



Red Hat OpenStack Platform 16.1

Service Telemetry Framework 1.5

Installing and deploying Service Telemetry Framework 1.5

Red Hat OpenStack Platform 16.1 Service Telemetry Framework 1.5

Installing and deploying Service Telemetry Framework 1.5

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Copyright © 2023 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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] 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[®] 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

Install the core components and deploy Service Telemetry Framework 1.5

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	4
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	5
CHAPTER 1. INTRODUCTION TO SERVICE TELEMETRY FRAMEWORK 1.5	6
1.1. SUPPORT FOR SERVICE TELEMETRY FRAMEWORK	6
1.2. SERVICE TELEMETRY FRAMEWORK ARCHITECTURE	6
1.3. INSTALLATION SIZE OF RED HAT OPENSIFT CONTAINER PLATFORM	9
CHAPTER 2. PREPARING YOUR RED HAT OPENSIFT CONTAINER PLATFORM ENVIRONMENT FOR SERVICE TELEMETRY FRAMEWORK	10
2.1. OBSERVABILITY STRATEGY IN SERVICE TELEMETRY FRAMEWORK	10
2.2. PERSISTENT VOLUMES	10
2.3. RESOURCE ALLOCATION	11
2.4. NETWORK CONSIDERATIONS FOR SERVICE TELEMETRY FRAMEWORK	11
CHAPTER 3. INSTALLING THE CORE COMPONENTS OF SERVICE TELEMETRY FRAMEWORK	12
3.1. DEPLOYING SERVICE TELEMETRY FRAMEWORK TO THE RED HAT OPENSIFT CONTAINER PLATFORM ENVIRONMENT	12
3.2. CREATING A SERVICETELEMETRY OBJECT IN RED HAT OPENSIFT CONTAINER PLATFORM	16
3.2.1. Primary parameters of the ServiceTelemetry object	20
The backends parameter	20
Enabling Prometheus as a storage back end for metrics	20
Configuring persistent storage for Prometheus	20
Enabling Elasticsearch as a storage back end for events	21
Configuring persistent storage for Elasticsearch	22
The clouds parameter	23
The alerting parameter	24
The graphing parameter	24
The highAvailability parameter	24
The transports parameter	24
3.3. ACCESSING USER INTERFACES FOR STF COMPONENTS	24
3.4. CONFIGURING AN ALTERNATE OBSERVABILITY STRATEGY	25
CHAPTER 4. CONFIGURING RED HAT OPENSTACK PLATFORM DIRECTOR FOR SERVICE TELEMETRY FRAMEWORK	27
4.1. DEPLOYING RED HAT OPENSTACK PLATFORM OVERCLOUD FOR SERVICE TELEMETRY FRAMEWORK USING DIRECTOR	27
4.1.1. Retrieving the AMQ Interconnect route address	27
4.1.2. Creating the base configuration for STF	28
4.1.3. Configuring the STF connection for the overcloud	30
4.1.4. Deploying the overcloud	31
4.1.5. Validating client-side installation	32
4.2. DISABLING RED HAT OPENSTACK PLATFORM SERVICES USED WITH SERVICE TELEMETRY FRAMEWORK	35
4.3. CONFIGURING MULTIPLE CLOUDS	36
4.3.1. Planning AMQP address prefixes	37
4.3.2. Deploying Smart Gateways	38
4.3.3. Deleting the default Smart Gateways	40
4.3.4. Setting a unique cloud domain	41
4.3.5. Creating the Red Hat OpenStack Platform environment file for multiple clouds	42
4.3.6. Querying metrics data from multiple clouds	45

CHAPTER 5. USING OPERATIONAL FEATURES OF SERVICE TELEMETRY FRAMEWORK	46
5.1. DASHBOARDS IN SERVICE TELEMETRY FRAMEWORK	46
5.1.1. Configuring Grafana to host the dashboard	47
5.1.2. Overriding the default Grafana container image	48
5.1.3. Importing dashboards	49
5.1.4. Retrieving and setting Grafana login credentials	50
5.2. METRICS RETENTION TIME PERIOD IN SERVICE TELEMETRY FRAMEWORK	51
5.2.1. Editing the metrics retention time period in Service Telemetry Framework	51
5.3. ALERTS IN SERVICE TELEMETRY FRAMEWORK	52
5.3.1. Creating an alert rule in Prometheus	53
5.3.2. Configuring custom alerts	54
5.3.3. Creating a standard alert route in Alertmanager	54
5.3.4. Creating an alert route with templating in Alertmanager	57
5.4. SENDING ALERTS AS SNMP TRAPS	58
5.4.1. Configuration parameters for snmpTraps	59
5.4.2. Overview of the MIB definition	60
5.4.3. Configuring SNMP traps	61
5.4.4. Creating alerts for SNMP traps	62
5.5. HIGH AVAILABILITY	63
5.5.1. Configuring high availability	63
5.6. OBSERVABILITY STRATEGY IN SERVICE TELEMETRY FRAMEWORK	64
5.6.1. Configuring an alternate observability strategy	64
5.7. RESOURCE USAGE OF RED HAT OPENSTACK PLATFORM SERVICES	65
5.7.1. Disabling resource usage monitoring of Red Hat OpenStack Platform services	65
5.8. RED HAT OPENSTACK PLATFORM API STATUS AND CONTAINERIZED SERVICES HEALTH	66
5.8.1. Disabling container health and API status monitoring	66
 CHAPTER 6. REMOVING SERVICE TELEMETRY FRAMEWORK FROM THE RED HAT OPENSIFT CONTAINER PLATFORM ENVIRONMENT	 67
6.1. DELETING THE NAMESPACE	67
6.2. REMOVING THE CERT-MANAGER OPERATOR	67
 CHAPTER 7. UPGRADING SERVICE TELEMETRY FRAMEWORK TO VERSION 1.5	 70
7.1. REMOVING THE SERVICE TELEMETRY FRAMEWORK 1.4 OPERATORS	70
7.1.1. Removing the Service Telemetry Operator	70
7.1.2. Removing the Smart Gateway Operator	71
7.1.3. Removing the AMQ Certificate Manager Operator	72
7.1.4. Removing the Grafana Operator	73
7.2. UPGRADING RED HAT OPENSIFT CONTAINER PLATFORM TO 4.10	74
7.3. INSTALLING THE SERVICE TELEMETRY FRAMEWORK 1.5 OPERATORS	74

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Using the Direct Documentation Feedback (DDF) function

Use the **Add Feedback** DDF function for direct comments on specific sentences, paragraphs, or code blocks.

1. View the documentation in the *Multi-page HTML* format.
2. Ensure that you see the **Feedback** button in the upper right corner of the document.
3. Highlight the part of text that you want to comment on.
4. Click **Add Feedback**.
5. Complete the **Add Feedback** field with your comments.
6. Optional: Add your email address so that the documentation team can contact you for clarification on your issue.
7. Click **Submit**.

CHAPTER 1. INTRODUCTION TO SERVICE TELEMETRY FRAMEWORK 1.5

Service Telemetry Framework (STF) collects monitoring data from Red Hat OpenStack Platform (RHOSP) or third-party nodes. You can use STF to perform the following tasks:

- Store or archive the monitoring data for historical information.
- View the monitoring data graphically on the dashboard.
- Use the monitoring data to trigger alerts or warnings.

The monitoring data can be either metric or event:

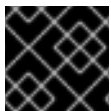
Metric

A numeric measurement of an application or system.

Event

Irregular and discrete occurrences that happen in a system.

The components of STF use a message bus for data transport. Other modular components that receive and store data are deployed as containers on Red Hat OpenShift Container Platform.



IMPORTANT

STF is compatible with Red Hat OpenShift Container Platform version 4.10 through 4.12.

Additional resources

- [Red Hat OpenShift Container Platform product documentation](#)
- [Service Telemetry Framework Performance and Scaling](#)
- [OpenShift Container Platform 4.12 Documentation](#)

1.1. SUPPORT FOR SERVICE TELEMETRY FRAMEWORK

Red Hat supports the core Operators and workloads, including AMQ Interconnect, Service Telemetry Operator, and Smart Gateway Operator. Red Hat does not support the community Operators or workload components, such as Elasticsearch, Prometheus, Alertmanager, Grafana, and their Operators.

You can only deploy STF in a fully connected network environment. You cannot deploy STF in Red Hat OpenShift Container Platform-disconnected environments or network proxy environments.

For more information about STF life cycle and support status, see the [Service Telemetry Framework Supported Version Matrix](#).

1.2. SERVICE TELEMETRY FRAMEWORK ARCHITECTURE

Service Telemetry Framework (STF) uses a client-server architecture, in which Red Hat OpenStack Platform (RHOSP) is the client and Red Hat OpenShift Container Platform is the server.

STF consists of the following components:

- Data collection
 - collectd: Collects infrastructure metrics and events.
 - Ceilometer: Collects RHOSP metrics and events.
- Transport
 - AMQ Interconnect: An AMQP 1.x compatible messaging bus that provides fast and reliable data transport to transfer the metrics to STF for storage.
 - Smart Gateway: A Golang application that takes metrics and events from the AMQP 1.x bus to deliver to Elasticsearch or Prometheus.
- Data storage
 - Prometheus: Time-series data storage that stores STF metrics received from the Smart Gateway.
 - Elasticsearch: Events data storage that stores STF events received from the Smart Gateway.
- Observation
 - Alertmanager: An alerting tool that uses Prometheus alert rules to manage alerts.
 - Grafana: A visualization and analytics application that you can use to query, visualize, and explore data.

The following table describes the application of the client and server components:

Table 1.1. Client and server components of STF

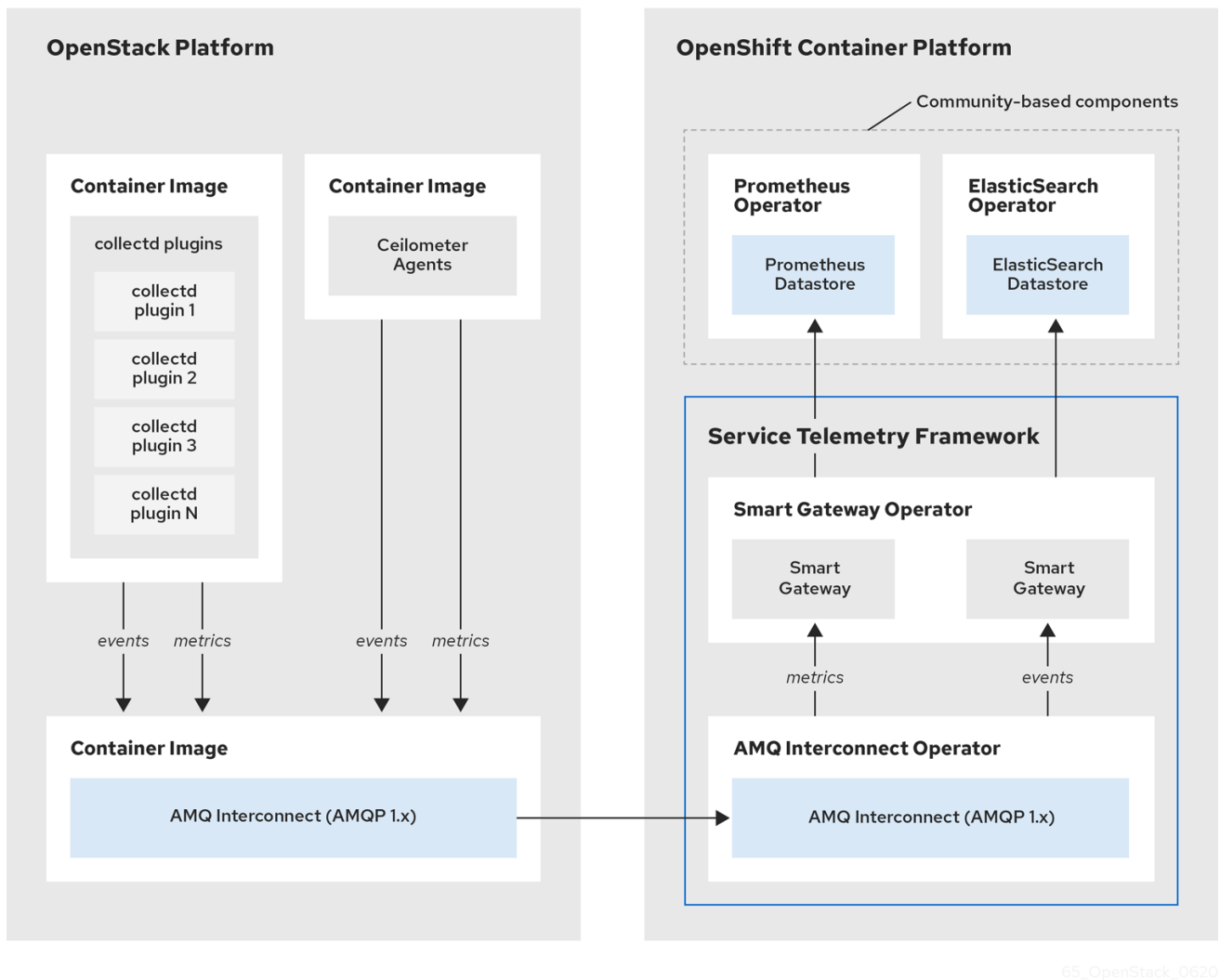
Component	Client	Server
An AMQP 1.x compatible messaging bus	yes	yes
Smart Gateway	no	yes
Prometheus	no	yes
Elasticsearch	no	yes
collectd	yes	no
Ceilometer	yes	no



IMPORTANT

To ensure that the monitoring platform can report operational problems with your cloud, do not install STF on the same infrastructure that you are monitoring.

