



# Red Hat OpenStack Platform 16.1

## Service Telemetry Framework 1.3

Service Telemetry Framework 1.3 のインストールおよびデプロイ



# Red Hat OpenStack Platform 16.1 Service Telemetry Framework 1.3

---

Service Telemetry Framework 1.3 のインストールおよびデプロイ

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

## 法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Service\_Telemetry\_Framework\_1.3.ent file |.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## 概要

コアコンポーネントをインストールし、Service Telemetry Framework 1.3 をデプロイします。

## 目次

<b>第1章 SERVICE TELEMETRY FRAMEWORK 1.3 の概要</b> .....	<b>6</b>
1.1. SERVICE TELEMETRY FRAMEWORK のサポート	6
1.2. SERVICE TELEMETRY FRAMEWORK のアーキテクチャー	6
1.3. RED HAT OPENSIFT CONTAINER PLATFORM のインストールサイズ	9
<b>第2章 SERVICE TELEMETRY FRAMEWORK 向けの RED HAT OPENSIFT CONTAINER PLATFORM 環境の準備</b>	<b>10</b>
2.1. 永続ボリューム	10
2.1.1. 一時ストレージ	10
2.2. リソースの割り当て	11
<b>第3章 SERVICE TELEMETRY FRAMEWORK のコアコンポーネントのインストール</b> .....	<b>12</b>
3.1. SERVICE TELEMETRY FRAMEWORK の RED HAT OPENSIFT CONTAINER PLATFORM 環境へのデプロイ	12
3.2. RED HAT OPENSIFT CONTAINER PLATFORM での SERVICE TELEMETRY オブジェクトの作成	15
3.2.1. ServiceTelemetry オブジェクトのパラメーター	18
バックエンドパラメーター	18
メトリクスのストレージバックエンドとしての Prometheus の有効化	18
Prometheus の永続ストレージの設定	19
ElasticSearch のイベントのストレージバックエンドとしての有効化	19
ElasticSearch の永続ストレージの設定	20
clouds パラメーター	21
alerting パラメーター	22
graphing パラメーター	22
highAvailability パラメーター	22
transports パラメーター	22
3.3. RED HAT OPENSIFT CONTAINER PLATFORM 環境からの SERVICE TELEMETRY FRAMEWORK の削除	22
3.3.1. namespace の削除	22
3.3.2. CatalogSource の削除	23
<b>第4章 サービス TELEMETRY フレームワーク向けの RED HAT OPENSTACK PLATFORM の設定</b> .....	<b>24</b>
4.1. SERVICE TELEMETRY FRAMEWORK 向けの RED HAT OPENSTACK PLATFORM オーバークラウドのデプロイ	24
4.1.1. AMQ Interconnect ルートアドレスの取得	24
4.1.2. STF のベース設定の作成	25
4.1.3. オーバークラウドの STF 接続の設定	27
4.1.4. オーバークラウドのデプロイ	27
4.1.5. クライアント側のインストールの検証	28
4.2. GNOCCHI および SERVICE TELEMETRY FRAMEWORK へのメトリックの送信	31
4.3. 標準以外のネットワークポロジへのデプロイ	33
4.4. 複数クラウドの設定	34
4.4.1. AMQP アドレス接頭辞の計画	35
4.4.2. Smart Gateway のデプロイ	36
4.4.3. デフォルトの Smart Gateway の削除	38
4.4.4. 一意のクラウドドメインの設定	39
4.4.5. 複数クラウドの Red Hat OpenStack Platform 環境ファイルの作成	40
4.4.6. 複数クラウドからのメトリクスデータのクエリー	42
<b>第5章 SERVICE TELEMETRY FRAMEWORK の運用機能の使用</b> .....	<b>43</b>
5.1. SERVICE TELEMETRY FRAMEWORK でのダッシュボード	43
5.1.1. ダッシュボードをホストするための Grafana の設定	43
5.1.2. デフォルトの Grafana コンテナイメージの上書き	45

5.1.3. ダッシュボードのインポート	45
5.1.4. Grafana ログイン認証情報の取得および設定	47
5.2. SERVICE TELEMETRY FRAMEWORK でのメトリクスの保持期間	47
5.2.1. Service Telemetry Framework でのメトリクスの保持期間の編集	48
5.3. SERVICE TELEMETRY FRAMEWORK でのアラート	49
5.3.1. Prometheus でのアラートルールの作成	49
5.3.2. カスタムアラートの設定	50
5.3.3. Alertmanager でのアラートルートの作成	51
5.4. SNMP トラップの設定	53
5.5. 高可用性	54
5.5.1. 高可用性の設定	55
5.6. 一時ストレージ	55
5.6.1. 一時ストレージの設定	55
5.7. RED HAT OPENSIFT CONTAINER PLATFORM でのルートの作成	56
5.8. RED HAT OPENSTACK PLATFORM サービスのリソース使用状況	58
5.8.1. Red Hat OpenStack Platform サービスのリソース使用状況の監視の無効化	58
5.9. RED HAT OPENSTACK PLATFORM API のステータスおよびコンテナ化されたサービスの健全性	58
5.9.1. コンテナの正常性および API ステータスマonitoringの無効化	59
<b>第6章 SERVICE TELEMETRY FRAMEWORK のバージョン 1.3 へのアップグレード</b>	<b>60</b>
6.1. SERVICE TELEMETRY FRAMEWORK 1.2 OPERATOR の削除	60
6.2. SERVICE TELEMETRY OPERATOR のサブスクリプト	63
<b>第7章 COLLECTD プラグイン</b>	<b>65</b>
collectd::plugin::aggregation	65
collectd::plugin::ampq	65
collectd::plugin::amqp1	65
collectd::plugin::apache	66
collectd::plugin::battery	66
collectd::plugin::bind	67
collectd::plugin::ceph	67
collectd::plugins::cgroups	68
collectd::plugin::connectivity	68
collectd::plugin::contrack	69
collectd::plugin::contextswitch	69
collectd::plugin::cpu	69
collectd::plugin::cpufreq	70
collectd::plugin::cpusleep	70
collectd::plugin::csv	70
collectd::plugin::curl_json	70
collectd::plugin::curl	70
collectd::plugin::curl_xml	70
collectd::plugin::dbi	70
collectd::plugin::df	70
collectd::plugin::disk	71
collectd::plugin::dns	72
collectd::plugin::dpsdk_telemetry	72
collectd::plugin::entropy	72
collectd::plugin::ethstat	72
collectd::plugin::exec	73
collectd::plugin::fhcount	73
collectd::plugin::filecount	73
collectd::plugin::fscache	73

---

collectd-hddtemp	73
collectd::plugin::hugepages	73
collectd::plugin::intel_rdt	74
collectd::plugin::interface	74
collectd::plugin::ipc	75
collectd::plugin::ipmi	75
collectd::plugin::iptables	75
collectd::plugin::irq	75
collectd::plugin::load	75
collectd::plugin::logfile	75
collectd::plugin::log_logstash	76
collectd::plugin::madwifi	76
collectd::plugin::match_empty_counter	76
collectd::plugin::match_hashed	76
collectd::plugin::match_regex	76
collectd::plugin::match_timediff	76
collectd::plugin::match_value	76
collectd::plugin::mbmon	76
collectd::plugin::mcelog	76
collectd::plugin::md	76
collectd::plugin::memcachec	76
collectd::plugin::memcached	76
collectd::plugin::memory	76
collectd::plugin::multimeter	77
collectd::plugin::mysql	77
collectd::plugin::netlink	77
collectd::plugin::network	77
collectd::plugin::nfs	78
collectd::plugin::notify_nagios	78
collectd::plugin::ntpd	78
collectd::plugin::numa	78
collectd::plugin::olsrd	78
collectd::plugin::openldap	78
collectd::plugin::openvpn	78
collectd::plugin::ovs_stats	78
collectd::plugin::pcie_errors	79
collectd::plugin::ping	79
collectd::plugin::powerdns	80
collectd::plugin::processes	80
collectd::plugin::processes	80
collectd::plugin::protocols	81
collectd::plugin::python	81
collectd::plugin::sensors	81
collectd::plugin::serial	81
collectd::plugin::smart	81
collectd::plugin::snmp	81
collectd::plugin::snmp_agent	81
collectd::plugin::statsd	81
collectd::plugin::swap	82
collectd::plugin::sysevent	82
collectd::plugin::syslog	82
collectd::plugin::table	82
collectd::plugin::tail	82

