonymous-user
  12 172.17.1.44:60290
openstack.org/om/container/controller-0/ceilometer-agent-
notification/25/5c02cee550f143ec9ea030db5cccba14 normal in no-security
no-auth
  16 172.17.1.44:36408                               metrics
normal in no-security anonymous-user
  899 172.17.1.44:39500                               10a2e99d-1b8a-4329-b48c-
4335e5f75c84 normal in no-security
no-auth
```

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 unsett deliv
presett psdrop 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
 endpoint in 1    7          250 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
 endpoint in 1    9          250 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 0 911 0
 endpoint in 1   11         250 0 0 0 0 911 0 0
0 0 0 0 0
 endpoint out 12  32    local temp.ISY6Mccol4J2Kp 250 0 0 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
 endpoint in 912 1834    mobile $management 0 250 0 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 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:

```
$ 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-r19bd]	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		anonymous-user		000:00:00:03	000:22:08:43
23	10.128.0.1:51950	Router.compute-0.redhat.local	edge	in		anonymous-user		000:00:00:03	000:22:08:43
24	10.128.0.1:52082	Router.controller-0.redhat.local	edge	in		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.

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

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

```
2020-04-21 18:20:10.293258 UTC
default-interconnect-7458fd4d69-bgzfb
```

Router Addresses

class	addr	phs	distrib	pri	local	remote	in	out	thru
mobile	anycast/ceilometer/event.sample	0	balanced	-	1	0	970	970	0
mobile	anycast/ceilometer/metering.sample	0	balanced	-	1	0	2,344,833	2,344,833	0



<b>mobile collectd/notify</b>	<b>0</b>	<b>multicast -</b>	<b>1</b>	<b>0</b>	<b>70</b>	<b>70</b>	<b>0</b>	<b>0</b>
<b>mobile collectd/telemetry</b>	<b>0</b>	<b>multicast -</b>	<b>1</b>	<b>0</b>	<b>216,128,890</b>			
<b>216,128,890</b>	<b>0</b>							

## CHAPTER 4. ADVANCED FEATURES

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

- [Section 4.1, “Customizing the deployment”](#)
- [Section 4.2, “Alerts”](#)
- [Section 4.3, “Configuring SNMP Traps”](#)
- [Section 4.4, “High availability”](#)
- [Section 4.5, “Dashboards”](#)
- [Section 4.6, “Multiple cloud configuration”](#)
- [Section 4.7, “Ephemeral storage”](#)
- [Section 4.8, “Monitoring the resource usage of Red Hat OpenStack Platform services”](#)

### 4.1. 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 2.3.11, “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 4.1.1, “Manifest override parameters”](#).

#### 4.1.1. Manifest override parameters

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

Table 4.1. Manifest override parameters

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

#### 4.1.2. Overriding a managed manifest

Edit the **ServiceTelemetry** object and provide a parameter and manifest. For a list of available manifest override parameters, see [Section 4.1, “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**

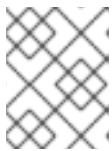
1. Log in to Red Hat OpenShift Container Platform.
2. Change to the `service-telemetry` namespace:

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

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

```
$ 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.

**NOTE**

The trailing pipe (`|`) after entering the manifest override parameter indicates that the value provided is multi-line.

```
$ oc edit stf default

apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  ...
spec:
  alertmanagerConfigManifest: | 1
    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' 2
  status:
    ...
```

- 1 Manifest override parameter is defined in the **spec** of the **ServiceTelemetry** object.
  - 2 End of the manifest override content.
5. Save and close.

## 4.2. 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 4.2.1, “Creating an alert rule in Prometheus”](#).
2. Create an alert route in Alertmanager. For more information, see [Section 4.2.3, “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/infracore/service-telemetry-operator/tree/master/deploy/alerts>

### 4.2.1. 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
```

```

namespace: service-telemetry
spec:
  groups:
  - name: ./openstack.rules
    rules:
    - alert: Metric Listener down
      expr: collectd_qpuid_router_status < 1 # To change the rule, edit the value of the
      expr parameter.
EOF

```

- 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
```

- 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_qpuid_router_status \u003c 1","duration":0,"labels":
{},"annotations":{},"alerts":[],"health":"ok","type":"alerting"}],"interval":30}}}]}
```