Figure 1.1. Service Telemetry Framework architecture overview



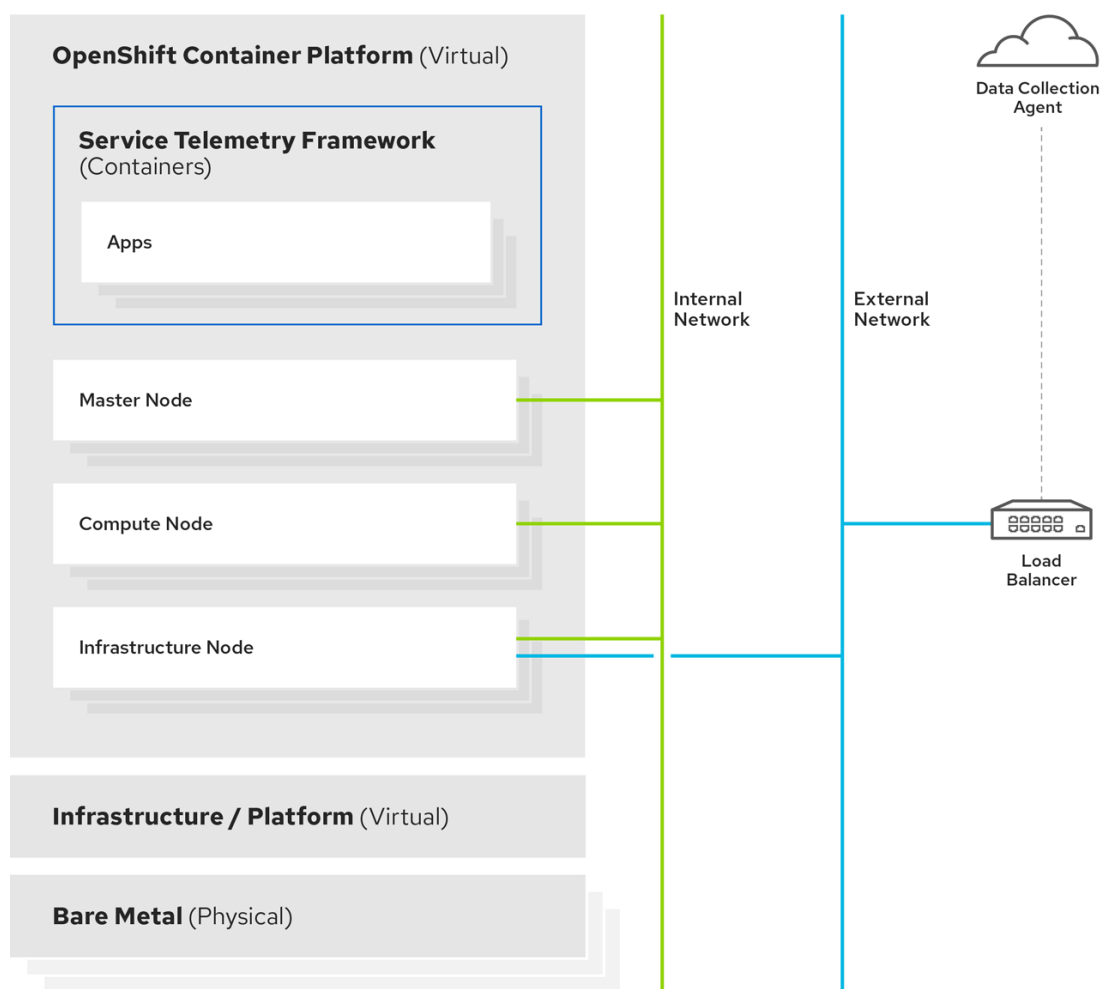
For client side metrics, collectd provides infrastructure metrics without project data, and Ceilometer provides RHOSP platform data based on projects or user workload. Both Ceilometer and collectd deliver data to Prometheus by using the AMQ Interconnect transport, delivering the data through the message 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 and Ceilometer deliver event data to the server side by using the AMQ Interconnect transport. Another Smart Gateway writes the data to the Elasticsearch datastore.

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

- Service Telemetry Framework 1.5
- Red Hat OpenShift Container Platform 4.10 through 4.12
- Infrastructure platform

Figure 1.2. Server-side STF monitoring infrastructure



65_OpenStack_0120

1.3. INSTALLATION SIZE OF RED HAT OPENSIFT CONTAINER PLATFORM

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

- The infrastructure that you select.
- The number of nodes that you want to monitor.
- The number of metrics that 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 an existing Red Hat OpenShift Container Platform environment.

For more information about minimum resources requirements when you install Red Hat OpenShift Container Platform 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 that you can install, see the corresponding installation documentation for your cloud platform of choice.

CHAPTER 2. PREPARING YOUR RED HAT OPENSIFT CONTAINER PLATFORM ENVIRONMENT FOR SERVICE TELEMETRY FRAMEWORK

To prepare your Red Hat OpenShift Container Platform environment for Service Telemetry Framework (STF), you must plan for persistent storage, adequate resources, event storage, and network considerations:

- Ensure that you have persistent storage available in your Red Hat OpenShift Container Platform cluster for a production-grade deployment. For more information, see [Section 2.2, “Persistent volumes”](#).
- Ensure that enough resources are available to run the Operators and the application containers. For more information, see [Section 2.3, “Resource allocation”](#).
- Ensure that you have a fully connected network environment. For more information, see [Section 2.4, “Network considerations for Service Telemetry Framework”](#).

2.1. OBSERVABILITY STRATEGY IN SERVICE TELEMETRY FRAMEWORK

Service Telemetry Framework (STF) does not include storage backends and alerting tools. STF uses community operators to deploy Prometheus, Alertmanager, Grafana, and Elasticsearch. STF makes requests to these community operators to create instances of each application configured to work with STF.

Instead of having Service Telemetry Operator create custom resource requests, you can use your own deployments of these applications or other compatible applications, and scrape the metrics Smart Gateways for delivery to your own Prometheus-compatible system for telemetry storage. If you set the **observabilityStrategy** to **none**, then storage backends will not be deployed so persistent storage will not be required by STF.

2.2. PERSISTENT VOLUMES

Service Telemetry Framework (STF) uses persistent storage in Red Hat OpenShift Container Platform to request persistent volumes so that Prometheus and Elasticsearch can store metrics and events.

When you enable persistent storage through the Service Telemetry Operator, the Persistent Volume Claims (PVC) 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 Red Hat OpenShift Container Platform default configured **storageClass**.

Additional resources

- For more information about configuring persistent storage for Red Hat OpenShift Container Platform, see [Understanding persistent storage](#).
- For more information about recommended configurable storage technology in Red Hat OpenShift Container Platform, see [Recommended configurable storage technology](#).
- For more information about configuring persistent storage for Prometheus in STF, see [the section called “Configuring persistent storage for Prometheus”](#).

- For more information about configuring persistent storage for Elasticsearch in STF, see [the section called "Configuring persistent storage for Elasticsearch"](#).

2.3. RESOURCE ALLOCATION

To enable the scheduling of pods within the Red Hat OpenShift Container Platform 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 Service Telemetry Framework (STF) depends on your environment and the number of nodes and clouds that you want to monitor.

Additional resources

- For recommendations about sizing for metrics collection, see [Service Telemetry Framework Performance and Scaling](#).
- For information about sizing requirements for Elasticsearch, see <https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-managing-compute-resources.html>.

2.4. NETWORK CONSIDERATIONS FOR SERVICE TELEMETRY FRAMEWORK

You can only deploy Service Telemetry Framework (STF) in a fully connected network environment. You cannot deploy STF in Red Hat OpenShift Container Platform-disconnected environments or network proxy environments.

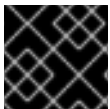
CHAPTER 3. INSTALLING THE CORE COMPONENTS OF SERVICE TELEMETRY FRAMEWORK

You can use Operators to load the Service Telemetry Framework (STF) components and objects. Operators manage each of the following STF core and community components:

- cert-manager
- AMQ Interconnect
- Smart Gateway
- Prometheus and AlertManager
- Elasticsearch
- Grafana

Prerequisites

- An Red Hat OpenShift Container Platform version inclusive of 4.10 through 4.12 is running.
- You have prepared your Red Hat OpenShift Container Platform environment and ensured that there is persistent storage and enough resources to run the STF components on top of the Red Hat OpenShift Container Platform environment. For more information, see [Service Telemetry Framework Performance and Scaling](#).
- Your environment is fully connected. STF does not work in a Red Hat OpenShift Container Platform-disconnected environments or network proxy environments.



IMPORTANT

STF is compatible with Red Hat OpenShift Container Platform version 4.10 through 4.12.

Additional resources

- For more information about Operators, see the [Understanding Operators](#) guide.
- For more information about Operator catalogs, see [Red Hat-provided Operator catalogs](#).

3.1. DEPLOYING SERVICE TELEMETRY FRAMEWORK TO THE RED HAT OPENSIFT CONTAINER PLATFORM ENVIRONMENT

Deploy Service Telemetry Framework (STF) to collect, store, and monitor events:

Procedure

1. Create a namespace to contain the STF components, for example, **service-telemetry**:

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

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

■


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

For more information, see [OperatorGroups](#).

3. Create a namespace for the cert-manager Operator:

```
$ oc create -f - <<EOF
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  name: openshift-cert-manager-operator
spec:
  finalizers:
  - kubernetes
EOF
```

4. Create an OperatorGroup for the cert-manager Operator:

```
$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-cert-manager-operator
  namespace: openshift-cert-manager-operator
spec: {}
EOF
```

5. Subscribe to the cert-manager Operator by using the redhat-operators CatalogSource:

```
$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-cert-manager-operator
  namespace: openshift-cert-manager-operator
spec:
  channel: tech-preview
  installPlanApproval: Automatic
  name: openshift-cert-manager-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

6. Validate your ClusterServiceVersion. Ensure that cert-manager Operator displays a phase of **Succeeded**:

```
$ oc get csv --namespace openshift-cert-manager-operator --
selector=operators.coreos.com/openshift-cert-manager-operator.openshift-cert-manager-
operator
```

NAME	DISPLAY	VERSION	REPLACES	PHASE
openshift-cert-manager.v1.7.1	cert-manager Operator	for Red Hat OpenShift	1.7.1-1	Succeeded

7. Subscribe to the AMQ Interconnect Operator by using the redhat-operators CatalogSource:

```
$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq7-interconnect-operator
  namespace: service-telemetry
spec:
  channel: 1.10.x
  installPlanApproval: Automatic
  name: amq7-interconnect-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

8. Validate your ClusterServiceVersion. Ensure that amq7-interconnect-operator.v1.10.x displays a phase of **Succeeded**:

```
$ oc get csv --selector=operators.coreos.com/amq7-interconnect-operator.service-telemetry
```

NAME	DISPLAY	VERSION	REPLACES	PHASE
amq7-interconnect-operator.v1.10.15	Red Hat Integration - AMQ Interconnect	1.10.15		
amq7-interconnect-operator.v1.10.4				Succeeded

9. To store metrics in Prometheus, you must enable the Prometheus Operator by using the community-operators CatalogSource:



WARNING

Community Operators are Operators which have not been vetted or verified by Red Hat. Community Operators should be used with caution because their stability is unknown. Red Hat provides no support for community Operators.

[Learn more about Red Hat's third party software support policy](#)

```
$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```

```

metadata:
  name: prometheus-operator
  namespace: service-telemetry
spec:
  channel: beta
  installPlanApproval: Automatic
  name: prometheus
  source: community-operators
  sourceNamespace: openshift-marketplace
EOF

```

10. Verify that the ClusterServiceVersion for Prometheus **Succeeded**:

```

$ oc get csv --selector=operators.coreos.com/prometheus.service-telemetry

NAME                                DISPLAY                VERSION  REPLACES                PHASE
prometheusoperator.0.56.3  Prometheus Operator  0.56.3  prometheusoperator.0.47.0
Succeeded

```

11. To store events in Elasticsearch, you must enable the Elastic Cloud on Kubernetes (ECK) Operator by using the certified-operators CatalogSource:



WARNING

Certified Operators are Operators from leading independent software vendors (ISVs). Red Hat partners with ISVs to package and ship, but not support, the certified Operators. Supported is provided by the ISV.

[Learn more about Red Hat's third party software support policy](#)

```

$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: elasticsearch-eck-operator-certified
  namespace: service-telemetry
spec:
  channel: stable
  installPlanApproval: Automatic
  name: elasticsearch-eck-operator-certified
  source: certified-operators
  sourceNamespace: openshift-marketplace
EOF

```

12. Verify that the ClusterServiceVersion for Elastic Cloud on Kubernetes **Succeeded**:

```

$ oc get csv --selector=operators.coreos.com/elasticsearch-eck-operator-certified.service-telemetry

```

NAME	DISPLAY	VERSION	REPLACES
PHASE			
elasticsearch-eck-operator-certified.v2.8.0	Elasticsearch (ECK) Operator	2.8.0	
elasticsearch-eck-operator-certified.v2.7.0	Succeeded		

13. Create the Service Telemetry Operator subscription to manage the STF instances:

```
$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: service-telemetry-operator
  namespace: service-telemetry
spec:
  channel: stable-1.5
  installPlanApproval: Automatic
  name: service-telemetry-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

14. Validate the Service Telemetry Operator and the dependent operators have their phase as Succeeded:

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

NAME	DISPLAY	VERSION
REPLACES	PHASE	
amq7-interconnect-operator.v1.10.15	Red Hat Integration - AMQ Interconnect	
1.10.15	amq7-interconnect-operator.v1.10.4	Succeeded
elasticsearch-eck-operator-certified.v2.8.0	Elasticsearch (ECK) Operator	2.8.0
elasticsearch-eck-operator-certified.v2.7.0	Succeeded	
openshift-cert-manager.v1.7.1	cert-manager Operator for Red Hat OpenShift	
1.7.1-1	Succeeded	
prometheusoperator.0.56.3	Prometheus Operator	0.56.3
prometheusoperator.0.47.0	Succeeded	
service-telemetry-operator.v1.5.1680516659	Service Telemetry Operator	
1.5.1680516659	Succeeded	
smart-gateway-operator.v5.0.1680516659	Smart Gateway Operator	
5.0.1680516659	Succeeded	

3.2. CREATING A SERVICETELEMETRY OBJECT IN RED HAT OPENSIFT CONTAINER PLATFORM

Create a **ServiceTelemetry** object in Red Hat OpenShift Container Platform to result in the Service Telemetry Operator creating the supporting components for a Service Telemetry Framework (STF) deployment. For more information, see [Section 3.2.1, "Primary parameters of the ServiceTelemetry object"](#).