collectd::plugin::tail_csv	82
collectd::plugin::target_notification	82
collectd::plugin::target_replace	82
collectd::plugin::target_scale	82
collectd::plugin::target_set	82
collectd::plugin::target_v5upgrade	82
collectd::plugin::tcpconns	82
collectd::plugin::ted	83
collectd::plugin::thermal	83
collectd::plugin::threshold	83
collectd::plugin::turbostat	83
collectd::plugin::unixsock	83
collectd::plugin::uptime	83
collectd::plugin::users	83
collectd::plugin::uuid	83
collectd::plugin::virt	83
collectd::plugin::vmem	84
collectd::plugin::vserver	84
collectd::plugin::wireless	84
collectd::plugin::write_graphite	84
collectd::plugin::write_http	85
collectd::plugin::write_kafka	85
collectd::plugin::write_log	86
collectd::plugin::zfs_arc	86





## 第1章 SERVICE TELEMETRY FRAMEWORK 1.3 の概要

Service Telemetry Framework (STF) は、Red Hat OpenStack Platform (`{OpenStackShort}`) またはサードパーティーのノードからモニタリングデータを収集します。STF を使用して、以下のタスクを実行できます。

- 履歴情報の監視データの格納またはアーカイブします。
- ダッシュボードで図表としてモニタリングデータを表示します。
- モニタリングデータを使用したアラートまたは警告をトリガーします。

モニタリングデータはメトリクスまたはイベントのいずれかです。

### メトリクス

アプリケーションまたはシステムの数値測定。

### Event

システムで不規則な状態や目立った事態の発生。

STF のコンポーネントは、データトランスポートにメッセージバスを使用します。データを受信して保存する他のモジュラーコンポーネントは、Red Hat OpenShift Container Platform のコンテナとしてデプロイされます。



### 重要

Service Telemetry Framework (STF) は、Red Hat OpenShift Container Platform バージョン 4.6 および 4.7 と互換性があります。

### 追加リソース

- Red Hat OpenShift Container Platform のデプロイ方法の詳細は、[Red Hat OpenShift Container Platform の製品ドキュメント](#)を参照してください。
- Red Hat OpenShift Container Platform は、クラウドプラットフォームまたはベアメタルにインストールできます。STF パフォーマンスおよびスケーリングの詳細は、<https://access.redhat.com/articles/4907241> を参照してください。
- Red Hat OpenShift Container Platform は、ベアメタルまたはその他のサポートされているクラウドプラットフォームにインストールできます。Red Hat OpenShift Container Platform のインストールの詳細は、[OpenShift Container Platform 4.7 ドキュメント](#)を参照してください。

## 1.1. SERVICE TELEMETRY FRAMEWORK のサポート

Red Hat は、最新バージョンの Service Telemetry Framework (STF) をサポートしています。以前のバージョンではサポートされていません。詳細は、「[Service Telemetry Framework Supported Version Matrix](#)」を参照してください。

## 1.2. SERVICE TELEMETRY FRAMEWORK のアーキテクチャー

Service Telemetry Framework (STF) は、クライアント/サーバーアーキテクチャーを使用します。Red Hat OpenStack Platform (`{OpenStackShort}`) はクライアントであり、Red Hat OpenShift Container Platform はサーバーです。

STF は、以下のコンポーネントで構成されます。

- データ収集
  - collectd: インフラストラクチャーメトリクスおよびイベントを収集します。
  - Ceilometer: {OpenStackShort} メトリクスおよびイベントを収集します。
- トランスポート
  - AMQ相互接続: AMQP 1.x互換のメッセージングバス。高速で信頼性の高いデータ転送を提供し、ストレージ用にメトリックをSTFに転送します。
  - Smart Gateway: AMQP 1.x バスからメトリクスおよびイベントを取得し、ElasticSearch または Prometheus に配信する Golang アプリケーション。
- データストレージ
  - Prometheus: Smart Gateway から受信される STF メトリクスを保存する時系列データストレージ。
  - Elasticsearch: Smart Gateway から受信される STF イベントを保存するデータストレージ。
- 観察
  - Alertmanager: Prometheus アラートルールを使用してアラートを管理するアラートツール。
  - Grafana: データのクエリー、視覚化、および管理に使用できる可視化アプリケーションおよび解析アプリケーション。

以下の表は、クライアントおよびサーバーコンポーネントのアプリケーションについて説明しています。

表1.1 STF のクライアントおよびサーバーコンポーネント

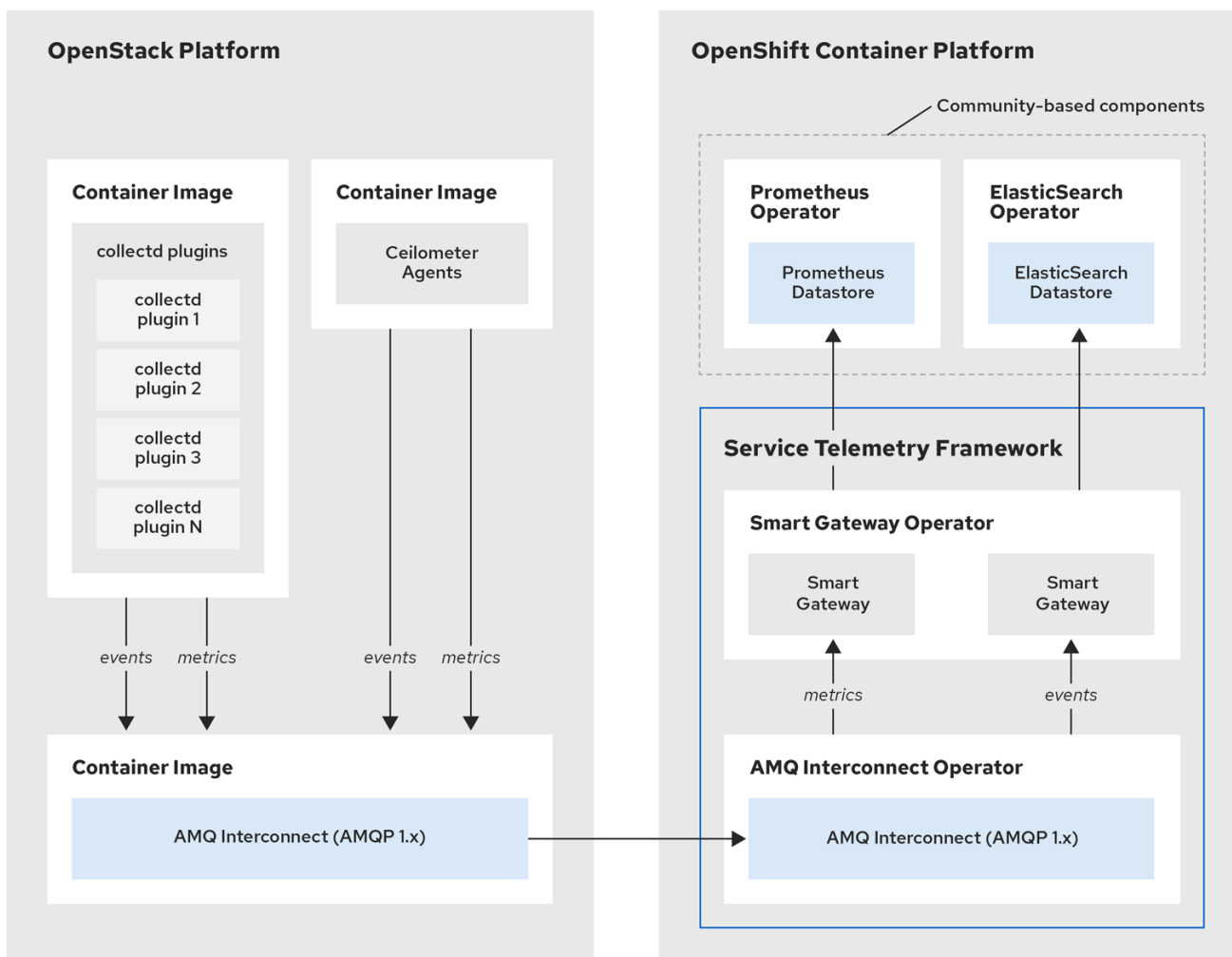
コンポーネント	クライアント	サーバー
AMQP 1.x と互換性のあるメッセージングバス	はい	はい
Smart Gateway	いいえ	はい
Prometheus	いいえ	はい
ElasticSearch	いいえ	はい
collectd	はい	いいえ
Ceilometer	はい	いいえ