- 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
```

- 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>

### 4.2.2. Configuring custom alerts

You can add custom alerts to the `PrometheusRule` object that you created in [Section 4.2.1, "Creating an alert rule in Prometheus"](#).

#### Procedure

- Use the `oc edit` command:

```
$ oc edit prometheusrules prometheus-alarm-rules
```

- 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/](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>

### 4.2.3. 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. For more information, see [Section 4.1.2, "Overriding a managed manifest"](#).

#### 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
  receiver: 'null'
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:\n  resolve_timeout: 10m\n  http_config: {}\n  smtp_hello: localhost\n  smtp_require_tls: true\n  pagerduty_url: https://events.pagerduty.com/v2/enqueue\n  hipchat_api_url: https://api.hipchat.com/\n  opsgenie_api_url: https://api.opsgenie.com/\n  wechat_api_url: https://qyapi.weixin.qq.com/cgi-bin/\n  victorops_api_url: https://alert.victorops.com/integrations/generic/20131114/alert/\n  route:\n    receiver: \"null\"\n    group_by:\n      - job\n    group_wait: 30s\n    group_interval: 5m\n    repeat_interval: 12h\n  receivers:\n    - name: \"null\"\n  templates: []\n",...}}
```

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

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

- 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](#).

## 4.3. 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 4.2, "Alerts"](#).

#### Prerequisites

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

#### Procedure

- To enable SNMP traps, modify the `ServiceTelemetry` object:

```
$ oc edit stf default
```

2. Set the `alerting.alertmanager.receivers.snmpTraps` parameters:

```

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.

## 4.4. 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.

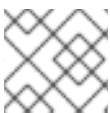


### NOTE

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

### 4.4.1. Configuring high availability

To configure STF for high availability, add **highAvailability.enabled: true** to the ServiceTelemetry object in OCP. You can 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:

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
...
spec:
...
  highAvailability:
    enabled: true
```

5. Save your changes and close the object.

## 4.5. 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 3.3, “Configuring Red Hat OpenStack Platform overcloud for Service Telemetry Framework”](#).

### 4.5.1. 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 2.3.5, “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
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:
  ...
  graphing:
    enabled: true
```

## 6. Verify that the Grafana instance deployed:

```
$ oc get pod -l app=grafana
```

NAME	READY	STATUS	RESTARTS	AGE
grafana-deployment-7fc7848b56-sbkhv	1/1	Running	0	1m

## 4.5.2. 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:

```
$ oc apply -f
https://raw.githubusercontent.com/infrawatch/dashboards/master/deploy/rhos-
dashboard.yaml
```

```
grafanadashboard.integreatly.org/rhos-dashboard created
```

2. Verify that the resources installed correctly:

```
$ oc get grafanadashboards
```

```
NAME          AGE
rhos-dashboard 7d21h
```

```
$ oc get grafanadatasources
```

```
NAME          AGE
default-ds-prometheus 20h
```

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
```

NAME	HOST/PORT	PATH	SERVICES
dashboards	dashboards-service- telemetry.apps.stfcloudops1.lab.upshift.rdu2.redhat.com	edge/Redirect	grafana-service 3000
	WILDCARD	None	

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.

### 4.5.3. 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.

## 4.5.4. The Grafana infrastructure dashboard

The infrastructure dashboard shows metrics for a single node at a time. Select a node from the upper left corner of the dashboard.

### 4.5.4.1. Top panels

Title	Unit	Description
Current Global Alerts	-	Current alerts fired by Prometheus
Recent Global Alerts	-	Recently fired alerts in 5m time steps
Status Panel	-	Node status: up, down, unavailable
Uptime	s/m/h/d/M/Y	Total operational time of node
CPU Cores	cores	Total number of cores
Memory	bytes	Total memory
Disk Size	bytes	Total storage size
Processes	processes	Total number of processes listed by type
Load Average	processes	Load average represents the average number of running and uninterruptible processes residing in the kernel execution queue.