Procedure

1. To create a **ServiceTelemetry** object that results in an STF deployment that uses the default values, create a **ServiceTelemetry** object with an empty **spec** parameter:

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

Creating a **ServiceTelemetry** object with an empty **spec** parameter results in an STF deployment with the following default settings:

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
spec:
  alerting:
    alertmanager:
      receivers:
        snmpTraps:
          alertOidLabel: oid
          community: public
          enabled: false
          port: 162
          retries: 5
          target: 192.168.24.254
          timeout: 1
          trapDefaultOid: 1.3.6.1.4.1.50495.15.1.2.1
          trapDefaultSeverity: ""
          trapOidPrefix: 1.3.6.1.4.1.50495.15
        storage:
          persistent:
            pvcStorageRequest: 20G
            strategy: persistent
          enabled: true
  backends:
    events:
      elasticsearch:
        certificates:
          caCertDuration: 70080h
          endpointCertDuration: 70080h
        storage:
          persistent:
            pvcStorageRequest: 20Gi
            strategy: persistent
          enabled: false
          version: 7.16.1
  logs:
    loki:
      storage:
        objectStorageSecret: test
        storageClass: standard
        enabled: false
```

```

    flavor: 1x.extra-small
    replicationFactor: 1
  metrics:
    prometheus:
      storage:
        persistent:
          pvcStorageRequest: 20G
          retention: 24h
          strategy: persistent
        enabled: true
        scrapInterval: 10s
  clouds:
    - events:
        collectors:
          - bridge:
              ringBufferCount: 15000
              ringBufferSize: 16384
              verbose: false
              collectorType: collectd
              debugEnabled: false
              subscriptionAddress: collectd/cloud1-notify
          - bridge:
              ringBufferCount: 15000
              ringBufferSize: 16384
              verbose: false
              collectorType: ceilometer
              debugEnabled: false
              subscriptionAddress: anycast/ceilometer/cloud1-event.sample
        metrics:
          collectors:
            - bridge:
                ringBufferCount: 15000
                ringBufferSize: 16384
                verbose: false
                collectorType: collectd
                debugEnabled: false
                subscriptionAddress: collectd/cloud1-telemetry
            - bridge:
                ringBufferCount: 15000
                ringBufferSize: 16384
                verbose: false
                collectorType: ceilometer
                debugEnabled: false
                subscriptionAddress: anycast/ceilometer/cloud1-metering.sample
            - bridge:
                ringBufferCount: 15000
                ringBufferSize: 16384
                verbose: false
                collectorType: sensubility
                debugEnabled: false
                subscriptionAddress: sensubility/cloud1-telemetry
          name: cloud1
  graphing:
    grafana:
      adminPassword: secret
      adminUser: root

```

```

    disableSignoutMenu: false
    ingressEnabled: false
    enabled: false
    highAvailability:
      enabled: false
    transports:
      qdr:
        certificates:
          caCertDuration: 70080h
          endpointCertDuration: 70080h
        web:
          enabled: false
          enabled: true
    observabilityStrategy: use_community

```

To override these defaults, add the configuration to the **spec** parameter.

2. View the STF deployment logs in the Service Telemetry Operator:

```

$ oc logs --selector name=service-telemetry-operator

...
----- Ansible Task Status Event StdOut -----

PLAY RECAP *****
localhost      : ok=90  changed=0  unreachable=0  failed=0  skipped=26
rescued=0  ignored=0

```

Verification

- To determine that all workloads are operating correctly, view the pods and the status of each pod.



NOTE

If you set the **backends.events.elasticsearch.enabled** parameter to **true**, the notification Smart Gateways report **Error** and **CrashLoopBackOff** error messages for a period of time before Elasticsearch starts.

```
$ oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
alertmanager-default-0	3/3	Running	0	4m7s
default-cloud1-ceil-meter-smartgateway-669c6cdf9-xvdvx	3/3	Running	0	3m46s
default-cloud1-coll-meter-smartgateway-585855c59d-858rf	3/3	Running	0	3m46s
default-cloud1-sens-meter-smartgateway-6f8dff645-hhgkw	3/3	Running	0	3m46s
default-interconnect-6994ff546-fx7jn	1/1	Running	0	4m18s
elastic-operator-9f44cdf6c-csvjq	1/1	Running	0	19m
interconnect-operator-646bfc886c-gx55n	1/1	Running	0	25m
prometheus-default-0	3/3	Running	0	3m33s
prometheus-operator-54d644d8d7-wzdlh	1/1	Running	0	20m
service-telemetry-operator-54f6f7b6d-nfhwX	1/1	Running	0	18m
smart-gateway-operator-9bbd7c56c-76w67	1/1	Running	0	18m

3.2.1. Primary parameters of the ServiceTelemetry object

The **ServiceTelemetry** object comprises the following primary configuration parameters:

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

You can configure each of these configuration parameters to provide different features in an STF deployment.

The backends parameter

Use the **backends** parameter to control which storage back ends are available for storage of metrics and events, and to control the enablement of Smart Gateways that the **clouds** parameter defines. For more information, see [the section called "The clouds parameter"](#).

You can use Prometheus as the metrics storage back end and Elasticsearch as the events storage back end. You can use the Service Telemetry Operator to create other custom resource objects that the Prometheus Operator and Elastic Cloud on Kubernetes Operator watch to create Prometheus and Elasticsearch workloads.

Enabling Prometheus as a storage back end for metrics

To enable Prometheus as a storage back end for metrics, you must configure the **ServiceTelemetry** object.

Procedure

1. Edit the **ServiceTelemetry** object:

```
$ oc edit stf default
```

2. Set the value of the `backends.metrics.prometheus.enabled` parameter to **true**:

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

Configuring persistent storage for Prometheus

Use the additional parameters that are defined in **backends.metrics.prometheus.storage.persistent** to configure persistent storage options for Prometheus, such as storage class and volume size.

Use **storageClass** to define the back end storage class. If you do not set this parameter, the Service Telemetry Operator uses the default storage class for the Red Hat OpenShift Container Platform cluster.

Use the **pvcStorageRequest** parameter to define the minimum required volume size to satisfy the storage request. If volumes are statically defined, it is possible that a volume size larger than requested is used. By default, Service Telemetry Operator requests a volume size of **20G** (20 Gigabytes).

Procedure

1. List the available storage classes:

```
$ oc get storageclasses
NAME          PROVISIONER          RECLAIMPOLICY  VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION  AGE
csi-manila-ceph  manila.csi.openstack.org  Delete        Immediate        false
20h
standard (default)  kubernetes.io/cinder    Delete        WaitForFirstConsumer  true
20h
standard-csi      cinder.csi.openstack.org  Delete        WaitForFirstConsumer  true
20h
```

2. Edit the **ServiceTelemetry** object:

```
$ oc edit stf default
```

3. Set the value of the `backends.metrics.prometheus.enabled` parameter to **true** and the value of `backends.metrics.prometheus.storage.strategy` to **persistent**:

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
spec:
  [...]
  backends:
    metrics:
      prometheus:
        enabled: true
        storage:
          strategy: persistent
          persistent:
            storageClass: standard-csi
            pvcStorageRequest: 50G
```

Enabling Elasticsearch as a storage back end for events

To enable Elasticsearch as a storage back end for events, you must configure the **ServiceTelemetry** object.

Procedure

1. Edit the **ServiceTelemetry** object:

```
$ oc edit stf default
```

- Set the value of the `backends.events.elasticsearch.enabled` parameter to **true**:

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
spec:
  [...]
  backends:
    events:
      elasticsearch:
        enabled: true
```

Configuring persistent storage for Elasticsearch

Use the additional parameters defined in **`backends.events.elasticsearch.storage.persistent`** to configure persistent storage options for Elasticsearch, such as storage class and volume size.

Use **`storageClass`** to define the back end storage class. If you do not set this parameter, the Service Telemetry Operator uses the default storage class for the Red Hat OpenShift Container Platform cluster.

Use the **`pvcStorageRequest`** parameter to define the minimum required volume size to satisfy the storage request. If volumes are statically defined, it is possible that a volume size larger than requested is used. By default, Service Telemetry Operator requests a volume size of **20Gi** (20 Gibibytes).

Procedure

- List the available storage classes:

```
$ oc get storageclasses
NAME          PROVISIONER          RECLAIMPOLICY  VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION  AGE
csi-manila-ceph  manila.csi.openstack.org Delete         Immediate      false
20h
standard (default)  kubernetes.io/cinder Delete         WaitForFirstConsumer true
20h
standard-csi      cinder.csi.openstack.org Delete         WaitForFirstConsumer true
20h
```

- Edit the **ServiceTelemetry** object:

```
$ oc edit stf default
```

- Set the value of the `backends.events.elasticsearch.enabled` parameter to **true** and the value of `backends.events.elasticsearch.storage.strategy` to **persistent**:

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
```

```

spec:
  [...]
  backends:
    events:
      elasticsearch:
        enabled: true
        version: 7.16.1
      storage:
        strategy: persistent
        persistent:
          storageClass: standard-csi
          pvcStorageRequest: 50G

```

The clouds parameter

Use the **clouds** parameter to define which Smart Gateway objects deploy, thereby providing the interface for multiple monitored cloud environments to connect to an instance of STF. If a supporting back end 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 the defined clouds. Each cloud consists of data types and collectors. Data types are **metrics** or **events**. Each data type consists of a list of collectors, the message bus subscription address, and a parameter to enable debugging. Available collectors for metrics are **collectd**, **ceilometer**, and **sensubility**. Available collectors for events 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, which provides subscriptions and data storage of metrics and events for collectd, Ceilometer, and Sensubility data collectors for a particular cloud instance:

```

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

```

Each item of the **clouds** parameter represents a cloud instance. A cloud instance consists of three top-level parameters: **name**, **metrics**, and **events**. The **metrics** and **events** parameters represent the

corresponding back end for storage of that data type. The **collectors** parameter specifies a list of objects made up of two required parameters, **collectorType** and **subscriptionAddress**, and these represent an instance of the Smart Gateway. The **collectorType** parameter specifies data collected by either collectd, Ceilometer, or Sensubility. The **subscriptionAddress** parameter provides the AMQ Interconnect address to which a Smart Gateway subscribes.

You can use the optional Boolean parameter **debugEnabled** within the **collectors** parameter to enable additional console debugging in the running Smart Gateway pod.

Additional resources

- For more information about deleting default Smart Gateways, see [Section 4.3.3, “Deleting the default Smart Gateways”](#).
- For more information about how to configure multiple clouds, see [Section 4.3, “Configuring multiple clouds”](#).

The alerting parameter

Use the **alerting** parameter to control creation of an Alertmanager instance and the configuration of the storage back end. By default, **alerting** is enabled. For more information, see [Section 5.3, “Alerts in Service Telemetry Framework”](#).

The graphing parameter

Use the **graphing** parameter to control the creation of a Grafana instance. By default, **graphing** is disabled. For more information, see [Section 5.1, “Dashboards in Service Telemetry Framework”](#).

The highAvailability parameter

Use the **highAvailability** parameter to control the instantiation of multiple copies of STF components to reduce recovery time of components that fail or are rescheduled. By default, **highAvailability** is disabled. For more information, see [Section 5.5, “High availability”](#).

The transports parameter

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

3.3. ACCESSING USER INTERFACES FOR STF COMPONENTS

In Red Hat OpenShift Container Platform, applications are exposed to the external network through a route. For more information about routes, see [Configuring ingress cluster traffic](#).

In Service Telemetry Framework (STF), HTTPS routes are exposed for each service that has a web-based interface. These routes are protected by Red Hat OpenShift Container Platform RBAC and any user that has a **ClusterRoleBinding** that enables them to view Red Hat OpenShift Container Platform Namespaces can log in. For more information about RBAC, see [Using RBAC to define and apply permissions](#).

Procedure

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

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

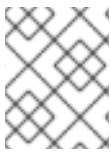
3. List the available web UI routes in the **service-telemetry** project:

```
$ oc get routes | grep web
default-alertmanager-proxy default-alertmanager-proxy-service-telemetry.apps.infra.watch
default-alertmanager-proxy web reencrypt/Redirect None
default-prometheus-proxy default-prometheus-proxy-service-telemetry.apps.infra.watch
default-prometheus-proxy web reencrypt/Redirect None
```

4. In a web browser, navigate to https://<route_address> to access the web interface for the corresponding service.

3.4. CONFIGURING AN ALTERNATE OBSERVABILITY STRATEGY

To configure STF to skip the deployment of storage, visualization, and alerting backends, add **observabilityStrategy: none** to the ServiceTelemetry spec. In this mode, only AMQ Interconnect routers and metrics Smart Gateways are deployed, and you must configure an external Prometheus-compatible system to collect metrics from the STF Smart Gateways.



NOTE

Currently, only metrics are supported when you set **observabilityStrategy** to **none**. Events Smart Gateways are not deployed.

Procedure

1. Create a **ServiceTelemetry** object with the property **observabilityStrategy: none** in the **spec** parameter. The manifest shows results in a default deployment of STF that is suitable for receiving telemetry from a single cloud with all metrics collector types.

```
$ oc apply -f - <<EOF
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
spec:
  observabilityStrategy: none
EOF
```

2. Delete the left over objects that are managed by community operators

```
$ for o in alertmanager/default prometheus/default elasticsearch/elasticsearch
grafana/default; do oc delete $o; done
```

3. To verify that all workloads are operating correctly, view the pods and the status of each pod:

```
$ oc get pods
NAME                                READY STATUS RESTARTS AGE
default-cloud1-ceil-meter-smartgateway-59c845d65b-gzhcs 3/3 Running 0 132m
default-cloud1-coll-meter-smartgateway-75bbd948b9-d5phm 3/3 Running 0 132m
default-cloud1-sens-meter-smartgateway-7fdbb57b6d-dh2g9 3/3 Running 0 132m
default-interconnect-668d5bbcd6-57b2l 1/1 Running 0 132m
interconnect-operator-b8f5bb647-tlp5t 1/1 Running 0 47h
service-telemetry-operator-566b9dd695-wkvjq 1/1 Running 0 156m
smart-gateway-operator-58d77dcf7-6xsq7 1/1 Running 0 47h
```

Additional resources

For more information about configuring additional clouds or to change the set of supported collectors, see [Section 4.3.2, “Deploying Smart Gateways”](#)

CHAPTER 4. CONFIGURING RED HAT OPENSTACK PLATFORM DIRECTOR FOR 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 (RHOSP) overcloud to enable data collection and transport.

STF can support both single and multiple clouds. The default configuration in RHOSP and STF set up for a single cloud installation.

- For a single RHOSP overcloud deployment with default configuration, see [Section 4.1, “Deploying Red Hat OpenStack Platform overcloud for Service Telemetry Framework using director”](#).
- To plan your RHOSP installation and configuration STF for multiple clouds, see [Section 4.3, “Configuring multiple clouds”](#).
- As part of an RHOSP overcloud deployment, you might need to configure additional features in your environment:
 - To disable the data collector services, see [Section 4.2, “Disabling Red Hat OpenStack Platform services used with Service Telemetry Framework”](#).

4.1. DEPLOYING RED HAT OPENSTACK PLATFORM OVERCLOUD FOR SERVICE TELEMETRY FRAMEWORK USING DIRECTOR

As part of the Red Hat OpenStack Platform (RHOSP) overcloud deployment using director, you must configure the data collectors and the data transport to Service Telemetry Framework (STF).

Procedure

1. [Retrieving the AMQ Interconnect route address](#)
2. [Creating the base configuration for STF](#)
3. [Configuring the STF connection for the overcloud](#)
4. [Deploying the overcloud](#)
5. [Validating client-side installation](#)

Additional resources

- For more information about deploying an OpenStack cloud using director, see [Director Installation and Usage](#).
- To collect data through AMQ Interconnect, see [the amqp1 plug-in](#).

4.1.1. Retrieving the AMQ Interconnect route address

When you configure the Red Hat OpenStack Platform (RHOSP) overcloud for Service Telemetry Framework (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 environment where STF is hosted.
2. Change to the **service-telemetry** project:

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

3. 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
```

4.1.2. Creating the base configuration for STF

To configure the base parameters to provide a compatible data collection and transport for Service Telemetry Framework (STF), you must create a file that defines the default data collection values.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Create a configuration file called **enable-stf.yaml** in the **/home/stack** directory.