### 重要

モニタリングプラットフォームがクラウドで動作する問題を報告できるようにするには、監視しているものと同じインフラストラクチャーに STF をインストールしないでください。

図1.1 Service Telemetry Framework アーキテクチャの概要



65\_OpenStack\_0620

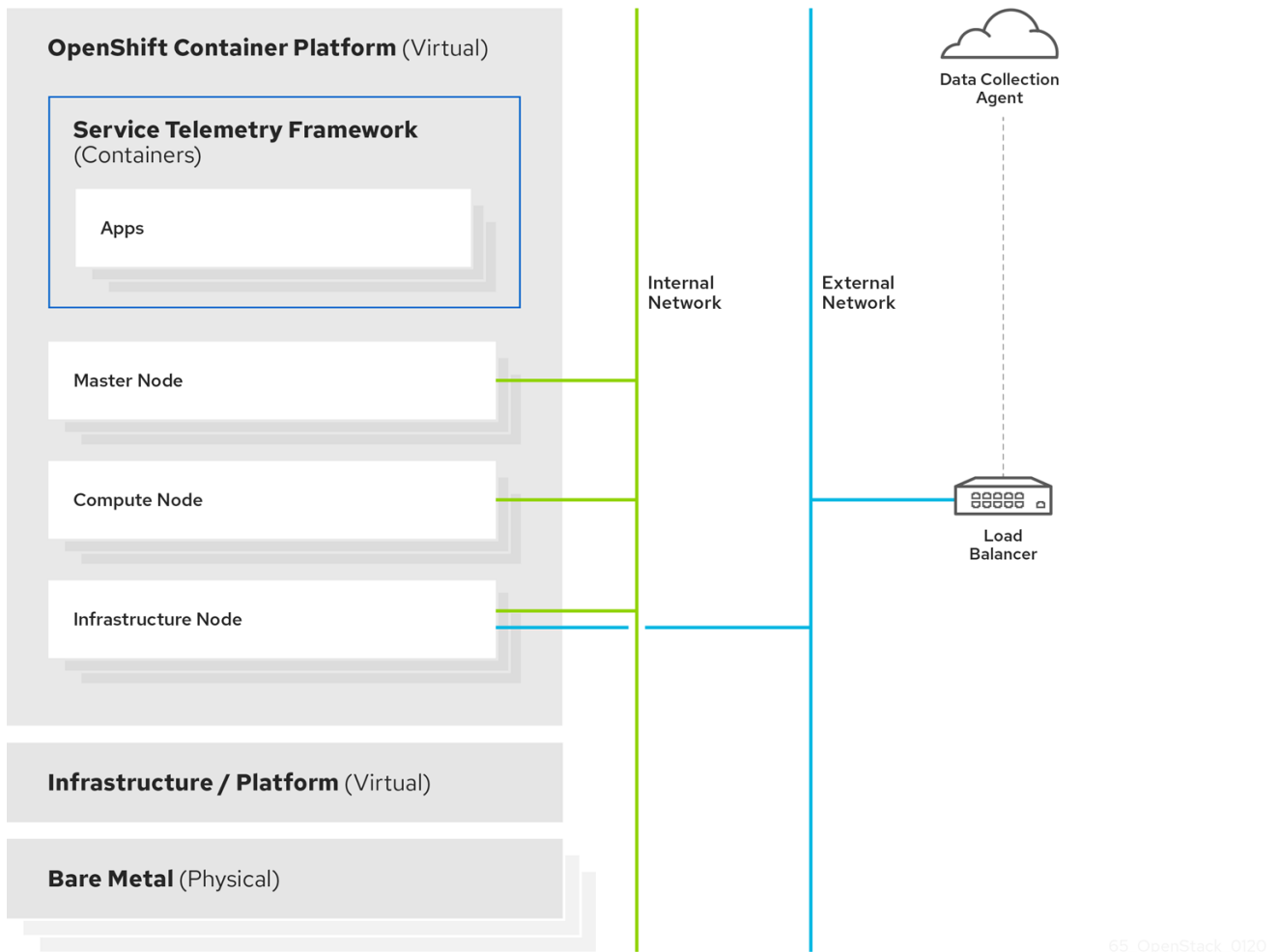
collectd はクライアント側でプロジェクトデータがないインフラストラクチャーメトリクスを提供し、Ceilometer はプロジェクトまたはユーザーのワークロードに基づいて Red Hat OpenStack Platform (OpenStackShort) プラットフォームデータを提供します。Ceilometer と collectd は、AMQ Interconnect トランスポートを使用してメッセージバスを介してデータを提供し、データを Prometheus に渡します。サーバー側では、Smart Gateway と呼ばれる Golang アプリケーションがバスからデータストリームを取得し、Prometheus のローカル収集エンドポイントとして公開します。

イベントを収集して保存する予定の場合は、collectd および Ceilometer が AMQ Interconnect トランスポートを使用し、イベントデータをサーバー側に渡します。別の Smart Gateway がデータを ElasticSearch データストアに書き込みます。

サーバー側の STF 監視インフラストラクチャーは、以下のレイヤーで構成されます。

- Service Telemetry Framework 1.3
- Red Hat OpenShift Container Platform 4.6 から 4.7
- インフラストラクチャープラットフォーム

図1.2 サーバー側の STF 監視インフラストラクチャー



### 1.3. RED HAT OPENSIFT CONTAINER PLATFORM のインストールサイズ

Red Hat OpenShift Container Platform のインストールサイズは、以下の要素によって異なります。

- 選択するインフラストラクチャー。
- 監視するノード数。
- 収集するメトリクスの数。
- メトリックの解決。
- データを保存する期間。

Service Telemetry Framework (STF) のインストールは、既存の Red Hat OpenShift Container Platform 環境によって異なります。

Red Hat OpenShift Container Platform をベアメタルにインストールする場合の最低リソース要件の詳細は、『[ベアメタルへのインストール](#)』の「[最小リソース要件](#)」を参照してください。インストールできる各種パブリックおよびプライベートのクラウドプラットフォームのインストール要件については、選択したクラウドプラットフォームに対応するインストールドキュメントを参照してください。

## 第2章 SERVICE TELEMETRY FRAMEWORK 向けの RED HAT OPENSIFT CONTAINER PLATFORM 環境の準備

Service Telemetry Framework (STF) 用に Red Hat OpenShift Container Platform 環境を準備するには、永続ストレージ、適切なリソース、およびイベントストレージについて計画する必要があります。