### 4.5.4.2. Networking panels

### Panels that display the network interfaces of the node.

Panel	Unit	Description
Physical Interfaces Ingress Errors	errors	Total errors with incoming data
Physical Interfaces Egress Errors	errors	Total errors with outgoing data
Physical Interfaces Ingress Error Rates	errors/s	Rate of incoming data errors
Physical Interfaces egress Error Rates	errors/s	Rate of outgoing data errors
Physical Interfaces Packets Ingress pps Incoming packets per second	Physical Interfaces Packets Egress	pps
Outgoing packets per second	Physical Interfaces Data Ingress	bytes/s
Incoming data rates	Physical Interfaces Data Egress	bytes/s
Outgoing data rates	Physical Interfaces Drop Rate Ingress	pps
Incoming packets drop rate	Physical Interfaces Drop Rate Egress	pps

### 4.5.4.3. CPU panels

#### Panels that display CPU usage of the node.

Panel	Unit	Description
Current CPU Usage	percent	Instantaneous usage at the time of the last query.
Aggregate CPU Usage	percent	Average non-idle CPU activity of all cores on a node.
Aggr. CPU Usage by Type	percent	Shows time spent for each type of thread averaged across all cores.

### 4.5.4.4. Memory panels

#### Panels that display memory usage on the node.

Panel	Unit	Description
Memory Used	percent	Amount of memory being used at time of last query.
Huge Pages Used	hugepages	Number of hugepages being used.  Memory

#### 4.5.4.5. Disk/file system

Panels that display space used on disk.

Panel	Unit	Description	Notes
Disk Space Usage	percent	Total disk use at time of last query.	
Inode Usage	percent	Total inode use at time of last query.	
Aggregate Disk Space Usage	bytes	Total disk space used and reserved.	Because this query relies on the <b>df</b> plugin, temporary file systems that do not necessarily use disk space are included in the results. The query tries to filter out most of these, but it might not be exhaustive.
Disk Traffic	bytes/s	Shows rates for both reading and writing.	
Disk Load	percent	Approximate percentage of total disk bandwidth being used. The weighted I/O time series includes the backlog that might be accumulating. For more information, see the <a href="#">collected disk plugin docs</a> .	
Operations/s	ops/s	Operations done per second	



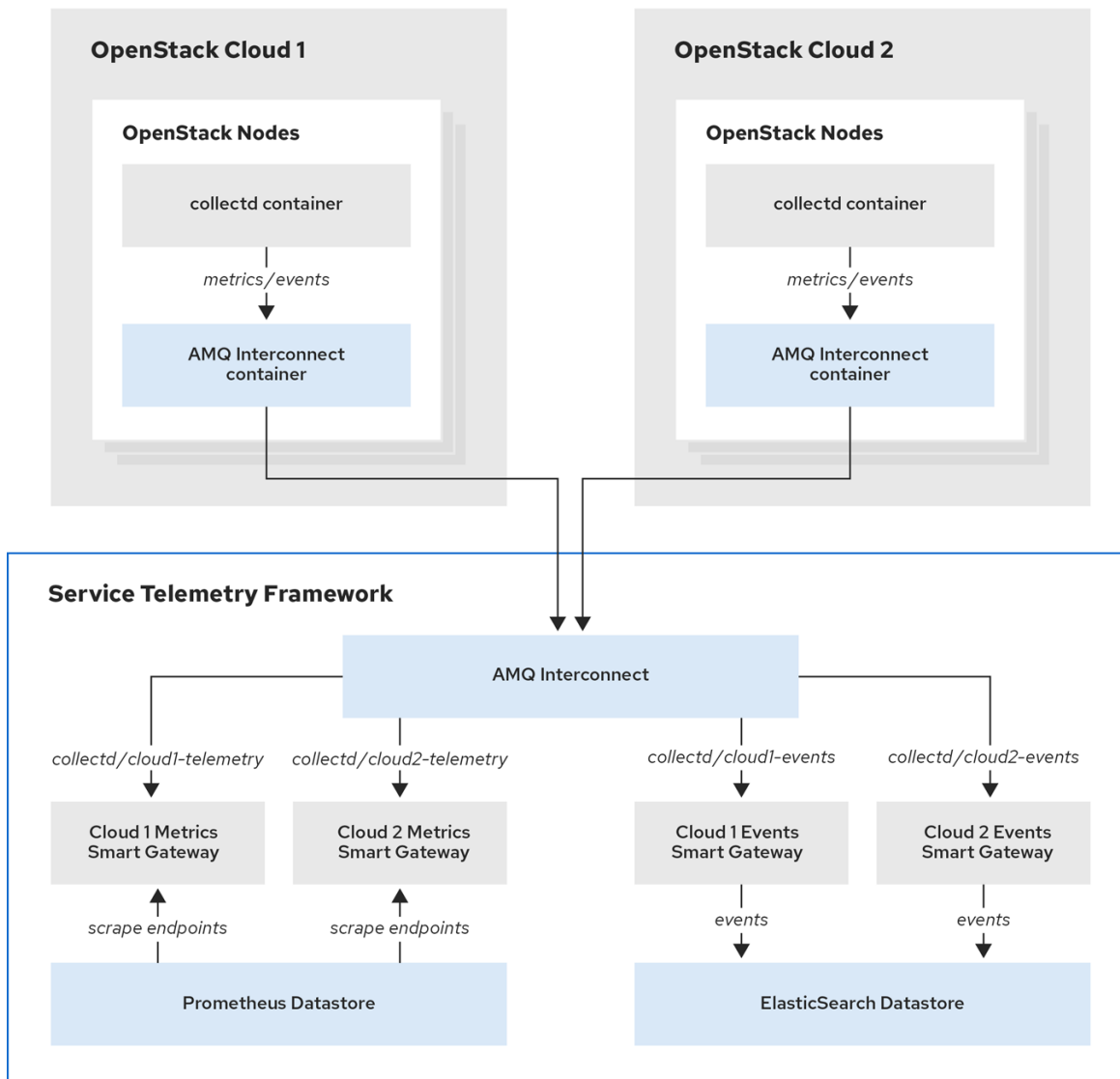
Panel	Unit	Description	Notes
Average I/O Operation Time	seconds	Average time each I/O operation took to complete. This average is not accurate, see the collectd <a href="#">disk plugin docs</a>	