IMPORTANT

Setting **EventPipelinePublishers** and **PipelinePublishers** to empty lists results in no event or metric data passing to RHOSP telemetry components, such as Gnocchi or Panko. If you need to send data to additional pipelines, the Ceilometer polling interval of **30** seconds, that you specify in **ExtraConfig**, might overwhelm the RHOSP telemetry components. You must increase the interval to a larger value, such as **300**, which results in less telemetry resolution in STF.

enable-stf.yaml

```
parameter_defaults:
  # only send to STF, not other publishers
  EventPipelinePublishers: []
  PipelinePublishers: []

  # manage the polling and pipeline configuration files for Ceilometer agents
  ManagePolling: true
  ManagePipeline: true

  # enable Ceilometer metrics and events
  CeilometerQdrPublishMetrics: true
  CeilometerQdrPublishEvents: true

  # enable collection of API status
  CollectdEnableSensubility: true
  CollectdSensubilityTransport: amqp1
```



```

# enable collection of containerized service metrics
CollectdEnableLibpodstats: true

# set collectd overrides for higher telemetry resolution and extra plugins
# to load
CollectdConnectionType: amqp1
CollectdAmqpInterval: 5
CollectdDefaultPollingInterval: 5
CollectdExtraPlugins:
- vmem

# set standard prefixes for where metrics and events are published to QDR
MetricsQdrAddresses:
- prefix: 'collectd'
  distribution: multicast
- prefix: 'anycast/ceilometer'
  distribution: multicast

ExtraConfig:
  ceilometer::agent::polling::polling_interval: 30
  ceilometer::agent::polling::polling_meters:
  - cpu
  - disk.*
  - ip.*
  - image.*
  - memory
  - memory.*
  - network.services.vpn.*
  - network.services.firewall.*
  - perf.*
  - port
  - port.*
  - switch
  - switch.*
  - storage.*
  - volume.*

# to avoid filling the memory buffers if disconnected from the message bus
# note: this may need an adjustment if there are many metrics to be sent.
collectd::plugin::amqp1::send_queue_limit: 5000

# receive extra information about virtual memory
collectd::plugin::vmem::verbose: true

# provide name and uuid in addition to hostname for better correlation
# to ceilometer data
collectd::plugin::virt::hostname_format: "name uuid hostname"

# provide the human-friendly name of the virtual instance
collectd::plugin::virt::plugin_instance_format: metadata

# set memcached collectd plugin to report its metrics by hostname
# rather than host IP, ensuring metrics in the dashboard remain uniform
collectd::plugin::memcached::instances:

```

```
local:
  host: "%{hiera('fqdn_canonical')}}"
  port: 11211
```

4.1.3. Configuring the STF connection for the overcloud

To configure the Service Telemetry Framework (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. The default configuration is for a single cloud instance with the default message bus topics. For configuration of multiple cloud deployments, see [Section 4.3, "Configuring multiple clouds"](#).

Prerequisites

- Retrieve the AMQ Interconnect route address. For more information, see [Section 4.1.1, "Retrieving the AMQ Interconnect route address"](#).

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Create a configuration file called **stf-connectors.yaml** in the **/home/stack** directory.
3. In the **stf-connectors.yaml** file, configure the **MetricsQdrConnectors** address to connect the AMQ Interconnect on the overcloud to the STF deployment. You configure the topic addresses for Sensubility, Ceilometer, and collectd in this file to match the defaults in STF. For more information about customizing topics and cloud configuration, see [Section 4.3, "Configuring multiple clouds"](#).

stf-connectors.yaml

```
resource_registry:
  OS::TripleO::Services::Collectd: /usr/share/openstack-tripleo-heat-
  templates/deployment/metrics/collectd-container-puppet.yaml

parameter_defaults:
  MetricsQdrConnectors:
    - host: default-interconnect-5671-service-telemetry.apps.infra.watch
      port: 443
      role: edge
      verifyHostname: false
      sslProfile: sslProfile

  MetricsQdrSSLProfiles:
    - name: sslProfile

  CeilometerQdrEventsConfig:
    driver: amqp
    topic: cloud1-event

  CeilometerQdrMetricsConfig:
    driver: amqp
    topic: cloud1-metering

  CollectdAmqpInstances:
```

```
cloud1-notify:
  notify: true
  format: JSON
  presettle: false
cloud1-telemetry:
  format: JSON
  presettle: false
```

CollectdSensubilityResultsChannel: sensubility/cloud1-telemetry

- The **resource_registry** configuration directly loads the collectd service because you do not include the **collectd-write-qdr.yaml** environment file for multiple cloud deployments.
- Replace the **host** parameter with the value that you retrieved in [Section 4.1.1, “Retrieving the AMQ Interconnect route address”](#).
- Replace the **host** sub-parameter of **MetricsQdrConnectors** with the value that you retrieved in [Section 4.1.1, “Retrieving the AMQ Interconnect route address”](#).
- Set **topic** value of **CeilometerQdrEventsConfig** to define the topic for Ceilometer events. The value is a unique topic identifier for the cloud such as **cloud1-event**.
- Set **topic** value of **CeilometerQdrMetricsConfig.topic** to define the topic for Ceilometer metrics. The value is a unique topic identifier for the cloud such as **cloud1-metering**.
- Set **CollectdAmqpInstances** sub-parameter to define the topic for collectd events. The section name is a unique topic identifier for the cloud such as **cloud1-notify**.
- Set **CollectdAmqpInstances** sub-parameter to define the topic for collectd metrics. The section name is a unique topic identifier for the cloud such as **cloud1-telemetry**.
- Set **CollectdSensubilityResultsChannel** to define the topic for collectd-sensubility events. The value is a unique topic identifier for the cloud such as **sensubility/cloud1-telemetry**.

NOTE

When you define the topics for collectd and Ceilometer, the value you provide is transposed into the full topic that the Smart Gateway client uses to listen for messages.

Ceilometer topic values are transposed into the topic address **anycast/ceilometer/<TOPIC>.sample** and collectd topic values are transposed into the topic address **collectd/<TOPIC>**. The value for sensubility is the full topic path and has no transposition from topic value to topic address.

For an example of a cloud configuration in the **ServiceTelemetry** object referring to the full topic address, see [the section called “The clouds parameter”](#).

4.1.4. Deploying the overcloud

Deploy or update the overcloud with the required environment files so that data is collected and transmitted to Service Telemetry Framework (STF).

Procedure

1. Log in to the undercloud host as the **stack** user.

2. Source the **stackrc** undercloud credentials file:

```
$ source ~/stackrc
```

3. Add your data collection and AMQ Interconnect environment files to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
-e [your environment files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/metrics/ceilometer-write-
qdr.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/metrics/qdr-edge-only.yaml \
-e /home/stack/enable-stf.yaml \
-e /home/stack/stf-connectors.yaml
```

- Include the **ceilometer-write-qdr.yaml** file to ensure that Ceilometer telemetry and events are sent to STF.
- Include the **qdr-edge-only.yaml** file to ensure that the message bus is enabled and connected to STF message bus routers.
- Include the **enable-stf.yaml** environment file to ensure that the defaults are configured correctly.
- Include the **stf-connectors.yaml** environment file to define the connection to STF.

4.1.5. Validating client-side installation

To validate data collection from the Service Telemetry Framework (STF) storage domain, query the data sources for delivered data. To validate individual nodes in the Red Hat OpenStack Platform (RHOSP) deployment, use SSH to connect to the console.

TIP

Some telemetry data is available only when RHOSP has active workloads.

Procedure

1. Log in to an overcloud node, for example, controller-0.
2. Ensure that the **metrics_qdr** and collection agent containers are running on the node:

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



NOTE

Use this command on compute nodes:

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

- 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
}
```

- 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.

- To ensure that messages are 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 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.ISY6Mcicol4J2Kp 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 0
endpoint out 912  1835    local temp.9Ok2resl9tmt+CT 250 0 0 0 0
0 0 0 0 0 0 0
```

- To list the addresses from RHOSP nodes to STF, connect to Red Hat OpenShift Container Platform to retrieve 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
```

- Connect to the pod and list the known connections. In this example, there are three **edge** connections from the RHOSP nodes with connection **id** 22, 23, and 24:

```
$ 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 dlvr uptime

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

```

8. 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
fallback

=====
=====
mobile anycast/ceilometer/event.sample 0 balanced - 1 0 970 970
0 0
mobile anycast/ceilometer/metering.sample 0 balanced - 1 0 2,344,833
2,344,833 0 0
mobile collectd/notify 0 multicast - 1 0 70 70 0 0
mobile collectd/telemetry 0 multicast - 1 0 216,128,890 216,128,890
0 0

```

4.2. DISABLING RED HAT OPENSTACK PLATFORM SERVICES USED WITH SERVICE TELEMETRY FRAMEWORK

Disable the services used when deploying Red Hat OpenStack Platform (RHOSP) and connecting it to Service Telemetry Framework (STF). There is no removal of logs or generated configuration files as part of the disablement of the services.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Source the **stackrc** undercloud credentials file:

```
$ source ~/stackrc
```

3. Create the **disable-stf.yaml** environment file:

```
$ cat > ~/disable-stf.yaml <<EOF
---
resource_registry:
  OS::TripleO::Services::CeilometerAgentCentral: OS::Heat::None
  OS::TripleO::Services::CeilometerAgentNotification: OS::Heat::None
  OS::TripleO::Services::CeilometerAgentIpmi: OS::Heat::None
  OS::TripleO::Services::ComputeCeilometerAgent: OS::Heat::None
  OS::TripleO::Services::Redis: OS::Heat::None
  OS::TripleO::Services::Collectd: OS::Heat::None
  OS::TripleO::Services::MetricsQdr: OS::Heat::None
EOF
```

4. Remove the following files from your RHOSP director deployment:

- **ceilometer-write-qdr.yaml**
- **qdr-edge-only.yaml**
- **enable-stf.yaml**
- **stf-connectors.yaml**

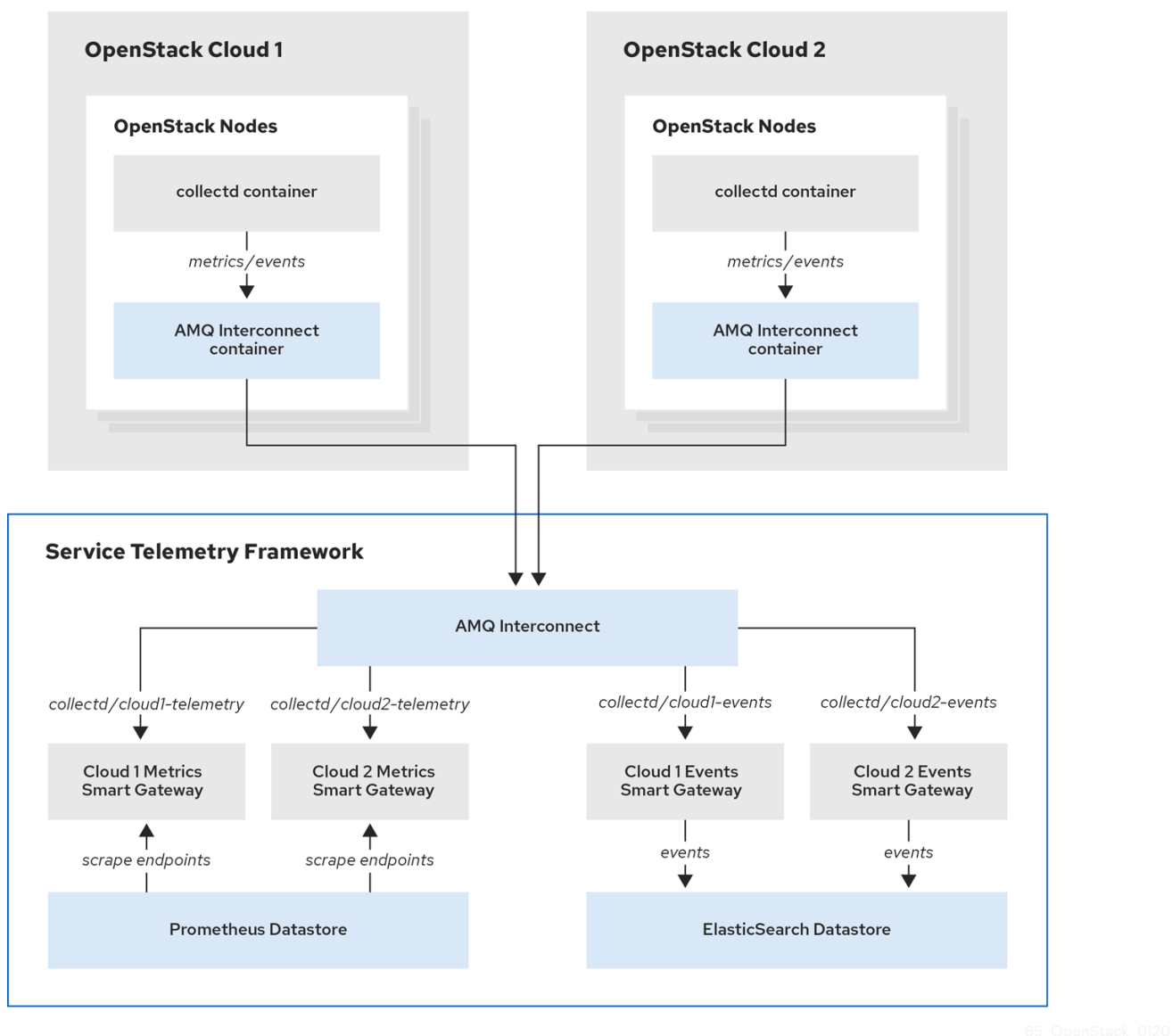
5. Update the RHOSP overcloud. Ensure that you use the **disable-stf.yaml** file early in the list of environment files. By adding **disable-stf.yaml** early in the list, other environment files can override the configuration that would disable the service:

```
(undercloud)$ openstack overcloud deploy --templates \
-e /home/stack/disable-stf.yaml \
-e [your environment files]
```

4.3. CONFIGURING MULTIPLE CLOUDS

You can configure multiple Red Hat OpenStack Platform (RHOSP) clouds to target a single instance of Service Telemetry Framework (STF). When you configure multiple clouds, every cloud must send metrics and events on their own unique message bus topic. In the STF deployment, Smart Gateway instances listen on these topics to save information to the common data store. Data that is stored by the Smart Gateway in the data storage domain is filtered by using the metadata that each of Smart Gateways creates.

Figure 4.1. Two RHOSP clouds connect to STF



65_OpenStack_0120

To configure the RHOSP overcloud for a multiple cloud scenario, complete the following tasks:

1. Plan the AMQP address prefixes that you want to use for each cloud. For more information, see [Section 4.3.1, "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 4.3.2, "Deploying Smart Gateways"](#).
3. Configure each cloud with a unique domain name. For more information, see [Section 4.3.4, "Setting a unique cloud domain"](#).
4. Create the base configuration for STF. For more information, see [Section 4.1.2, "Creating the base configuration for STF"](#).
5. Configure each cloud to send its metrics and events to STF on the correct address. For more information, see [Section 4.3.5, "Creating the Red Hat OpenStack Platform environment file for multiple clouds"](#).

4.3.1. Planning AMQP address prefixes

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

- **collectd/cloud1-telemetry**
- **collectd/cloud1-notify**
- **sensubility/cloud1-telemetry**
- **anycast/ceilometer/cloud1-metering.sample**
- **anycast/ceilometer/cloud1-event.sample**
- **collectd/cloud2-telemetry**
- **collectd/cloud2-notify**
- **sensubility/cloud2-telemetry**
- **anycast/ceilometer/cloud2-metering.sample**
- **anycast/ceilometer/cloud2-event.sample**
- **collectd/us-east-1-telemetry**
- **collectd/us-west-3-telemetry**

4.3.2. 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, one for Ceilometer events, and one for collectd-sensubility metrics. Configure each of the Smart Gateways to listen on the AMQP address that you define for the corresponding cloud. To define Smart Gateways, configure 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 you deploy 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 in **cloud1** with the following subscription addresses:

collector	type	default subscription address
collectd	metrics	collectd/telemetry
collectd	events	collectd/notify
collectd-sensubility	metrics	sensubility/telemetry

Ceilometer	metrics	anycast/ceilometer/metering.sample
Ceilometer	events	anycast/ceilometer/event.sample

Prerequisites

- You have determined your cloud naming scheme. For more information about determining your naming scheme, see [Section 4.3.1, “Planning AMQP address prefixes”](#).
- You have created your list of clouds objects. For more information about creating the content for the **clouds** parameter, see [the section called “The clouds parameter”](#).

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 a **clouds** parameter with your configuration:



WARNING

Long cloud names might exceed the maximum pod name of 63 characters. Ensure that the combination of the **ServiceTelemetry** name **default** and the **clouds.name** does not exceed 19 characters. Cloud names cannot contain any special characters, such as -. Limit cloud names to alphanumeric (a-z, 0-9).

Topic addresses have no character limitation and can be different from the **clouds.name** value.

```
$ 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: sensubility
      subscriptionAddress: sensubility/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  2/2   Running 0      13h
default-cloud1-ceil-meter-smartgateway-58f885c76d-xmwn  2/2   Running 0      13h
default-cloud1-coll-event-smartgateway-58fbbd4485-rl9bd  2/2   Running 0      13h
default-cloud1-coll-meter-smartgateway-7c6fc495c4-jn728  2/2   Running 0      13h
default-cloud1-sens-meter-smartgateway-8h4tc445a2-mm683  2/2   Running 0      13h

```

4.3.3. Deleting the default Smart Gateways

After you configure Service Telemetry Framework (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 were created but are no longer listed in the ServiceTelemetry **clouds** list of objects. To enable the removal of SmartGateway objects that are not defined by the **clouds** parameter, you must set the **cloudsRemoveOnMissing** parameter to **true** in the **ServiceTelemetry** manifest.

TIP

If you do not want to deploy any Smart Gateways, define an empty clouds list by 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 that you want the Service Telemetry Operator to manage. For more information, see [the section called “The clouds parameter”](#).
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
```

4.3.4. Setting a unique cloud domain

To ensure that AMQ Interconnect router connections from Red Hat OpenStack Platform (RHOSP) to Service Telemetry Framework (STF) are unique and do not conflict, configure the **CloudDomain** parameter.



WARNING

Ensure that you do not change host or domain names in an existing deployment. Host and domain name configuration is supported in new cloud deployments only.

Procedure

1. Create a new environment file, for example, **hostnames.yaml**.
2. Set the **CloudDomain** parameter in the environment file, as shown in the following example:

hostnames.yaml

```
parameter_defaults:
  CloudDomain: newyork-west-04
  CephStorageHostnameFormat: 'ceph-%index%'
  ObjectStorageHostnameFormat: 'swift-%index%'
  ComputeHostnameFormat: 'compute-%index%'
```

3. Add the new environment file to your deployment.

Additional resources

Additional resources

- [Section 4.3.5, “Creating the Red Hat OpenStack Platform environment file for multiple clouds”](#)
- [Core Overcloud Parameters](#) in the *Overcloud Parameters* guide

4.3.5. Creating the Red Hat OpenStack Platform environment file for multiple clouds

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



NOTE

If you enabled container health and API status monitoring, you must also modify the **CollectdSensubilityResultsChannel** parameter. For more information, see [Section 5.8, “Red Hat OpenStack Platform API status and containerized services health”](#).

Prerequisites

- You have created your list of clouds objects. For more information about creating the content for the clouds parameter, see the [clouds configuration parameter](#).
- You have retrieved the AMQ Interconnect route address. For more information, see [Section 4.1.1, “Retrieving the AMQ Interconnect route address”](#).
- You have created the base configuration for STF. For more information, see [Section 4.1.2, “Creating the base configuration for STF”](#).
- You have created a unique domain name environment file. For more information, see [Section 4.3.4, “Setting a unique cloud domain”](#).

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Create a configuration file called **stf-connectors.yaml** in the **/home/stack** directory.
3. In the **stf-connectors.yaml** file, configure the **MetricsQdrConnectors** address to connect to the AMQ Interconnect on the overcloud deployment. Configure the **CeilometerQdrEventsConfig**, **CeilometerQdrMetricsConfig**, **CollectdAmqpInstances**, and **CollectdSensubilityResultsChannel** topic values 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

parameter_defaults:
  MetricsQdrConnectors:
    - host: default-interconnect-5671-service-telemetry.apps.infra.watch
      port: 443
```

```

    role: edge
    verifyHostname: false
    sslProfile: sslProfile

MetricsQdrSSLProfiles:
  - name: sslProfile

CeilometerQdrEventsConfig:
  driver: amqp
  topic: cloud1-event

CeilometerQdrMetricsConfig:
  driver: amqp
  topic: cloud1-metering

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

CollectdSensubilityResultsChannel: sensubility/cloud1-telemetry

```

- The **resource_registry** configuration directly loads the collectd service because you do not include the **collectd-write-qdr.yaml** environment file for multiple cloud deployments.
- Replace the **host** parameter with the value that you retrieved in [Section 4.1.1, “Retrieving the AMQ Interconnect route address”](#).
- Replace the **host** sub-parameter of **MetricsQdrConnectors** with the value that you retrieved in [Section 4.1.1, “Retrieving the AMQ Interconnect route address”](#).
- Set **topic** value of **CeilometerQdrEventsConfig** to define the topic for Ceilometer events. The value is a unique topic identifier for the cloud such as **cloud1-event**.
- Set **topic** value of **CeilometerQdrMetricsConfig.topic** to define the topic for Ceilometer metrics. The value is a unique topic identifier for the cloud such as **cloud1-metering**.
- Set **CollectdAmqpInstances** sub-paramter to define the topic for collectd events. The section name is a unique topic identifier for the cloud such as **cloud1-notify**.
- Set **CollectdAmqpInstances** sub-parameter to define the topic for collectd metrics. The section name is a unique topic identifier for the cloud such as **cloud1-telemetry**.
- Set **CollectdSensubilityResultsChannel** to define the topic for collectd-sensubility events. The value is a unique topic identifier for the cloud such as **sensubility/cloud1-telemetry**.



NOTE

When you define the topics for `collectd` and `Ceilometer`, the value you provide is transposed into the full topic that the Smart Gateway client uses to listen for messages.

Ceilometer topic values are transposed into the topic address **anycast/ceilometer/<TOPIC>.sample** and `collectd` topic values are transposed into the topic address **collectd/<TOPIC>**. The value for sensubility is the full topic path and has no transposition from topic value to topic address.

For an example of a cloud configuration in the **ServiceTelemetry** object referring to the full topic address, see [the section called "The clouds parameter"](#).

4. Ensure that the naming convention in the **stf-connectors.yaml** file aligns with the **spec.bridge.amqpUrl** field in the Smart Gateway configuration. For example, configure the **CeilometerQdrEventsConfig.topic** field to a value of **cloud1-event**.
5. Log in to the undercloud host as the **stack** user.
6. Source the **stackrc** undercloud credentials file:

```
$ source stackrc
```

7. Include the **stf-connectors.yaml** file and unique domain name environment file **hostnames.yaml** in the **openstack overcloud deployment** command, with any other environment files relevant to your environment:



WARNING

If you use the **collectd-write-qdr.yaml** file with a custom **CollectdAmqpInstances** parameter, data publishes to the custom and default topics. In a multiple cloud environment, the configuration of the **resource_registry** parameter in the **stf-connectors.yaml** file loads the `collectd` service.

```
(undercloud)$ openstack overcloud deploy --templates \
-e [your environment files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/metrics/ceilometer-write-qdr.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/metrics/qdr-edge-only.yaml \
-e /home/stack/hostnames.yaml \
-e /home/stack/enable-stf.yaml \
-e /home/stack/stf-connectors.yaml
```

8. Deploy the Red Hat OpenStack Platform overcloud.

Additional resources

- For information about how to validate the deployment, see [Section 4.1.5, “Validating client-side installation”](#).

4.3.6. Querying metrics data from multiple clouds

Data stored in Prometheus has a **service** label 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"}**.

CHAPTER 5. USING OPERATIONAL FEATURES OF SERVICE TELEMETRY FRAMEWORK

You can use the following operational features to provide additional functionality to the Service Telemetry Framework (STF):

- [Configuring dashboards](#)
- [Configuring the metrics retention time period](#)
- [Configuring alerts](#)
- [Configuring SNMP traps](#)
- [Configuring high availability](#)
- [Configuring an alternate observability strategy](#)
- [Monitoring the resource use of OpenStack services](#)
- [Monitoring container health and API status](#)

5.1. DASHBOARDS IN SERVICE TELEMETRY FRAMEWORK

Use the third-party application, Grafana, to visualize system-level metrics that the data collectors collectd and Ceilometer gather for each individual host node.

For more information about configuring data collectors, see [Section 4.1, “Deploying Red Hat OpenStack Platform overcloud for Service Telemetry Framework using director”](#).

You can use dashboards to monitor a cloud:

Infrastructure dashboard

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

Cloud view dashboard

Use the cloud view dashboard to view panels to monitor service resource usage, API stats, and cloud events. You must enable API health monitoring and service monitoring to provide the data for this dashboard. API health monitoring is enabled by default in the STF base configuration. For more information, see [Section 4.1.2, “Creating the base configuration for STF”](#).

- For more information about API health monitoring, see [Section 5.8, “Red Hat OpenStack Platform API status and containerized services health”](#).
- For more information about RHOSP service monitoring, see [Section 5.7, “Resource usage of Red Hat OpenStack Platform services”](#).

Virtual machine view dashboard

Use the virtual machine view dashboard to view panels to monitor virtual machine infrastructure usage. Select a cloud and project from the upper left corner of the dashboard. You must enable event storage if you want to enable the event annotations on this dashboard. For more information, see [Section 3.2, “Creating a ServiceTelemetry object in Red Hat OpenShift Container Platform”](#).

Memcached view dashboard

Use the memcached view dashboard to view panels to monitor connections, availability, system metrics and cache performance. Select a cloud from the upper left corner of the dashboard.

5.1.1. Configuring 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 community-operators CatalogSource. If you use the Service Telemetry Operator to deploy Grafana, it results in a Grafana instance and the configuration of the default data sources for the local STF deployment.

Procedure

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

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

3. Subscribe to the Grafana Operator by using the community-operators CatalogSource:



WARNING

Community Operators are Operators which have not been vetted or verified by Red Hat. Community Operators should be used with caution because their stability is unknown. Red Hat provides no support for community Operators.

[Learn more about Red Hat's third party software support policy](#)

```
$ oc apply -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: grafana-operator
  namespace: service-telemetry
spec:
  channel: v4
  installPlanApproval: Automatic
  name: grafana-operator
  source: community-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. Verify that the Operator launched successfully. In the command output, if the value of the **PHASE** column is **Succeeded**, the Operator launched successfully:

```
$ oc get csv --selector operators.coreos.com/grafana-operator.service-telemetry
```

NAME	DISPLAY	VERSION	REPLACES	PHASE
------	---------	---------	----------	-------

```
grafana-operator.v4.10.1 Grafana Operator 4.10.1 grafana-operator.v4.10.0 Succeeded
```

- To launch a Grafana instance, create or modify the **ServiceTelemetry** object. Set **graphing.enabled** and **graphing.grafana.ingressEnabled** to **true**. Optionally, set the value of **graphing.grafana.baseImage** to the Grafana workload container image that will be deployed:

```
$ oc edit stf default

apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
...
spec:
  ...
  graphing:
    enabled: true
  grafana:
    ingressEnabled: true
    baseImage: 'registry.redhat.io/rhel8/grafana:7'
```

- 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

- Verify that the Grafana data sources installed correctly:

```
$ oc get grafanadatasources
```

NAME	AGE
default-datasources	20h

- Verify that the Grafana route exists:

```
$ oc get route grafana-route
```

NAME	HOST/PORT	PATH	SERVICES	PORT
grafana-route	grafana-route-service-telemetry.apps.infra.watch		grafana-service	3000
edge	None			

5.1.2. Overriding the default Grafana container image

The dashboards in Service Telemetry Framework (STF) require features that are available only in Grafana version 8.1.0 and later. By default, the Service Telemetry Operator installs a compatible version. You can override the base Grafana image by specifying the image path to an image registry with **graphing.grafana.baseImage**.