- 実稼働環境レベルのデプロイメントができるように、永続ストレージが Red Hat OpenShift Container Platform クラスターで利用できるようにしてください。詳細は、[「永続ボリューム」](#) を参照してください。
- Operator およびアプリケーションコンテナを実行するために十分なリソースが利用可能であることを確認します。詳細は、[「リソースの割り当て」](#) を参照してください。

### 2.1. 永続ボリューム

Service Telemetry Framework(STF)は Red Hat OpenShift Container Platform の永続ストレージを使用して永続ボリュームを要求し、Prometheus および Elasticsearch がメトリクスおよびイベントを格納できるようにします。

永続ストレージが Service Telemetry Operator で有効な場合には、STF デプロイメントで要求される Persistent Volume Claim (永続ボリューム要求、PVC) のアクセスモードは RWO (ReadWriteOnce) になります。お使いの環境に事前にプロビジョニングされた永続ボリュームが含まれている場合は、Red Hat OpenShift Container Platform でデフォルト設定されている **storageClass** で RWO のボリュームが利用できるようにしてください。

#### 追加リソース

- Red Hat OpenShift Container Platform の永続ストレージの設定の詳細は、[「永続ストレージについて」](#) を参照してください。
- Red Hat OpenShift Container Platform で設定可能な推奨のストレージ技術の詳細は、[「設定可能な推奨のストレージ技術」](#) を参照してください。
- STF での Prometheus 用の永続ストレージの設定に関する詳細は、[「Prometheus の永続ストレージの設定」](#) を参照してください。
- STF での Elasticsearch の永続ストレージの設定に関する詳細は、[「ElasticSearch の永続ストレージの設定」](#) を参照してください。

#### 2.1.1. 一時ストレージ

一時ストレージを使用して、Red Hat OpenShift Container Platform クラスターにデータを永続的に保存せずに Service Telemetry Framework (STF) を実行できます。



#### 警告

一時ストレージを使用する場合は、Pod の再起動、更新、別のノードでの再スケジューリングが発生した場合に、データが損失する可能性があります。一時ストレージは、開発またはテスト環境にのみ使用してください。

## 2.2. リソースの割り当て

Red Hat OpenShift Container Platform インフラストラクチャー内での Pod のスケジューリングを有効にするには、実行中のコンポーネント向けにリソースが必要になります。十分なリソースが割り当てられていない場合には、Pod をスケジュールできないため **Pending** 状態のままになります。

Service Telemetry Framework (STF)の実行に必要なリソースの量は、環境とモニターするノードおよびクラウドの数によって異なります。

### 追加リソース

- メトリクス収集のサイジングについては、「[Service Telemetry Framework Performance and Scaling](#)」を参照してください。
- Elasticsearch のサイジング要件の詳細は <https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-managing-compute-resources.html> を参照してください。

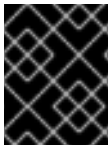
## 第3章 SERVICE TELEMETRY FRAMEWORK のコアコンポーネントのインストール

Operator を使用して Service Telemetry Framework (STF) コンポーネントおよびオブジェクトを読み込むことができます。Operator は以下の STF コアおよびコミュニティコンポーネントのそれぞれを管理します。

- AMQ Interconnect
- Smart Gateway
- Prometheus および AlertManager
- ElasticSearch
- Grafana

### 要件

- 4.6 かた 4.7 までが含まれる Red Hat OpenShift Container Platform バージョンが実行中である。
- Red Hat OpenShift Container Platform 環境を準備し、永続ストレージがあり、Red Hat OpenShift Container Platform 環境の上部で STF コンポーネントを実行するのに十分なリソースがあることを確認している。詳細は、[Service Telemetry Framework Performance and Scaling](#) を参照してください。



### 重要

STF は、4.6 から 4.7 までの Red Hat OpenShift Container Platform バージョンと互換性があります。

### 追加リソース

- Operator の詳細は、『[Operator について](#)』を参照してください。

## 3.1. SERVICE TELEMETRY FRAMEWORK の RED HAT OPENSIFT CONTAINER PLATFORM 環境へのデプロイ

Service Telemetry Framework (STF) をデプロイして、イベントを収集し、保存し、監視します。

### 手順

1. STF コンポーネントが含まれる namespace を作成します (例: **service-telemetry**) 。

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

2. Operator Pod をスケジュールできるように、namespace に OperatorGroup を作成します。

```
$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
```



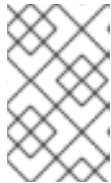
```

name: service-telemetry-operator-group
namespace: service-telemetry
spec:
  targetNamespaces:
  - service-telemetry
EOF

```

詳細は、「[OperatorGroups](#)」を参照してください。

3. OperatorHub.io Community Catalog Source を有効にし、データストレージおよび可視化 Operator をインストールします。



### 注記

Red Hat は、AMQ Interconnect、AMQ Certificate Manager、Service Telemetry Operator、および Smart Gateway Operator を含むコア Operator およびワークロードをサポートします。

```

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

```

4. redhat-operators CatalogSource を使用して AMQ Certificate Manager Operator にサブスクライブします。



### 注記

AMQ Certificate Manager は **openshift-operators** namespace にデプロイしてから、クラスター全体のすべての namespace で利用可能になります。その結果、namespace が多数あるクラスターでは、Operator が **service-telemetry** namespace で利用可能になるまでに数分の時間がかかる場合があります。他の namespace スコープの Operator と使用する場合には、AMQ Certificate Manager Operator は Operator Lifecycle Manager の依存関係管理と互換性がありません。

```

$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq7-cert-manager-operator
  namespace: openshift-operators
spec:
  channel: 1.x
  installPlanApproval: Automatic
  name: amq7-cert-manager-operator

```

```
source: redhat-operators
sourceNamespace: openshift-marketplace
EOF
```

- ClusterServiceVersion を検証します。amq7-cert-manager.v1.0.1 にフェーズ **Succeeded** があることを確認します。

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

NAME	DISPLAY	VERSION	REPLACES	PHASE
amq7-cert-manager.v1.0.1	Red Hat Integration - AMQ Certificate Manager	1.0.1		Succeeded

- イベントを ElasticSearch に保存する予定の場合は、Elastic Cloud on Kubernetes (ECK) Operator を有効にする必要があります。ECK Operator を有効にするには、Red Hat OpenShift Container Platform 環境で以下のマニフェストを作成します。

```
$ 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
```

- Kubernetes **Succeeded** で ElasticSearch Cloud の ClusterServiceVersion を確認します。

```
$ oc get csv
```

NAME	DISPLAY	VERSION	REPLACES	PHASE
...				
elasticsearch-eck-operator-certified.v1.7.1	Elasticsearch (ECK) Operator	1.7.1		
elasticsearch-eck-operator-certified.v1.6.0				Succeeded
...				