## 4.6. 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 4.6.1, “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 4.6.2, “Deploying Smart Gateways”](#).
3. Configure each cloud to send its metrics and events to STF on the correct address. For more information, see [Section 4.6.4, “Creating the OpenStack environment file”](#).

Figure 4.1. Two Red Hat OpenStack Platform clouds connect to STF



65\_OpenStack\_0120

#### 4.6.1. 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`
- `anycast/ceilometer/cloud2-metering.sample`
- `anycast/ceilometer/cloud2-event.sample`
- `collectd/us-east-1-telemetry`
- `collectd/us-west-3-telemetry`

#### 4.6.2. 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:

collector	type	default subscription address
collectd	metrics	collectd/telemetry
collectd	events	collectd/notify
Ceilometer	metrics	anycast/ceilometer/metering.sample
Ceilometer	events	anycast/ceilometer/event.sample

#### 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\\_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 `acLOUDS` 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
```

NAME	READY	STATUS	RESTARTS	AGE
default-cloud1-ceil-event-smartgateway-6cfb65478c-g5q82	1/1	Running	0	13h
default-cloud1-ceil-meter-smartgateway-58f885c76d-xmxwn	1/1	Running	0	13h
default-cloud1-coll-event-smartgateway-58fbbd4485-rl9bd	1/1	Running	0	13h
default-cloud1-coll-meter-smartgateway-7c6fc495c4-jn728	2/2	Running	0	13h

### 4.6.3. 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 2.3.10.2, “clouds”](#).
2. Edit the `ServiceTelemetry` object and add the `cloudsRemoveOnMissing` parameter:

```
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
```

**4.6.4. 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. For more information, see [Section 4.6.1, “Planning AMQP address prefixes”](#).

**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:

`stf-connectors.yaml`

```

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: []
  MetricPipelinePublishers: []
  CeilometerEnablePanko: false
  CeilometerQdrPublishEvents: true
  CeilometerQdrEventsConfig:
    driver: amqp
    topic: cloud1-event 1
  CeilometerQdrMetricsConfig:
    driver: amqp
    topic: cloud1-metering 2