Procedure

- Ensure that you have the correct version of Grafana:

```
$ oc get pod -l "app=grafana" -jsonpath='{.items[0].spec.containers[0].image}'
docker.io/grafana/grafana:7.3.10
```

- If the running image is older than 8.1.0, patch the ServiceTelemetry object to update the image. Service Telemetry Operator updates the Grafana manifest, which restarts the Grafana deployment:

```
$ oc patch stf/default --type merge -p '{"spec":{"graphing":{"grafana":{"baseImage":"docker.io/grafana/grafana:8.1.5"}}}}'
```

- Verify that a new Grafana pod exists and has a **STATUS** value of **Running**:

```
$ oc get pod -l "app=grafana"
NAME                                READY   STATUS    RESTARTS   AGE
grafana-deployment-fb9799b58-j2hj2  1/1     Running   0           10s
```

- Verify that the new instance is running the updated image:

```
$ oc get pod -l "app=grafana" -jsonpath='{.items[0].spec.containers[0].image}'
docker.io/grafana/grafana:8.1.0
```

5.1.3. 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

- Import the infrastructure dashboard:

```
$ oc apply -f https://raw.githubusercontent.com/infrawatch/dashboards/master/deploy/stf-1/rhos-dashboard.yaml

grafanadashboard.integreatly.org/rhos-dashboard-1 created
```

- Import the cloud dashboard:



WARNING

In the **stf-connectors.yaml** file, ensure you set the value of the collectd **virt** plugin parameter **hostname_format** to **name uuid hostname**, otherwise some of the panels on the cloud dashboard display no information. For more information about the **virt** plugin, see [collectd plugins](#).

```
$ oc apply -f https://raw.githubusercontent.com/infrawatch/dashboards/master/deploy/stf-1/rhos-cloud-dashboard.yaml

grafanadashboard.integreatly.org/rhos-cloud-dashboard-1 created
```

3. Import the cloud events dashboard:

```
$ oc apply -f https://raw.githubusercontent.com/infrawatch/dashboards/master/deploy/stf-1/rhos-cloudevents-dashboard.yaml

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

4. Import the virtual machine dashboard:

```
$ oc apply -f https://raw.githubusercontent.com/infrawatch/dashboards/master/deploy/stf-1/virtual-machine-view.yaml

grafanadashboard.integreatly.org/virtual-machine-view-1 configured
```

5. Import the memcached dashboard:

```
$ oc apply -f https://raw.githubusercontent.com/infrawatch/dashboards/master/deploy/stf-1/memcached-dashboard.yaml

grafanadashboard.integreatly.org/memcached-dashboard-1 created
```

6. Verify that the dashboards are available:

```
$ oc get grafanadashboards

NAME                      AGE
memcached-dashboard-1     7s
rhos-cloud-dashboard-1    23s
rhos-cloudevents-dashboard 18s
rhos-dashboard-1          29s
virtual-machine-view-1    13s
```

7. Retrieve the Grafana route address:

```
$ oc get route grafana-route -ojsonpath='{.spec.host}'

grafana-route-service-telemetry.apps.infra.watch
```

8. In a web browser, navigate to `https://<grafana_route_address>`. Replace `<grafana_route_address>` with the value that you retrieved in the previous step.
9. To view the dashboard, click **Dashboards** and **Manage**.

5.1.4. Retrieving and setting Grafana login credentials

When Grafana is enabled, you can login using openshift authentication, or the default username and password set by the Grafana Operator.

You can override the credentials in the **ServiceTelemetry** object to have Service Telemetry Framework (STF) set the username and password for Grafana instead.

Procedure

1. Log in to Red Hat OpenShift Container Platform.

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

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

3. Retrieve the existing username and password from the STF object:

```
$ oc get stf default -o jsonpath="{.spec.graphing.grafana['adminUser','adminPassword']}"
```

4. To modify the default values of the Grafana administrator username and password through the ServiceTelemetry object, use the **graphing.grafana.adminUser** and **graphing.grafana.adminPassword** parameters.

```
$ oc edit stf default
```

5. Wait for the grafana pod to restart with the new credentials in place

```
$ oc get po -l app=grafana -w
```

5.2. METRICS RETENTION TIME PERIOD IN SERVICE TELEMETRY FRAMEWORK

The default retention time for metrics stored in Service Telemetry Framework (STF) is 24 hours, which provides enough data for trends to develop for the purposes of alerting.

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

Additional resources

- To adjust STF for additional metrics retention time, see [Section 5.2.1, “Editing the metrics retention time period in Service Telemetry Framework”](#).
- For recommendations about Prometheus data storage and estimating storage space, see <https://prometheus.io/docs/prometheus/latest/storage/#operational-aspects>
- For more information about Thanos, see <https://thanos.io/>

5.2.1. Editing the metrics retention time period in Service Telemetry Framework

You can adjust Service Telemetry Framework (STF) for additional metrics retention time.

Procedure

1. Log in to Red Hat OpenShift Container Platform.

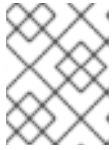
2. Change to the service-telemetry namespace:

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

3. Edit the ServiceTelemetry object:

```
$ oc edit stf default
```

4. Add **retention: 7d** to the storage section of `backends.metrics.prometheus.storage` to increase the retention period to seven days:



NOTE

If you set a long retention period, retrieving data from heavily populated Prometheus systems can result in queries returning results slowly.

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
spec:
  ...
  backends:
    metrics:
      prometheus:
        enabled: true
        storage:
          strategy: persistent
          retention: 7d
  ...
```

5. Save your changes and close the object.
6. Wait for prometheus to restart with the new settings.

```
$ oc get po -l app.kubernetes.io/name=prometheus -w
```

7. Verify the new retention setting by checking the command line arguments used in the pod.

```
$ oc describe po prometheus-default-0 | grep retention.time
--storage.tsdb.retention.time=24h
```

Additional resources

- For more information about the metrics retention time, see [Section 5.2, “Metrics retention time period in Service Telemetry Framework”](#).

5.3. ALERTS IN SERVICE TELEMETRY FRAMEWORK

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 by using email, on-call notification systems, or chat platforms.

To create an alert, complete the following tasks:

1. Create an alert rule in Prometheus. For more information, see [Section 5.3.1, “Creating an alert rule in Prometheus”](#).

2. Create an alert route in Alertmanager. There are two ways in which you can create an alert route:

- [Creating a standard alert route in Alertmanager](#) .
- [Creating an alert route with templating in Alertmanager](#) .

Additional resources

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

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

5.3.1. 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: Collectd metrics receive rate is zero
      expr: rate(sg_total_collectd_msg_received_count[1m]) == 0
EOF
```

To change the rule, edit the value of the **expr** parameter.

4. To verify that the Operator loaded the rules into Prometheus, run the **curl** command against the default-prometheus-proxy route with basic authentication:

```
$ curl -k --user "internal:$(oc get secret default-prometheus-htpasswd -o-template='{{
.data.password | base64decode }}')" https://$(oc get route default-prometheus-proxy -o-
template='{{ .spec.host }}')/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": [
          {
            "state": "inactive",
            "name": "Collectd metrics receive count is zero",
            "query": "rate(sg_total_collectd_msg_received_count[1m]) == 0",
            "duration": 0,
            "labels": {},
            "annotations": {},
            "alerts": [
              {
                "health": "ok",
                "evaluationTime": "0.00034627",
                "lastEvaluation": "2021-12-07T17:23:22.160448028Z",
                "type": "alerting"
              }
            ],
            "interval": 30,
            "evaluationTime": "0.000353787",
            "lastEvaluation": "2021-12-07T17:23:22.160444017Z"
          }
        ]
      }
    ]
  }
}
```

Additional resources

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

5.3.2. Configuring custom alerts

You can add custom alerts to the **PrometheusRule** object that you created in [Section 5.3.1, “Creating an alert rule in Prometheus”](#).

Procedure

1. Use the **oc edit** command:

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

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

Additional resources

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

5.3.3. Creating a standard 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 a Red Hat OpenShift Container Platform secret. By default, Service Telemetry Framework (STF) 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, you must add a **alertmanagerConfigManifest** parameter to the Service Telemetry Operator that results in an updated secret, managed by the Prometheus Operator.



NOTE

If your **alertmanagerConfigManifest** contains a custom template, for example, to construct the title and text of the sent alert, you must deploy the contents of the **alertmanagerConfigManifest** using a base64-encoded configuration. For more information, see [Section 5.3.4, "Creating an alert route with templating in Alertmanager"](#).

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 the new parameter **alertmanagerConfigManifest** and the **Secret** object contents to define the **alertmanager.yaml** configuration for Alertmanager:



NOTE

This step loads the default template that the Service Telemetry Operator manages. 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 of the parameter **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 has been 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'

```

6. Run the **wget** command from the prometheus pod against the **alertmanager-proxy** service to retrieve the status and **configYAML** contents, and verify that the supplied configuration matches the configuration in Alertmanager:

```

$ oc exec -it prometheus-default-0 -c prometheus -- sh -c "wget --header \"Authorization:
Bearer \$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)\" https://default-
alertmanager-proxy:9095/api/v1/status -q -O -"

{"status":"success","data":{"configYAML":"...",...}}

```

7. Verify that the **configYAML** field contains the changes you expect.
8. 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 [Prometheus user guide on alerting](#).

5.3.4. Creating an alert route with templating 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 a Red Hat OpenShift Container Platform secret. By default, Service Telemetry Framework (STF) 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'
```

If the **alertmanagerConfigManifest** parameter contains a custom template, for example, to construct the title and text of the sent alert, you must deploy the contents of the **alertmanagerConfigManifest** by using a base64-encoded configuration.

Procedure

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

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

3. Create the necessary alertmanager config in a file called `alertmanager.yaml`, for example:

```
$ cat > alertmanager.yaml <<EOF
global:
  resolve_timeout: 10m
  slack_api_url: <slack_api_url>
receivers:
  - name: slack
    slack_configs:
      - channel: #stf-alerts
        title: |-
          ...
        text: >-
          ...
route:
  group_by: ['job']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: 'slack'
EOF
```

4. Generate the config manifest and add it to the **ServiceTelemetry** object for your STF deployment:

```
$ CONFIG_MANIFEST=$(oc create secret --dry-run=client generic alertmanager-default --
from-file=alertmanager.yaml -o json)
$ oc patch stf default --type=merge -p '{"spec":
{"alertmanagerConfigManifest":"$CONFIG_MANIFEST"}'
```

5. Verify that the configuration has been applied to the secret:



NOTE

There will be a short delay as the operators update each object

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

```
global:
  resolve_timeout: 10m
  slack_api_url: <slack_api_url>
receivers:
- name: slack
  slack_configs:
  - channel: #stf-alerts
    title: |-
      ...
    text: >-
      ...
route:
  group_by: ['job']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: 'slack'
```

6. Run the **wget** command from the prometheus pod against the **alertmanager-proxy** service to retrieve the status and **configYAML** contents, and verify that the supplied configuration matches the configuration in Alertmanager:

```
$ oc exec -it prometheus-default-0 -c prometheus -- /bin/sh -c "wget --header \"Authorization:
Bearer \$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)\" https://default-
alertmanager-proxy:9095/api/v1/status -q -O -"

{"status":"success","data":{"configYAML":"...","..."/>
```

7. Verify that the **configYAML** field contains the changes you expect.

Additional resources

- For more information about the Red Hat OpenShift Container Platform secret and the Prometheus operator, see [Prometheus user guide on alerting](#).

5.4. SENDING ALERTS AS SNMP TRAPS

To enable SNMP traps, modify the **ServiceTelemetry** object and configure the **snmpTraps** parameters. SNMP traps are sent using version 2c.

5.4.1. Configuration parameters for snmpTraps

The **snmpTraps** parameter contains the following sub-parameters for configuring the alert receiver:

enabled

Set the value of this sub-parameter to true to enable the SNMP trap alert receiver. The default value is false.

target

Target address to send SNMP traps. Value is a string. Default is **192.168.24.254**.

port

Target port to send SNMP traps. Value is an integer. Default is **162**.

community

Target community to send SNMP traps to. Value is a string. Default is **public**.

retries

SNMP trap retry delivery limit. Value is an integer. Default is **5**.

timeout

SNMP trap delivery timeout defined in seconds. Value is an integer. Default is **1**.

alertOidLabel

Label name in the alert that defines the OID value to send the SNMP trap as. Value is a string. Default is **oid**.

trapOidPrefix

SNMP trap OID prefix for variable bindings. Value is a string. Default is **1.3.6.1.4.1.50495.15**.

trapDefaultOid

SNMP trap OID when no alert OID label has been specified with the alert. Value is a string. Default is **1.3.6.1.4.1.50495.15.1.2.1**.

trapDefaultSeverity

SNMP trap severity when no alert severity has been set. Value is a string. Defaults to an empty string.

Configure the **snmpTraps** parameter as part of the **alerting.alertmanager.receivers** definition in the **ServiceTelemetry** object:

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
spec:
  alerting:
    alertmanager:
      receivers:
        snmpTraps:
          alertOidLabel: oid
          community: public
          enabled: true
          port: 162
          retries: 5
          target: 192.168.25.254
          timeout: 1
          trapDefaultOid: 1.3.6.1.4.1.50495.15.1.2.1
```

```
trapDefaultSeverity: ""
trapOidPrefix: 1.3.6.1.4.1.50495.15
```

```
...
```

5.4.2. Overview of the MIB definition

Delivery of SNMP traps uses object identifier (OID) value **1.3.6.1.4.1.50495.15.1.2.1** by default. The management information base (MIB) schema is available at <https://github.com/infrawatch/prometheus-webhook-snmp/blob/master/PROMETHEUS-ALERT-CEPH-MIB.txt>.