- Service Telemetry Operator サブスクリプションを作成し、STF インスタンスを管理します。

```
$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: service-telemetry-operator
  namespace: service-telemetry
spec:
  channel: stable-1.3
  installPlanApproval: Automatic
  name: service-telemetry-operator
```

```
source: redhat-operators
sourceNamespace: openshift-marketplace
EOF
```

- Service Telemetry Operator および依存する Operator を検証します。

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

amq7-cert-manager.v1.0.1          Red Hat Integration - AMQ Certificate Manager
1.0.1                             Succeeded
amq7-interconnect-operator.v1.10.1 Red Hat Integration - AMQ Interconnect
1.10.1      amq7-interconnect-operator.v1.2.4      Succeeded
elasticsearch-eck-operator-certified.v1.8.0 Elasticsearch (ECK) Operator          1.8.0
elasticsearch-eck-operator-certified.v1.7.1 Succeeded
prometheusoperator.0.47.0         Prometheus Operator                      0.47.0
prometheusoperator.0.37.0         Succeeded
service-telemetry-operator.v1.3.1632925572 Service Telemetry Operator
1.3.1632925572                    Succeeded
smart-gateway-operator.v3.0.1632925565 Smart Gateway Operator
3.0.1632925565                    Succeeded
```

## 3.2. RED HAT OPENSIFT CONTAINER PLATFORM での SERVICETELEMETRY オブジェクトの作成

Red Hat OpenShift Container Platform で **ServiceTelemetry** オブジェクトを作成します。これにより、Service Telemetry Operator が Service Telemetry Framework (STF) デプロイメントのサポートコンポーネントを作成します。詳細は、[「ServiceTelemetry オブジェクトのパラメーター」](#) を参照してください。

### 手順

- デフォルト値を使用して STF をデプロイする **ServiceTelemetry** オブジェクトを作成するには、空の **spec** パラメーターで **ServiceTelemetry** オブジェクトを作成します。

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

デフォルト値を上書きするには、上書きするパラメーターを定義します。この例では、**enabled** を **true** に設定して ElasticSearch を有効にします。

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

```
events:
  elasticsearch:
    enabled: true
EOF
```

空の **spec** パラメーターを使用して **ServiceTelemetry** オブジェクトを作成すると、STF デプロイメントに以下のデフォルト値が設定されます。

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
spec:
  alerting:
    alertmanager:
      storage:
        persistent:
          pvcStorageRequest: 20G
          storageSelector: {}
      receivers:
        snmpTraps:
          enabled: false
          target: 192.168.24.254
      strategy: persistent
    enabled: true
  backends:
    events:
      elasticsearch:
        enabled: false
      storage:
        persistent:
          pvcStorageRequest: 20Gi
          storageSelector: {}
      strategy: persistent
    metrics:
      prometheus:
        enabled: true
        scrapeInterval: 10s
      storage:
        persistent:
          pvcStorageRequest: 20G
          storageSelector: {}
        retention: 24h
        strategy: persistent
  graphing:
    enabled: false
    grafana:
      adminPassword: secret
      adminUser: root
      disableSignoutMenu: false
      ingressEnabled: false
      baseImage: docker.io/grafana/grafana:8.1.2
  highAvailability:
    enabled: false
  transports:
    qdr:
```

```

enabled: true
web:
  enabled: false
clouds:
  - name: cloud1
    metrics:
      collectors:
        - collectorType: collectd
          subscriptionAddress: collectd/telemetry
          debugEnabled: false
        - collectorType: ceilometer
          subscriptionAddress: anycast/ceilometer/metering.sample
          debugEnabled: false
        - collectorType: sensubility
          subscriptionAddress: sensubility/telemetry
          debugEnabled: false
    events:
      collectors:
        - collectorType: collectd
          subscriptionAddress: collectd/notify
          debugEnabled: false
        - collectorType: ceilometer
          subscriptionAddress: anycast/ceilometer/event.sample
          debugEnabled: false

```

これらのデフォルトを上書きするには、設定を **spec** パラメーターに追加します。

- Service Telemetry Operator で STF デプロイメントログを表示します。

```

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

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

PLAY RECAP *****
localhost          :ok=57  changed=0  unreachable=0  failed=0  skipped=20
rescued=0  ignored=0

```

## 検証

- Pod および各 Pod のステータスを表示し、すべてのワークロードが正常に動作していることを確認するには、以下を実行します。



### 注記

**backends.events.elasticsearch.enabled** を **true** 設定した場合、通知スマートゲートウェイは ElasticSearch を開始するまでの時間のために **Error** と **CrashLoopBackOff** エラーメッセージを報告します。

```

$ oc get pods

NAME                                READY STATUS RESTARTS AGE
alertmanager-default-0              2/2   Running 0      17m
default-cloud1-ceil-meter-smartgateway-6484b98b68-vd48z  2/2   Running 0      17m

```

```

default-cloud1-coll-meter-smartgateway-799f687658-4gxp 2/2 Running 0 17m
default-cloud1-sens-meter-smartgateway-c7f4f7fc8-c57b4 2/2 Running 0 17m
default-interconnect-54658f5d4-pzrpt 1/1 Running 0 17m
elastic-operator-66b7bc49c4-sxkc2 1/1 Running 0 52m
interconnect-operator-69df6b9cb6-7hhp9 1/1 Running 0 50m
prometheus-default-0 2/2 Running 1 17m
prometheus-operator-6458b74d86-wbdqp 1/1 Running 0 51m
service-telemetry-operator-864646787c-hd9pm 1/1 Running 0 51m
smart-gateway-operator-79778cf548-mz5z7 1/1 Running 0 51m

```

### 3.2.1. ServiceTelemetry オブジェクトのパラメーター

**ServiceTelemetry** オブジェクトは、以下の主要な設定パラメーターで構成されます。

- アラート
- バックエンド
- クラウド
- グラフ
- **highAvailability**
- トランスポート

これらの設定パラメーターをそれぞれ設定し、STF デプロイメントで異なる機能を提供できます。



#### 重要

**servicetelemetry.infra.watch/v1alpha1** のサポートは STF 1.3 から削除されました。

#### バックエンドパラメーター

**backends** パラメーターを使用して、メトリクスおよびイベントの保存に使用できるストレージバックエンドを制御し、**clouds** パラメーターで定義されている Smart Gateway の有効化を制御します。詳細は、「[clouds パラメーター](#)」を参照してください。

現時点で、Prometheus をメトリクスストレージバックエンドとして、ElasticSearch をイベントストレージバックエンドとして使用できます。

#### メトリクスのストレージバックエンドとしての Prometheus の有効化

Prometheus をメトリクスのストレージバックエンドとして有効にするには、**ServiceTelemetry** オブジェクトを設定する必要があります。

#### 手順

- **ServiceTelemetry** オブジェクトを設定します。

```

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

```

```
metrics:
  prometheus:
    enabled: true
```

### Prometheus の永続ストレージの設定

Backend. **metrics.prometheus.storage.persistent** で定義された追加のパラメーターを使用して、ストレージクラスやボリュームサイズなどの Prometheus の永続ストレージオプションを設定します。

**StorageClass** を使用してバックエンドストレージクラスを定義します。このパラメーターを設定しない場合、Service Telemetry Operator は Red Hat OpenShift Container Platform クラスターのデフォルトのストレージクラスを使用します。

Pvc **StorageRequest** パラメーターを使用して、ストレージ要求を満たすために必要な最小ボリュームサイズを定義します。ボリュームを静的に定義すると、要求よりも大きなボリュームサイズが使用される可能性があります。デフォルトで、Service Telemetry Operator はボリュームサイズ **20G**（20 ギガバイト）を要求します。

### 手順

- 利用可能なストレージクラスを一覧表示します。

```
$ 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
```

- **ServiceTelemetry** オブジェクトを設定します。

```
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
```

### ElasticSearch のイベントのストレージバックエンドとしての有効化

ElasticSearch をイベントのストレージバックエンドとして有効にするには、**ServiceTelemetry** オブジェクトを設定する必要があります。

### 手順

- **ServiceTelemetry** オブジェクトを設定します。

```

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

```

#### ElasticSearch の永続ストレージの設定

Backends. **events.elasticsearch.storage.persistent** で定義された追加のパラメーターを使用して、ストレージクラスやボリュームサイズなどの ElasticSearch の永続ストレージオプションを設定します。

**StorageClass** を使用してバックエンドストレージクラスを定義します。このパラメーターを設定しない場合、Service Telemetry Operator は Red Hat OpenShift Container Platform クラスターのデフォルトのストレージクラスを使用します。

Pvc **StorageRequest** パラメーターを使用して、ストレージ要求を満たすために必要な最小ボリュームサイズを定義します。ボリュームを静的に定義すると、要求よりも大きなボリュームサイズが使用される可能性があります。デフォルトで、Service Telemetry Operator はボリュームサイズ **20Gi** (20 ギガバイト) を要求します。