  CollectdConnectionType: amqp1
  CollectdAmqpInterval: 5
  CollectdDefaultPollingInterval: 5

  CollectdAmqpInstances:
    cloud1-notify: 3
      notify: true
      format: JSON
      presettle: false
    cloud1-telemetry: 4
      format: JSON
      presettle: true

  MetricsQdrAddresses:
    - prefix: collectd
      distribution: multicast

```

```
- prefix: anycast/ceilometer
  distribution: multicast
```

**MetricsQdrSSLProfiles:**

```
- name: sslProfile
```

**MetricsQdrConnectors:**

```
- host: stf-default-interconnect-5671-service-telemetry.apps.infra.watch 5
  port: 443
  role: edge
  verifyHostname: false
  sslProfile: sslProfile
```

- 1** Define the topic for Ceilometer events. This value is the address format of `anycast/ceilometer/cloud1-event.sample`.
  - 2** Define the topic for Ceilometer metrics. This value is the address format of `anycast/ceilometer/cloud1-metering.sample`.
  - 3** Define the topic for collectd events. This value is the format of `collectd/cloud1-notify`.
  - 4** Define the topic for collectd metrics. This value is the format of `collectd/cloud1-telemetry`.
  - 5** Adjust the `MetricsQdrConnectors` host to the address of the STF route.
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/custom_templates/`.
  4. Source the authentication file:

```
[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 3.3.3, “Validating client-side installation”](#).

#### 4.6.5. 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"}`.

## 4.7. EPHEMERAL STORAGE

Use ephemeral storage to run Service Telemetry Framework (STF) without persistently storing data in your Red Hat OpenShift Container Platform (OCP) cluster. Ephemeral storage is not recommended in a production environment due to the volatility of the data in the platform when operating correctly and as designed. For example, restarting a pod or rescheduling the workload to another node results in the loss of any local data written since the pod started.

### 4.7.1. 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 `thespec` section of the relevant component:

```
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.

## 4.8. 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` plug-in 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 3.3, “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`:

```
CollectdEnableLibpodstats: true
```

3. Continue with the overcloud deployment procedure.

## APPENDIX A. COLLECTD PLUG-INS

This section contains a complete list of collectd plug-ins and configurations for Red Hat OpenStack Platform 16.1.

### collectd-aggregation

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

### collectd-amqp1

### collectd-apache

- `collectd::plugin::apache::instances` (ex.: `{localhost ⇒ {url ⇒ http://localhost/mod\_status?auto}}`)
- `collectd::plugin::apache::interval`

### collectd-apcups

### collectd-battery

- `collectd::plugin::battery::values_percentage`
- `collectd::plugin::battery::report_degraded`
- `collectd::plugin::battery::query_state_fs`
- `collectd::plugin::battery::interval`

### collectd-ceph

- `collectd::plugin::ceph::daemons`
- `collectd::plugin::ceph::longrunavglatency`
- `collectd::plugin::ceph::convertspecialmetricitypes`

### collectd-cgroups

- `collectd::plugin::cgroups::ignore_selected`
- `collectd::plugin::cgroups::interval`

### collectd-contrack

- `None`

### collectd-contextswitch

- `collectd::plugin::contextswitch::interval`

### collectd-cpu

- `collectd::plugin::cpu::reportbystate`
- `collectd::plugin::cpu::reportbycpu`
- `collectd::plugin::cpu::valuespercentage`
- `collectd::plugin::cpu::reportnumcpu`
- `collectd::plugin::cpu::reportgueststate`
- `collectd::plugin::cpu::subtractgueststate`
- `collectd::plugin::cpu::interval`

#### `collectd-cpufreq`

- `None`

#### `collectd-cpusleep`

#### `collectd-csv`

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

#### `collectd-df`

- `collectd::plugin::df::devices`
- `collectd::plugin::df::fstypes`
- `collectd::plugin::df::ignoreselected`
- `collectd::plugin::df::mountpoints`
- `collectd::plugin::df::reportbydevice`
- `collectd::plugin::df::reportinodes`
- `collectd::plugin::df::reportreserved`
- `collectd::plugin::df::valuesabsolute`
- `collectd::plugin::df::valuespercentage`
- `collectd::plugin::df::interval`

#### `collectd-disk`

- `collectd::plugin::disk::disks`
- `collectd::plugin::disk::ignoreselected`
- `collectd::plugin::disk::udevnameattr`

- `collectd::plugin::disk::interval`

#### `collectd-entropy`

- `collectd::plugin::entropy::interval`

#### `collectd-ethstat`

- `collectd::plugin::ethstat::interfaces`
- `collectd::plugin::ethstat::maps`
- `collectd::plugin::ethstat::mappedonly`
- `collectd::plugin::ethstat::interval`

#### `collectd-exec`

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

#### `collectd-fhcount`

- `collectd::plugin::fhcount::valuesabsolute`
- `collectd::plugin::fhcount::valuespercentage`
- `collectd::plugin::fhcount::interval`

#### `collectd-filecount`

- `collectd::plugin::filecount::directories`
- `collectd::plugin::filecount::interval`

#### `collectd-fscache`

- `None`

#### `collectd-hddtemp`

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

#### `collectd-hugepages`

- `collectd::plugin::hugepages::report_per_node_hp`

- `collectd::plugin::hugepages::report_root_hp`
- `collectd::plugin::hugepages::values_pages`
- `collectd::plugin::hugepages::values_bytes`
- `collectd::plugin::hugepages::values_percentage`
- `collectd::plugin::hugepages::interval`

#### `collectd-intel_rdt`

#### `collectd-interface`

- `collectd::plugin::interface::interfaces`
- `collectd::plugin::interface::ignoreselected`
- `collectd::plugin::interface::reportinactive`
- `Collectd::plugin::interface::interval`

#### `collectd-ipc`

- `None`

#### `collectd-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-irq`

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

#### `collectd-load`

- `collectd::plugin::load::report_relative`
- `collectd::plugin::load::interval`

#### `collectd-logfile`

- `collectd::plugin::logfile::log_level`

- collectd::plugin::logfile::log\_file
- collectd::plugin::logfile::log\_timestamp
- collectd::plugin::logfile::print\_severity
- collectd::plugin::logfile::interval

collectd-madwifi

collectd-mbmon

collectd-md

collectd-memcached

- collectd::plugin::memcached::instances
- collectd::plugin::memcached::interval

collectd-memory

- collectd::plugin::memory::valuesabsolute
- collectd::plugin::memory::valuespercentage
- collectd::plugin::memory::interval collectd-multimeter

collectd-multimeter

collectd-mysql

- collectd::plugin::mysql::interval

collectd-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-network

- collectd::plugin::network::timetolive
- collectd::plugin::network::maxpacketize

- `collectd::plugin::network::forward`
- `collectd::plugin::network::reportstats`
- `collectd::plugin::network::listeners`
- `collectd::plugin::network::servers`
- `collectd::plugin::network::interval`

#### `collectd-nfs`

- `collectd::plugin::nfs::interval`

#### `collectd-ntpd`

- `collectd::plugin::ntpd::host`
- `collectd::plugin::ntpd::port`