The OID number is comprised of the following component values: * The value **1.3.6.1.4.1** is a global OID defined for private enterprises. * The next identifier **50495** is a private enterprise number assigned by IANA for the Ceph organization. * The other values are child OIDs of the parent.

15

prometheus objects

15.1

prometheus alerts

15.1.2

prometheus alert traps

15.1.2.1

prometheus alert trap default

The prometheus alert trap default is an object comprised of several other sub-objects to OID **1.3.6.1.4.1.50495.15** which is defined by the **alerting.alertmanager.receivers.snmpTraps.trapOidPrefix** parameter:

<trapOidPrefix>.1.1.1

alert name

<trapOidPrefix>.1.1.2

status

<trapOidPrefix>.1.1.3

severity

<trapOidPrefix>.1.1.4

instance

<trapOidPrefix>.1.1.5

job

<trapOidPrefix>.1.1.6

description

<trapOidPrefix>.1.1.7

labels

<trapOidPrefix>.1.1.8

timestamp

<trapOidPrefix>.1.1.9

rawdata

The following is example output from a simple SNMP trap receiver that outputs the received trap to the console:

```
SNMPv2-MIB::snmpTrapOID.0 = OID: SNMPv2-SMI::enterprises.50495.15.1.2.1
SNMPv2-SMI::enterprises.50495.15.1.1.1 = STRING: "TEST ALERT FROM PROMETHEUS
PLEASE ACKNOWLEDGE"
SNMPv2-SMI::enterprises.50495.15.1.1.2 = STRING: "firing"
SNMPv2-SMI::enterprises.50495.15.1.1.3 = STRING: "warning"
SNMPv2-SMI::enterprises.50495.15.1.1.4 = ""
SNMPv2-SMI::enterprises.50495.15.1.1.5 = ""
SNMPv2-SMI::enterprises.50495.15.1.1.6 = STRING: "TEST ALERT FROM "
SNMPv2-SMI::enterprises.50495.15.1.1.7 = STRING: "{\"cluster\": \"TEST\", \"container\": \"sg-
core\", \"endpoint\": \"prom-https\", \"prometheus\": \"service-telemetry/default\", \"service\": \"default-
cloud1-coll-meter\", \"source\": \"SG\"}"
SNMPv2-SMI::enterprises.50495.15.1.1.8 = Timeticks: (1676476389) 194 days, 0:52:43.89
SNMPv2-SMI::enterprises.50495.15.1.1.9 = STRING: "{\"status\": \"firing\", \"labels\": {\"cluster\":
\"TEST\", \"container\": \"sg-core\", \"endpoint\": \"prom-https\", \"prometheus\": \"service-
telemetry/default\", \"service\": \"default-cloud1-coll-meter\", \"source\": \"SG\"}, \"annotations\":
{\"action\": \"TESTING PLEASE ACKNOWLEDGE, NO FURTHER ACTION REQUIRED ONLY A
TEST\", \"startsAt\": \"2023-02-15T15:53:09.109Z\", \"endsAt\": \"0001-01-01T00:00:00Z\",
\"generatorURL\": \"http://prometheus-default-0:9090/graph?
g0.expr=sg_total_collectd_msg_received_count+%3E+1&g0.tab=1\", \"fingerprint\":
\"feefeb77c577a02f\"}"
```

5.4.3. Configuring SNMP traps

Prerequisites

- Ensure that you know the IP address or hostname of the SNMP trap receiver where you want to send the alerts to.

Procedure

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

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

3. To enable SNMP traps, modify the **ServiceTelemetry** object:

```
$ oc edit stf default
```

4. 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
```

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

Additional Information

For more information about available parameters for **snmpTraps**, see [Section 5.4.1, "Configuration parameters for snmpTraps"](#).

5.4.4. Creating alerts for SNMP traps

You can create alerts that are configured for delivery by SNMP traps by adding labels that are parsed by the prometheus-webhook-snmp middleware to define the trap information and delivered object identifiers (OID). Adding the **oid** or **severity** labels is only required if you need to change the default values for a particular alert definition.

NOTE

When you set the oid label, the top-level SNMP trap OID changes, but the sub-OIDs remain defined by the global **trapOidPrefix** value plus the child OID values **.1.1.1** through **.1.1.9**. For more information about the MIB definition, see [Section 5.4.2, "Overview of the MIB definition"](#).

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 and an **oid** label that contains the SNMP trap OID override value:

```
$ oc apply -f - <<EOF
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  creationTimestamp: null
  labels:
    prometheus: default
    role: alert-rules
  name: prometheus-alarm-rules-snmp
  namespace: service-telemetry
spec:
  groups:
    - name: ./openstack.rules
      rules:
        - alert: Collectd metrics receive rate is zero
          expr: rate(sg_total_collectd_msg_received_count[1m]) == 0
          labels:
            oid: 1.3.6.1.4.1.50495.15.1.2.1
            severity: critical
EOF
```

Additional information

For more information about configuring alerts, see [Section 5.3, “Alerts in Service Telemetry Framework”](#).

5.5. HIGH AVAILABILITY

With high availability, Service Telemetry Framework (STF) can rapidly recover from failures in its component services. Although Red Hat OpenShift Container Platform 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, which reduces recovery time to approximately 2 seconds. To protect against failure of an Red Hat OpenShift Container Platform node, deploy STF to an Red Hat OpenShift Container Platform cluster with three or more nodes.



WARNING

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
 - Metrics Smart Gateway
- Recovery time from a lost pod in any of these services reduces to approximately 2 seconds.

5.5.1. Configuring high availability

To configure Service Telemetry Framework (STF) for high availability, add **highAvailability.enabled: true** to the ServiceTelemetry object in Red Hat OpenShift Container Platform. 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.

5.6. OBSERVABILITY STRATEGY IN SERVICE TELEMETRY FRAMEWORK

Service Telemetry Framework (STF) does not include storage backends and alerting tools. STF uses community operators to deploy Prometheus, Alertmanager, Grafana, and Elasticsearch. STF makes requests to these community operators to create instances of each application configured to work with STF.

Instead of having Service Telemetry Operator create custom resource requests, you can use your own deployments of these applications or other compatible applications, and scrape the metrics Smart Gateways for delivery to your own Prometheus-compatible system for telemetry storage. If you set the **`observabilityStrategy`** to **`none`**, then storage backends will not be deployed so persistent storage will not be required by STF.

5.6.1. Configuring an alternate observability strategy

To configure STF to skip the deployment of storage, visualization, and alerting backends, add **`observabilityStrategy: none`** to the `ServiceTelemetry` spec. In this mode, only AMQ Interconnect routers and metrics Smart Gateways are deployed, and you must configure an external Prometheus-compatible system to collect metrics from the STF Smart Gateways.



NOTE

Currently, only metrics are supported when you set **`observabilityStrategy`** to **`none`**. Events Smart Gateways are not deployed.

Procedure

1. Create a **`ServiceTelemetry`** object with the property **`observabilityStrategy: none`** in the **`spec`** parameter. The manifest shows results in a default deployment of STF that is suitable for receiving telemetry from a single cloud with all metrics collector types.

```
$ oc apply -f - <<EOF
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
```

```
namespace: service-telemetry
spec:
  observabilityStrategy: none
EOF
```

2. Delete the left over objects that are managed by community operators

```
$ for o in alertmanager/default prometheus/default elasticsearch/elasticsearch
grafana/default; do oc delete $o; done
```

3. To verify that all workloads are operating correctly, view the pods and the status of each pod:

```
$ oc get pods
NAME                                READY STATUS RESTARTS AGE
default-cloud1-ceil-meter-smartgateway-59c845d65b-gzhcs 3/3 Running 0 132m
default-cloud1-coll-meter-smartgateway-75bbd948b9-d5phm 3/3 Running 0 132m
default-cloud1-sens-meter-smartgateway-7fdbb57b6d-dh2g9 3/3 Running 0 132m
default-interconnect-668d5bbcd6-57b2l 1/1 Running 0 132m
interconnect-operator-b8f5bb647-tlp5t 1/1 Running 0 47h
service-telemetry-operator-566b9dd695-wkvjq 1/1 Running 0 156m
smart-gateway-operator-58d77dcf7-6xsq7 1/1 Running 0 47h
```

Additional resources

For more information about configuring additional clouds or to change the set of supported collectors, see [Section 4.3.2, “Deploying Smart Gateways”](#)

5.7. RESOURCE USAGE OF RED HAT OPENSTACK PLATFORM SERVICES

You can monitor the resource usage of the Red Hat OpenStack Platform (RHOSP) services, such as the APIs and other infrastructure processes, to identify bottlenecks in the overcloud by showing services that run out of compute power. Resource usage monitoring is enabled by default.

Additional resources

- To disable resource usage monitoring, see [Section 5.7.1, “Disabling resource usage monitoring of Red Hat OpenStack Platform services”](#).

5.7.1. Disabling resource usage monitoring of Red Hat OpenStack Platform services

To disable the monitoring of RHOSP containerized service resource usage, you must set the **CollectdEnableLibpodstats** parameter to **false**.

Prerequisites

- You have created the **stf-connectors.yaml** file. For more information, see [Section 4.1, “Deploying Red Hat OpenStack Platform overcloud for Service Telemetry Framework using director”](#).
- You are using the most current version of Red Hat OpenStack Platform (RHOSP) 16.1.

Procedure

1. Open the **stf-connectors.yaml** file and add the **CollectdEnableLibpodstats** parameter to override the setting in **enable-stf.yaml**. Ensure that **stf-connectors.yaml** is called from the **openstack overcloud deploy** command after **enable-stf.yaml**:

```
CollectdEnableLibpodstats: false
```

2. Continue with the overcloud deployment procedure. For more information, see [Section 4.1.4, "Deploying the overcloud"](#).

5.8. RED HAT OPENSTACK PLATFORM API STATUS AND CONTAINERIZED SERVICES HEALTH

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

Monitoring of RHOSP container health and API status is enabled by default.

Additional resources

- To disable RHOSP container health and API status monitoring, see [Section 5.8.1, "Disabling container health and API status monitoring"](#).

5.8.1. Disabling container health and API status monitoring

To disable RHOSP containerized service health and API status monitoring, you must set the **CollectdEnableSensubility** parameter to **false**.

Prerequisites

- You have created the **stf-connectors.yaml** file in your templates directory. For more information, see [Section 4.1, "Deploying Red Hat OpenStack Platform overcloud for Service Telemetry Framework using director"](#).
- You are using the most current version of Red Hat OpenStack Platform (RHOSP) 16.1.

Procedure

1. Open the **stf-connectors.yaml** and add the **CollectdEnableSensubility** parameter to override the setting in **enable-stf.yaml**. Ensure that **stf-connectors.yaml** is called from the **openstack overcloud deploy** command after **enable-stf.yaml**:

```
CollectdEnableSensubility: false
```

2. Continue with the overcloud deployment procedure. For more information, see [Section 4.1.4, "Deploying the overcloud"](#).

Additional resources

- For more information about multiple cloud addresses, see [Section 4.3, "Configuring multiple clouds"](#).

CHAPTER 6. REMOVING SERVICE TELEMETRY FRAMEWORK FROM THE RED HAT OPENSIFT CONTAINER PLATFORM ENVIRONMENT

Remove Service Telemetry Framework (STF) from an Red Hat OpenShift Container Platform environment if you no longer require the STF functionality.

To remove STF from the Red Hat OpenShift Container Platform environment, you must perform the following tasks:

1. Delete the namespace.
2. Remove the cert-manager Operator.

6.1. DELETING THE NAMESPACE

To remove the operational resources for STF from Red Hat OpenShift Container Platform, 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.
```

6.2. REMOVING THE CERT-MANAGER OPERATOR

If you are not using the cert-manager Operator for any other applications, delete the Subscription, ClusterServiceVersion, and CustomResourceDefinitions.

Procedure

1. Delete the Subscription from the **openshift-cert-manager-operator** namespace:

```
$ oc delete --namespace=openshift-cert-manager-operator subscription openshift-cert-  
manager-operator  
  
subscription.operators.coreos.com "openshift-cert-manager-operator" deleted
```

2. Retrieve the version number of your installed ClusterServiceVersion:

```
$ oc get --namespace=openshift-cert-manager-operator subscription openshift-cert-manager-  
operator -oyaml | grep currentCSV
```

Example output:

```
currentCSV: openshift-cert-manager.v1.7.1
```

3. Delete the ClusterServiceVersion from the **openshift-cert-manager-operator** namespace:

```
$ oc delete --namespace=openshift-cert-manager-operator csv openshift-cert-manager.v1.7.1
```

Example output:

```
clusterserviceversion.operators.coreos.com "openshift-cert-manager.v1.7.1" deleted
```

4. Get the current list of CustomResourceDefinitions provided by the Operator so they can be deleted after removal of the ClusterServiceVersion:

```
$ oc get csv -n openshift-cert-manager-operator openshift-cert-manager.v1.7.1 -oyaml | grep "kind: CustomResourceDefinition" -A2 | grep name | awk '{print $2}'
```

Example output:

```
certificaterequests.cert-manager.io
certificates.cert-manager.io
certmanagers.config.openshift.io
certmanagers.operator.openshift.io
challenges.acme.cert-manager.io
clusterissuers.cert-manager.io
issuers.cert-manager.io
orders.acme.cert-manager.io
```

5. Delete the CustomResourceDefinitions related to the cert-manager Operator:

```
$ oc delete crd certificaterequests.cert-manager.io certificates.cert-manager.io
certmanagers.config.openshift.io certmanagers.operator.openshift.io challenges.acme.cert-
manager.io clusterissuers.cert-manager.io issuers.cert-manager.io orders.acme.cert-
manager.io
```

Example output:

```
customresourcedefinition.apiextensions.k8s.io "certificaterequests.cert-manager.io" deleted
customresourcedefinition.apiextensions.k8s.io "certificates.cert-manager.io" deleted
customresourcedefinition.apiextensions.k8s.io "certmanagers.config.openshift.io" deleted
customresourcedefinition.apiextensions.k8s.io "certmanagers.operator.openshift.io" deleted
customresourcedefinition.apiextensions.k8s.io "challenges.acme.cert-manager.io" deleted
customresourcedefinition.apiextensions.k8s.io "clusterissuers.cert-manager.io" deleted
customresourcedefinition.apiextensions.k8s.io "issuers.cert-manager.io" deleted
customresourcedefinition.apiextensions.k8s.io "orders.acme.cert-manager.io" deleted
```

6. Delete the namespaces owned by the cert-manager Operator:

```
$ oc delete project openshift-cert-manager openshift-cert-manager-operator
```

Example output:


```
project.project.openshift.io "openshift-cert-manager" deleted  
project.project.openshift.io "openshift-cert-manager-operator" deleted
```

Additional information

- [Deleting Operators from a cluster](#) .