#### 手順

- 利用可能なストレージクラスを一覧表示します。

```

$ 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

```

- **ServiceTelemetry** オブジェクトを設定します。

```

apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
spec:
  backends:
  events:
    elasticsearch:
      enabled: true
    storage:
      strategy: persistent

```



```

persistent:
  storageClass: standard-csi
  pvcStorageRequest: 50G

```

### clouds パラメーター

**clouds** パラメーターを使用して、デプロイされる Smart Gateway オブジェクトを提議し、STF のインスタンスに接続する、複数の監視対象のクラウド環境にインターフェースを提供されます。サポートするバックエンドが利用可能な場合に、デフォルトのクラウド設定のメトリクスおよびイベント Smart Gateway が作成されます。デフォルトで、Service Telemetry Operator は **cloud1** の Smart Gateway を作成します。

クラウドオブジェクトの一覧を作成して、定義されたクラウドに作成される Smart Gateway を制御できます。各クラウドはデータタイプとコレクターで構成されます。データタイプは **metrics** または **events** イベントです。各データタイプは、コレクターの一覧、メッセージバスサブスクリプションアドレス、およびデバッグを有効にするパラメーターで構成されます。メトリックに使用できるコレクターは、**collectd**、**ceilometer**、および **sensubility** です。イベントで利用可能なコレクターは **collectd** および **ceilometer** です。これらのコレクターの各サブスクリプションアドレスが、クラウド、データタイプ、およびコレクターの組み合わせに対して一意であることを確認します。

デフォルトの **cloud1** 設定は、特定のクラウドインスタンスの collectd、Ceilometer、および Sensubility データコレクターのメトリクスおよびイベントのサブスクリプションおよびデータストレージを提供する以下の **ServiceTelemetry** オブジェクトによって表されます。

```

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

```

**clouds** パラメーターの各項目はクラウドインスタンスを表します。クラウドインスタンスは、**name**、**metrics**、および **events** の 3 つの最上位のパラメーターで構成されます。**metrics** および **events** パラメーターは、対象のデータタイプのストレージに対応するバックエンドを表します。**collectors** パラメーターは、2 つの必須パラメーター **collectorType** と **subscriptionAddress** で構成されるオブジェクトの一覧を指定し、これらは Smart Gateway のインスタンスを表します。**collectorType** パラメーターは、collectd、Ceilometer、または Sensubility のいずれかによって収集されるデータを指定します。**subscriptionAddress** パラメーターは、Smart Gateway がサブスクライブする AMQ Interconnect アドレスを提供します。

**collectors** パラメーター内でオプションのブール値パラメーター **debugEnabled** を使用して、実行中の Smart Gateway Pod で追加のコンソールのデバッグを有効にすることができます。

### 追加リソース

- デフォルトの Smart Gateway の削除に関する詳細は、「[デフォルトの Smart Gateway の削除](#)」を参照してください。
- 複数のクラウドを設定する方法は、「[複数クラウドの設定](#)」を参照してください。

### alerting パラメーター

**alerting** パラメーターを使用して、Alertmanager インスタンスの作成とストレージバックエンドの設定を制御します。デフォルトでは **alerting** は有効になっています。詳細は、「[Service Telemetry Framework でのアラート](#)」を参照してください。

### graphing パラメーター

**graphing** パラメーターを使用して Grafana インスタンスの作成を制御します。デフォルトでは、**graphing** は無効になっています。詳細は、「[Service Telemetry Framework でのダッシュボード](#)」を参照してください。

### highAvailability パラメーター

**highAvailability** パラメーターを使用して複数の STF コンポーネントコピーのインスタンス化を制御し、失敗または再スケジュールされたコンポーネントの復旧時間を短縮します。デフォルトでは、**highAvailability** は無効になっています。詳細は、「[高可用性](#)」を参照してください。

### transports パラメーター

STF デプロイメントに対するメッセージバスの有効化を制御するには、**transports** パラメーターを使用します。AMQ Interconnect は、現在唯一サポート対象となっているトランスポートです。デフォルトでは、**qdr** トランスポートが有効です。

## 3.3. RED HAT OPENSIFT CONTAINER PLATFORM 環境からの SERVICE TELEMETRY FRAMEWORK の削除

STF 機能を必要としなくなった場合に、Red Hat OpenShift Container Platform 環境から Service Telemetry Framework (STF) を削除します。

### 手順

1. [namespace](#) を削除します。
2. [カタログソース](#) を削除します。

#### 3.3.1. namespace の削除

Red Hat OpenShift Container Platform から STF の操作リソースを削除するには、[namespace](#) を削除します。

### 手順

1. **oc delete** コマンドを実行します。

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

2. リソースが [namespace](#) から削除されていることを確認します。

```
$ oc get all
No resources found.
```

### 3.3.2. CatalogSource の削除

Service Telemetry Framework (STF) を再びインストールしない場合には、CatalogSource を削除します。CatalogSource を削除すると、STF に関連する PackageManifest は Operator Lifecycle Manager カタログから自動的に削除されます。

#### 手順

1. インストールの処理時に OperatorHub.io Community Catalog Source を有効にし、このカタログソースがなくなったら削除します。

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

#### 追加リソース

OperatorHub.io Community Catalog Source の詳細は、[「Service Telemetry Framework の Red Hat OpenShift Container Platform 環境へのデプロイ」](#) を参照してください。

## 第4章 サービス TELEMETRY フレームワーク向けの RED HAT OPENSTACK PLATFORM の設定

メトリクス、イベント、またはその両方のコレクション、および Service Telemetry Framework (STF) ストレージドメインに送信するには、Red Hat OpenStack Platform (`{OpenStackShort}`) オーバークラウドを設定して、データ収集とトランスポートを有効にする必要があります。

STF は単一クラウドと複数のクラウドの両方をサポートすることができます。単一のクラウドインストール用に設定された `{OpenStackShort}` および STF のデフォルト設定。

- デフォルト設定の単一の `{OpenStackShort}` オーバークラウドデプロイメントについては、「[Service Telemetry Framework 向けの Red Hat OpenStack Platform オーバークラウドのデプロイ](#)」を参照してください。
- `{OpenStackShort}` のインストールおよび設定 STF で複数のクラウドを計画するには、「[複数クラウドの設定](#)」を参照してください。
- `{OpenStackShort}` オーバークラウドデプロイメントの一環として、お使いの環境で追加機能を設定する必要がある場合があります。
  - 分散コンピュートノード (DCN) やスパイン/リーフなど、ルーティング対応 L3 ドメインを使用する `{OpenStackShort}` クラウドノードで、データ収集と STF への転送をデプロイするには、「[標準以外のネットワークトポロジーへのデプロイ](#)」を参照してください。
  - Gnocchi と STF の両方にメトリックを送信する場合は「[Gnocchi および Service Telemetry Framework へのメトリックの送信](#)」を参照してください。

### 4.1. SERVICE TELEMETRY FRAMEWORK 向けの RED HAT OPENSTACK PLATFORM オーバークラウドのデプロイ

Red Hat OpenStack Platform (`{OpenStackShort}`) オーバークラウドを設定するには、データコレクターと、Service Telemetry Framework (STF) へのデータトランスポートを設定し、オーバークラウドをデプロイする必要があります。

#### 手順

1. [AMQ Interconnect ルートアドレスの取得](#)
2. [STF のベース設定の作成](#)
3. [オーバークラウドの STF 接続の設定](#)
4. [オーバークラウドのデプロイ](#)
5. [クライアント側のインストールの検証](#)

#### 追加リソース

- AMQ Interconnect からデータを収集するには、[amqp1 プラグイン](#) を参照してください。

#### 4.1.1. AMQ Interconnect ルートアドレスの取得

Service Telemetry Framework (STF) 向けに Red Hat OpenStack Platform (`{OpenStackShort}`) オープンクラウドを設定する場合に、STF 接続ファイルに AMQ Interconnect ルートアドレスを指定する必要があります。