- `collectd::plugin::ntpd::reverselookups`
- `collectd::plugin::ntpd::includeunitid`
- `collectd::plugin::ntpd::interval`

#### `collectd-numa`

- `None`

#### `collectd-olsrd`

#### `collectd-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-ovs_events`

- `collectd::plugin::ovs_events::address`
- `collectd::plugin::ovs_events::dispatch`
- `collectd::plugin::ovs_events::interfaces`
- `collectd::plugin::ovs_events::send_notification`

- collectd::plugin::ovs\_events::\$port
- collectd::plugin::ovs\_events::socket

#### collectd-ovs\_stats

- collectd::plugin::ovs\_stats::address
- collectd::plugin::ovs\_stats::bridges
- collectd::plugin::ovs\_stats::port
- collectd::plugin::ovs\_stats::socket

#### collectd-ping

- collectd::plugin::ping::hosts
- collectd::plugin::ping::timeout
- collectd::plugin::ping::ttl
- collectd::plugin::ping::source\_address
- collectd::plugin::ping::device
- collectd::plugin::ping::max\_missed
- collectd::plugin::ping::size
- collectd::plugin::ping::interval

#### collectd-powerdns

- collectd::plugin::powerdns::interval
- collectd::plugin::powerdns::servers
- collectd::plugin::powerdns::recursors
- collectd::plugin::powerdns::local\_socket
- collectd::plugin::powerdns::interval

#### collectd-processes

- collectd::plugin::processes::processes
- collectd::plugin::processes::process\_matches
- collectd::plugin::processes::collect\_context\_switch
- collectd::plugin::processes::collect\_file\_descriptor
- collectd::plugin::processes::collect\_memory\_maps
- collectd::plugin::powerdns::interval



---

### collectd-protocols

- collectd::plugin::protocols::ignoreselected
- collectd::plugin::protocols::values

### collectd-python

### collectd-serial

### collectd-smart

- collectd::plugin::smart::disks
- collectd::plugin::smart::ignoreselected
- collectd::plugin::smart::interval

### collectd-snmp\_agent

### collectd-statsd

- collectd::plugin::statsd::host
- collectd::plugin::statsd::port
- collectd::plugin::statsd::deletecounters
- collectd::plugin::statsd::deletetimers
- collectd::plugin::statsd::deletegauges
- collectd::plugin::statsd::deletesets
- collectd::plugin::statsd::countersum
- collectd::plugin::statsd::timerpercentile
- collectd::plugin::statsd::timerlower
- collectd::plugin::statsd::timerupper
- collectd::plugin::statsd::timersum
- collectd::plugin::statsd::timercount
- collectd::plugin::statsd::interval

### collectd-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-syslog`

- `collectd::plugin::syslog::log_level`
- `collectd::plugin::syslog::notify_level`
- `collectd::plugin::syslog::interval`

#### `collectd-table`

- `collectd::plugin::table::tables`
- `collectd::plugin::table::interval`

#### `collectd-tail`

- `collectd::plugin::tail::files`
- `collectd::plugin::tail::interval`

#### `collectd-tail_csv`

- `collectd::plugin::tail_csv::metrics`
- `collectd::plugin::tail_csv::files`

#### `collectd-tcpconns`

- `collectd::plugin::tcpconns::localports`
- `collectd::plugin::tcpconns::remoteports`
- `collectd::plugin::tcpconns::listening`
- `collectd::plugin::tcpconns::allportssummary`
- `collectd::plugin::tcpconns::interval`

#### `collectd-ted`

#### `collectd-thermal`

- `collectd::plugin::thermal::devices`
- `collectd::plugin::thermal::ignoreselected`
- `collectd::plugin::thermal::interval`

#### `collectd-threshold`

- `collectd::plugin::threshold::types`
- `collectd::plugin::threshold::plugins`
- `collectd::plugin::threshold::hosts`
- `collectd::plugin::threshold::interval`

#### `collectd-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-unixsock`

#### `collectd-uptime`

- `collectd::plugin::uptime::interval`

#### `collectd-users`

- `collectd::plugin::users::interval`

#### `collectd-uuid`

- `collectd::plugin::uuid::uuid_file`
- `collectd::plugin::uuid::interval`

#### `collectd-virt`

- `collectd::plugin::virt::connection`
- `collectd::plugin::virt::refresh_interval`
- `collectd::plugin::virt::domain`
- `collectd::plugin::virt::block_device`
- `collectd::plugin::virt::interface_device`
- `collectd::plugin::virt::ignore_selected`
- `collectd::plugin::virt::hostname_format`

- collectd::plugin::virt::interface\_format
- collectd::plugin::virt::extra\_stats
- collectd::plugin::virt::interval

#### collectd-vmem

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

#### collectd-vserver

#### collectd-wireless

#### collectd-write\_graphite

- collectd::plugin::write\_graphite::carbons
- collectd::plugin::write\_graphite::carbon\_defaults
- collectd::plugin::write\_graphite::globals

#### collectd-write\_kafka

- collectd::plugin::write\_kafka::kafka\_host
- collectd::plugin::write\_kafka::kafka\_port
- collectd::plugin::write\_kafka::kafka\_hosts
- collectd::plugin::write\_kafka::topics

#### collectd-write\_log

- collectd::plugin::write\_log::format

#### collectd-zfs\_arc

- None