CHAPTER 7. UPGRADING SERVICE TELEMETRY FRAMEWORK TO VERSION 1.5

To upgrade Service Telemetry Framework (STF) 1.4 to STF 1.5, you must complete the following steps:

- Replace AMQ Certificate Manager with Certificate Manager.
- Remove the **ClusterServiceVersion** and **Subscription** objects for Smart Gateway Operator and Service Telemetry Operator in the **service-telemetry** namespace on your Red Hat OpenShift Container Platform environment.
- Upgrade Red Hat OpenShift Container Platform from 4.8 to 4.10.
- Re-enable the operators that you removed.

Prerequisites

- You have backed up your data. There is an outage during the Red Hat OpenShift Container Platform upgrade. You cannot reconfigure the **ServiceTelemetry** and **SmartGateway** objects during the Operators replacement.
- You have prepared your environment for upgrade from Red Hat OpenShift Container Platform 4.8 to the supported version, 4.10.
- The Red Hat OpenShift Container Platform cluster is fully-connected. STF does not support disconnected or restricted-network clusters.

7.1. REMOVING THE SERVICE TELEMETRY FRAMEWORK 1.4 OPERATORS

Remove the Service Telemetry Framework (STF) 1.4 Operators and the AMQ Certificate Manager Operator from the Red Hat OpenShift Container Platform 4.8.

Procedure

1. Remove the Service Telemetry Operator.
2. Remove the Smart Gateway Operator.
3. Remove the AMQ Certificate Manager Operator.
4. Remove the Grafana Operator.

Additional resources

- For more information about removing Operators from the Red Hat OpenShift Container Platform, see [Deleting Operators from a cluster](#).

7.1.1. Removing the Service Telemetry Operator

As part of upgrading your Service Telemetry Framework (STF) installation, you must remove the Service Telemetry Operator in the **service-telemetry** namespace on your Red Hat OpenShift Container Platform environment.

Procedure

1. Change to the **service-telemetry** project:

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

2. Remove the Service Telemetry Operator Subscription:

```
$ oc delete sub --selector=operators.coreos.com/service-telemetry-operator.service-
telemetry
subscription.operators.coreos.com "service-telemetry-operator" deleted
```

3. Remove the Service Telemetry Operator **ClusterServiceVersion**:

```
$ oc delete csv --selector=operators.coreos.com/service-telemetry-operator.service-
telemetry
clusterserviceversion.operators.coreos.com "service-telemetry-operator.v1.4.1669718959"
deleted
```

Verification

1. Verify that the Service Telemetry Operator deployment is not running:

```
$ oc get deploy --selector=operators.coreos.com/service-telemetry-operator.service-
telemetry
No resources found in service-telemetry namespace.
```

2. Verify the Service Telemetry Operator subscription is absent:

```
$ oc get sub --selector=operators.coreos.com/service-telemetry-operator.service-telemetry
No resources found in service-telemetry namespace.
```

3. Verify the Service Telemetry Operator ClusterServiceVersion is absent:

```
$ oc get csv --selector=operators.coreos.com/service-telemetry-operator.service-telemetry
No resources found in service-telemetry namespace.
```

7.1.2. Removing the Smart Gateway Operator

As part of upgrading your Service Telemetry Framework (STF) installation, you must remove the Smart Gateway Operator in the **service-telemetry** namespace on your Red Hat OpenShift Container Platform environment.

Procedure

1. Change to the **service-telemetry** project:

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

-
2. Remove the Smart Gateway Operator Subscription:

```
$ oc delete sub --selector=operators.coreos.com/smart-gateway-operator.service-telemetry
subscription.operators.coreos.com "smart-gateway-operator-stable-1.4-redhat-operators-
openshift-marketplace" deleted
```

3. Remove the Smart Gateway Operator **ClusterServiceVersion**:

```
$ oc delete csv --selector=operators.coreos.com/smart-gateway-operator.service-telemetry
clusterserviceversion.operators.coreos.com "smart-gateway-operator.v4.0.1669718962"
deleted
```

Verification

1. Verify that the Smart Gateway Operator deployment is not running:

```
$ oc get deploy --selector=operators.coreos.com/smart-gateway-operator.service-telemetry

No resources found in service-telemetry namespace.
```

2. Verify the Smart Gateway Operator subscription is absent:

```
$ oc get sub --selector=operators.coreos.com/smart-gateway-operator.service-telemetry

No resources found in service-telemetry namespace.
```

3. Verify the Smart Gateway Operator ClusterServiceVersion is absent:

```
$ oc get csv --selector=operators.coreos.com/smart-gateway-operator.service-telemetry

No resources found in service-telemetry namespace.
```

7.1.3. Removing the AMQ Certificate Manager Operator

Procedure

1. Remove the AMQ Certificate Manager Operator Subscription:

```
$ oc delete sub --namespace openshift-operators --selector=operators.coreos.com/amq7-
cert-manager-operator.openshift-operators

subscription.operators.coreos.com "amq7-cert-manager-operator" deleted
```

2. Remove the AMQ Certificate Manager Operator **ClusterServiceVersion**:

```
$ oc delete csv --namespace openshift-operators --selector=operators.coreos.com/amq7-
cert-manager-operator.openshift-operators

clusterserviceversion.operators.coreos.com "amq7-cert-manager.v1.0.11" deleted
```

Verification

1. Verify that the AMQ Certificate Manager Operator deployment is not running:

```
$ oc get deploy --namespace openshift-operators --selector=operators.coreos.com/amq7-cert-manager-operator.openshift-operators
```

No resources found in openshift-operators namespace.

2. Verify that the AMQ Certificate Manager Operator subscription is absent:

```
$ oc get sub --namespace openshift-operators --selector=operators.coreos.com/amq7-cert-manager-operator.service-telemetry
```

No resources found in openshift-operators namespace.

3. Verify that the AMQ Certificate Manager Operator Cluster Service Version is absent:

```
$ oc get csv --namespace openshift-operators --selector=operators.coreos.com/amq7-cert-manager-operator.openshift-operators
```

No resources found in openshift-operators namespace.

7.1.4. Removing the Grafana Operator

Procedure

1. Remove the Grafana Operator Subscription:

```
$ oc delete sub --selector=operators.coreos.com/grafana-operator.service-telemetry
subscription.operators.coreos.com "grafana-operator" deleted
```

2. Remove the Grafana Operator **ClusterServiceVersion**:

```
$ oc delete csv --selector=operators.coreos.com/grafana-operator.service-telemetry
clusterserviceversion.operators.coreos.com "grafana-operator.v3.10.3" deleted
```

Verification

1. Verify the Grafana Operator deployment is not running:

```
$ oc get deploy --selector=operators.coreos.com/grafana-operator.service-telemetry
```

No resources found in service-telemetry namespace.

2. Verify the Grafana Operator subscription is absent:

```
$ oc get sub --selector=operators.coreos.com/grafana-operator.service-telemetry
```

No resources found in service-telemetry namespace.

3. Verify the Grafana Operator Cluster Service Version is absent:

```
$ oc get csv --selector=operators.coreos.com/grafana-operator.service-telemetry
```

No resources found in service-telemetry namespace.

7.2. UPGRADING RED HAT OPENSIFT CONTAINER PLATFORM TO 4.10

Service Telemetry Framework (STF) 1.5 is only compatible with Red Hat OpenShift Container Platform 4.10. For more information about upgrading your Red Hat OpenShift Container Platform from 4.8 to 4.10, see [Updating clusters overview](#).

Prerequisites

- You removed the STF 1.4 Operators.
- You removed the AMQ Certificate Manager Operator and Grafana Operator. You must remove the Operators before you upgrade Red Hat OpenShift Container Platform because the Operator APIs are incompatible with 4.10. For more information about preparing your Red Hat OpenShift Container Platform for upgrade from 4.8 to 4.10, see [Understanding OpenShift Container Platform updates](#).
- Verify the suitability of your Red Hat OpenShift Container Platform upgrade:

```
$ oc adm upgrade
```

You cannot upgrade the cluster if you encounter the following error message:

```
Cluster operator operator-lifecycle-manager should not be upgraded between minor versions:
ClusterServiceVersions blocking cluster upgrade: service-telemetry/grafana-operator.v3.10.3
is incompatible with OpenShift minor versions greater than 4.8,openshift-operators/amq7-
cert-manager.v1.0.11 is incompatible with OpenShift minor versions greater than 4.8
```

7.3. INSTALLING THE SERVICE TELEMETRY FRAMEWORK 1.5 OPERATORS

Install the Service Telemetry Framework (STF) 1.5 Operators and the Certificate Manager for OpenShift Operator on your Red Hat OpenShift Container Platform environment. See [Section 1.1, "Support for Service Telemetry Framework"](#) for more information about STF support status and life cycle.

Prerequisites

- You have upgraded your Red Hat OpenShift Container Platform environment to 4.10. For more information about upgrading Red Hat OpenShift Container Platform, see [Section 7.2, "Upgrading Red Hat OpenShift Container Platform to 4.10"](#).
- Your Red Hat OpenShift Container Platform environment network is fully-connected.

Procedure

1. Change to the **service-telemetry** project:

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

2. Create a **namespace** for the **cert-manager** Operator:

```
$ oc create -f - <<EOF
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  name: openshift-cert-manager-operator
spec:
  finalizers:
    - kubernetes
EOF
```

3. Create an **OperatorGroup** for the cert-manager Operator:

```
$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-cert-manager-operator
  namespace: openshift-cert-manager-operator
spec: {}
EOF
```

4. Subscribe to the **cert-manager** Operator with the **redhat-operators** CatalogSource:

```
$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-cert-manager-operator
  namespace: openshift-cert-manager-operator
spec:
  channel: tech-preview
  installPlanApproval: Automatic
  name: openshift-cert-manager-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

5. Validate your **ClusterServiceVersion**. Ensure that the phase of **cert-manager** Operator is **Succeeded**:

```
$ oc get csv --namespace openshift-cert-manager-operator --
selector=operators.coreos.com/openshift-cert-manager-operator.openshift-cert-manager-
operator
```

NAME	DISPLAY	VERSION	REPLACES	PHASE
openshift-cert-manager.v1.7.1	cert-manager Operator	for Red Hat OpenShift	1.7.1-1	Succeeded

6. Optional: Resubscribe to the Grafana Operator. For more information, see: test [Section 5.1.1](#), “Configuring Grafana to host the dashboard”.
7. Create the Service Telemetry Operator subscription to manage the STF instances:

```
$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: service-telemetry-operator
  namespace: service-telemetry
spec:
  channel: stable-1.5
  installPlanApproval: Automatic
  name: service-telemetry-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

8. Validate the Service Telemetry Operator and the dependent operators:

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

NAME	DISPLAY	VERSION
REPLACES	PHASE	
amq7-interconnect-operator.v1.10.13	Red Hat Integration - AMQ Interconnect	
1.10.13	amq7-interconnect-operator.v1.10.4	Succeeded
elasticsearch-ecck-operator-certified.v2.5.0	Elasticsearch (ECK) Operator	2.5.0
elasticsearch-ecck-operator-certified.v2.4.0	Succeeded	
openshift-cert-manager.v1.7.1	cert-manager Operator for Red Hat OpenShift	
1.7.1-1	Succeeded	
prometheusoperator.0.47.0	Prometheus Operator	0.47.0
prometheusoperator.0.37.0	Succeeded	
service-telemetry-operator.v1.5.1669950702	Service Telemetry Operator	
1.5.1669950702	Succeeded	
smart-gateway-operator.v5.0.1669950681	Smart Gateway Operator	
5.0.1669950681	Succeeded	

Verification

- Verify that the Service Telemetry Operator has successfully reconciled.

```
$ oc logs -f --selector=name=service-telemetry-operator

[...]
----- Ansible Task Status Event StdOut (infra.watch/v1beta1, Kind=ServiceTelemetry,
default/service-telemetry) -----

PLAY RECAP *****
localhost      : ok=115 changed=0  unreachable=0  failed=0   skipped=21
rescued=0  ignored=0

$ oc get pods
```


NAME	READY	STATUS	RESTARTS	AGE
alertmanager-default-0	3/3	Running	0	20h
default-cloud1-ceil-event-smartgateway-6d57ffbdbc-5mrj8	2/2	Running	1 (3m42s ago)	4m21s
default-cloud1-ceil-meter-smartgateway-67684d88c8-62mp7	3/3	Running	1 (3m43s ago)	4m20s
default-cloud1-coll-event-smartgateway-66cddd5567-qb6p2	2/2	Running	1 (3m42s ago)	4m19s
default-cloud1-coll-meter-smartgateway-76d5ff6db5-z5ch8	3/3	Running	0	4m20s
default-cloud1-sens-meter-smartgateway-7b59669fdd-c42zg	3/3	Running	1 (3m43s ago)	4m20s
default-interconnect-845c4b647c-wwfcm	1/1	Running	0	4m10s
elastic-operator-57b57964c5-6q88r	1/1	Running	8 (17h ago)	20h
elasticsearch-es-default-0	1/1	Running	0	21h
grafana-deployment-59c54f7d7c-zjnhm	2/2	Running	0	20h
interconnect-operator-848889bf8b-bq2zx	1/1	Running	0	20h
prometheus-default-0	3/3	Running	1 (20h ago)	20h
prometheus-operator-5d7b69fd46-c2xlv	1/1	Running	0	20h
service-telemetry-operator-79fb664dfb-nj8jn	1/1	Running	0	5m11s
smart-gateway-operator-79557664f8-ql7xr	1/1	Running	0	5m7s