## 手順

1. Red Hat OpenShift Container Platform 環境にログインします。
2. **service-telemetry** プロジェクトで、AMQ Interconnect ルートアドレスを取得します。

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

### 4.1.2. STF のベース設定の作成

Service Telemetry Framework (STF) と互換性があるデータ収集とトランスポートを提供するようにベースパラメーターを設定するには、デフォルトのデータ収集値を定義するファイルを作成する必要があります。

## 手順

1. Red Hat OpenStack Platform (`{OpenStackShort}`) アンダークラウドに **stack** ユーザーとしてログオンします。
2. `/home/stack` ディレクトリーに **enable-stf.yaml** という名前の設定ファイルを作成します。

### 重要

**EventPipelinePublishers** および **PipelinePublishers** を空白の一覧として設定すると、Gnocchi や Panko などの `{OpenStackShort}` のレガシー Telemetry コンポーネントにイベントやメトリックデータが渡されません。データを追加のパイプラインに送信する必要がある場合に、**ExtraConfig** で指定されるように Ceilometer のポーリングを 30 秒間隔で行うと、レガシーコンポーネントに過剰な負荷をかける可能性があるため、間隔を **300** などの大きな値に増やす必要があります。ポーリング間隔の値を増やすと、STF の Telemetry 解決が少なくなります。

STF および Gnocchi を使用して Telemetry のコレクションを有効にする場合は、「[Gnocchi および Service Telemetry Framework へのメトリックの送信](#)」を参照してください。

```
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
CollectdSensubilityResultsChannel: sensubility/telemetry

# 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.*
  - perf.*
  - port
  - port.*
  - switch
  - switch.*
  - storage.*
  - volume.*

# to avoid filling the memory buffers if disconnected from the message bus
collectd::plugin::amqp1::send_queue_limit: 50

# 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. オーバークラウドの STF 接続の設定

Service Telemetry Framework (STF) 接続を設定するには、オーバークラウド用の AMQ Interconnect の接続設定など、ファイルを STF デプロイメントに対して作成する必要があります。STF でイベントコレクションおよび対象イベントのストレージを有効にして、オーバークラウドをデプロイします。デフォルト設定は、デフォルトのメッセージバスピックを使用して単一のクラウドインスタンスに対して指定されます。複数のクラウドデプロイメントの設定については、「[複数クラウドの設定](#)」を参照してください。

#### 要件

- AMQ Interconnect ルートアドレスを取得します。詳細は、「[AMQ Interconnect ルートアドレスの取得](#)」を参照してください。

#### 手順

1. {OpenStackShort} アンダークラウドに **stack** ユーザーとしてログインします。
2. /home/stack ディレクトリーに **stf-connectors.yaml** という設定ファイルを作成します。
3. **stf-connectors.yaml** ファイルで、オーバークラウド上の AMQ Interconnect を STF デプロイメントに接続するように **MetricsQdrConnectors** アドレスを設定します。
  - **host** パラメーターは、「[AMQ Interconnect ルートアドレスの取得](#)」で取得した **HOST/PORT** の値に置き換えます。

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

  MetricsQdrSSLProfiles:
    - name: sslProfile
```

### 4.1.4. オーバークラウドのデプロイ

必要な環境ファイルでオーバークラウドをデプロイまたは更新すると、データが収集されて、Service Telemetry Framework (STF) に送信されます。

#### 手順

1. Red Hat OpenStack Platform ({OpenStackShort}) アンダークラウドに **stack** ユーザーとしてログインします。
2. source コマンドで認証ファイルを読み込みます。



```
[stack@undercloud-0 ~]$ source stackrc
```

```
(undercloud) [stack@undercloud-0 ~]$
```

3. 以下のファイルを {OpenStackShort} director デプロイメントに追加して、データ収集および AMQ Interconnect を設定します。

- collectd Telemetry およびイベントが STF に送信されることを確認する **collectd-write-qdr.yaml** ファイル
- Ceilometer Telemetry およびイベントが STF に送信されることを確認する **ceilometer-write-qdr.yaml** ファイル
- メッセージバスが有効で、STF メッセージバスルーターに接続されていることを確認する **qdr-edge-only.yaml** ファイル
- デフォルト値が正しく設定されていることを確認する **enable-stf.yaml** 環境ファイル
- STF への接続を定義する **stf-connectors.yaml** 環境ファイル

```
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud deploy <other_arguments>
--templates /usr/share/openstack-tripleo-heat-templates \
--environment-file <...other_environment_files...> \
--environment-file /usr/share/openstack-tripleo-heat-templates/environments/metrics/ceilometer-write-qdr.yaml \
--environment-file /usr/share/openstack-tripleo-heat-templates/environments/metrics/collectd-write-qdr.yaml \
--environment-file /usr/share/openstack-tripleo-heat-templates/environments/metrics/qdr-edge-only.yaml \
--environment-file /home/stack/enable-stf.yaml \
--environment-file /home/stack/stf-connectors.yaml
```

4. オーバークラウドをデプロイします。

#### 4.1.5. クライアント側のインストールの検証

Service Telemetry Framework (STF) ストレージドメインからデータ収集を検証するには、配信されたデータに対してデータソースをクエリーします。Red Hat OpenStack Platform ({OpenStackShort} デプロイメントの個別ノードを検証するには、SSH を使用してコンソールに接続します。

#### ヒント

一部の Telemetry データは、{OpenStackShort} にアクティブなワークロードがある場合にのみ利用できます。

#### 手順

1. オーバークラウドノード (例: controller-0) にログインします。
2. **metrics\_qdr** コンテナがノードで実行されていることを確認します。

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



3. AMQ Interconnect が実行されている内部ネットワークアドレスを返します (ポート **5666** でリッスンする **172.17.1.44** など)。

```
$ sudo podman exec -it metrics_qdr cat /etc/qpid-dispatch/qdrouterd.conf

listener {
  host: 172.17.1.44
  port: 5666
  authenticatePeer: no
  saslMechanisms: ANONYMOUS
}
```

4. ローカル AMQ Interconnect への接続一覧を返します。

```
$ 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
```

以下の 4 つの接続があります。

- STF への送信接続
- ceilometer からの受信接続
- collectd からの受信接続
- **qdstat** クライアントからの受信接続  
STF の送信接続は **MetricsQdrConnectors** ホストパラメーターに提供され、STF ストレージドメインのルートとなります。他のホストは、この AMQ Interconnect に対するクライアント接続の内部ネットワークアドレスです。

5. メッセージが配信されていることを確認するには、リンクを一覧表示してメッセージ配信の **deliv** 列に **\_edge** アドレスを表示します。

```
$ sudo podman exec -it metrics_qdr qdstat --bus=172.17.1.44:5666 --links

Router Links
 type dir conn id id peer class addr          phs cap pri undel unsett deliv
```

```

presett psdrop acc rej rel  mod delay rate
=====
=====
=====
endpoint out 1 5 local _edge 250 0 0 0 2979926 0 0
0 0 2979926 0 0 0
endpoint in 1 6 250 0 0 0 0 0 0 0 0
0 0 0 0
endpoint in 1 7 250 0 0 0 0 0 0 0 0
0 0 0 0
endpoint out 1 8 250 0 0 0 0 0 0 0 0
0 0 0 0
endpoint in 1 9 250 0 0 0 0 0 0 0 0
0 0 0 0
endpoint out 1 10 250 0 0 0 911 911 0 0
0 0 0 911 0
endpoint in 1 11 250 0 0 0 0 911 0 0
0 0 0 0 0
endpoint out 12 32 local temp.ISY6Mccol4J2Kp 250 0 0 0 0 0
0 0 0 0 0 0 0
endpoint in 16 41 250 0 0 0 2979924 0 0
0 0 2979924 0 0 0
endpoint in 912 1834 mobile $management 0 250 0 0 0 1 0
0 1 0 0 0 0 0
endpoint out 912 1835 local temp.9Ok2resl9tmt+CT 250 0 0 0 0
0 0 0 0 0 0 0
    
```

- 6. {OpenStackShort} ノードから STF へのアドレスを一覧表示するには、Red Hat OpenShift Container Platform に接続して AMQ Interconnect Pod 名を取得し、接続を一覧表示します。利用可能な AMQ Interconnect Pod を一覧表示します。

```

$ oc get pods -l application=default-interconnect

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

- 7. Pod に接続し、既知の接続を一覧表示します。この例では、OpenStackShort} ノードからの edge 接続として、接続 id 22、23、および 24 の 3 つがあります。

```

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

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

Connections
id host container role dir security
authentication tenant last dlv uptime
=====
=====
=====
5 10.129.0.110:48498 bridge-3f5 edge in no-security
anonymous-user 000:00:00:02 000:17:36:29
6 10.129.0.111:43254 rcv[default-cloud1-ceil-meter-smartgateway-58f885c76d-xmxwn]
    
```

```

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

```

8. ネットワークによって配信されるメッセージ数を表示するには、各アドレスを **oc exec** コマンドで使用します。

```

$ 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. GNOCCHI および SERVICE TELEMETRY FRAMEWORK へのメトリックの送信

Service Telemetry Framework (STF) および Gnocchi にメトリクスを同時に送信するには、環境ファイルをデプロイメントに追加して、追加のパブリッシャーを有効にする必要があります。

### 要件

- オーバークラウドから STF への AMQ Interconnect の接続設定が含まれるファイルを作成している。詳細は、「[オーバークラウドの STF 接続の設定](#)」を参照してください。

### 手順

1. **/home/stack** ディレクトリーに **gnocchi-connectors.yaml** という名前の環境ファイルを作成します。

```

resource_registry:
  OS::TripleO::Services::GnocchiApi: /usr/share/openstack-tripleo-heat-
templates/deployment/gnocchi/gnocchi-api-container-puppet.yaml
  OS::TripleO::Services::GnocchiMetricd: /usr/share/openstack-tripleo-heat-
templates/deployment/gnocchi/gnocchi-metricd-container-puppet.yaml
  OS::TripleO::Services::GnocchiStatsd: /usr/share/openstack-tripleo-heat-
templates/deployment/gnocchi/gnocchi-statsd-container-puppet.yaml
  OS::TripleO::Services::AodhApi: /usr/share/openstack-tripleo-heat-
templates/deployment/aodh/aodh-api-container-puppet.yaml
  OS::TripleO::Services::AodhEvaluator: /usr/share/openstack-tripleo-heat-
templates/deployment/aodh/aodh-evaluator-container-puppet.yaml
  OS::TripleO::Services::AodhNotifier: /usr/share/openstack-tripleo-heat-
templates/deployment/aodh/aodh-notifier-container-puppet.yaml
  OS::TripleO::Services::AodhListener: /usr/share/openstack-tripleo-heat-
templates/deployment/aodh/aodh-listener-container-puppet.yaml

parameter_defaults:
  CeilometerEnableGnocchi: true
  CeilometerEnablePanko: false
  GnocchiArchivePolicy: 'high'
  GnocchiBackend: 'rbd'
  GnocchiRbdPoolName: 'metrics'

  EventPipelinePublishers: ['gnocchi://?filter_project=service']
  PipelinePublishers: ['gnocchi://?filter_project=service']

```

2. デプロイメントコマンドに環境ファイル **gnocchi-connectors.yaml** を追加しま  
す。<other\_arguments> は、お使いの環境に適用されるファイルに置き換えます。

```

$ openstack overcloud deploy _<other_arguments>_
--templates /usr/share/openstack-tripleo-heat-templates \
--environment-file _<...other_environment_files...>_ \
--environment-file /usr/share/openstack-tripleo-heat-
templates/environments/metrics/ceilometer-write-qdr.yaml \
--environment-file /usr/share/openstack-tripleo-heat-
templates/environments/metrics/collectd-write-qdr.yaml \
--environment-file /usr/share/openstack-tripleo-heat-templates/environments/metrics/qdr-
edge-only.yaml \
--environment-file /home/stack/enable-stf.yaml \
--environment-file /home/stack/stf-connectors.yaml \
--environment-file /home/stack/gnocchi-connectors.yaml

```

3. 設定が正常に行われたことを確認するには、コントローラーノードの **/var/lib/config-  
data/puppet-generated/ceilometer/etc/ceilometer/pipeline.yaml** ファイルの内容を確認しま  
す。ファイルの **publishers** セクションに、**notifier** と **Gnocchi** の両方に関する情報が含まれ  
ていることを確認します。

```

sources:
  - name: meter_source
  meters:
    - "*"
  sinks:
    - meter_sink
sinks:
  - name: meter_sink

```

```
publishers:
- gnocchi://?filter_project=service
- notifier://172.17.1.35:5666/?driver=amqp&topic=metering
```

### 4.3. 標準以外のネットワークトポロジーへのデプロイ

ノードがデフォルトの **InternalApi** ネットワークとは別のネットワーク上にある場合は、AMQ Interconnect がデータを Service Telemetry Framework (STF) サーバーインスタンスに転送できるように、設定の調整を行う必要があります。以下のシナリオは、スパイン/リーフ型または DCN トポロジーでは標準的なものです。DCN の設定の詳細は、『[スパイン/リーフ型ネットワーク](#)』を参照してください。

Red Hat OpenStack Platform({OpenStackShort})16.1 と STF を使用し、Ceph、ブロック、またはオブジェクトストレージノードを監視する予定の場合には、スパイン/リーフ型および DCN ネットワーク設定に加える設定変更と同様の設定変更を行う必要があります。Ceph ノードを監視するには、**CephStorageExtraConfig** パラメーターを使用して、AMQ Interconnect および collectd 設定ファイルに読み込むネットワークインターフェースを定義します。

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

同様に、環境が Block および Object Storage ロールを使用する場合は、**BlockStorageExtraConfig** および **ObjectStorageExtraConfig** パラメーターを指定する必要があります。

スパイン/リーフ型トポロジーをデプロイするには、ロールとネットワークを作成し、それらのネットワークを利用可能なロールに割り当てる必要があります。{OpenstackShort} デプロイメントで STF のデータ収集およびトランスポートを設定する場合には、ロールのデフォルトのネットワークは **InternalApi** になります。Ceph、ブロックおよびオブジェクトストレージロールの場合には、デフォルトのネットワークは **Storage** です。スパイン/リーフ型構成により、異なるネットワークが異なるリーフグループに割り当てられ、その名前は通常一意となるので、{OpenStackShort} 環境ファイルの **parameter\_defaults** セクションで追加の設定が必要になります。

#### 手順

- それぞれのリーフロールで利用可能なネットワーク名ネットワーク名の定義例は、『[Spine Leaf Networking](#)』の「[ネットワークデータファイルの作成](#)」を参照してください。Leaf Group (ロール) の作成と、これらのグループへのネットワークの割り当ての詳細は、『[Spine Leaf Networking](#)』の「[Creating a roles data file](#)」を参照してください。
- 各リーフロールの **ExtraConfig** セクションに以下の設定例を追加します。以下の例では、**internal\_api\_subnet** はネットワーク定義の **name\_lower** パラメーターで定義した値 (Leaf 0 の名前に **\_subnet** が追加されている) で、**ComputeLeaf0** leaf ロールの接続先のネットワークです。この場合、ネットワーク ID 0 のネットワーク ID はリーフ 0 の Compute ロールに対応し、デフォルトの内部 API ネットワーク名とは異なる値を表します。**ComputeLeaf0** leaf ロールには、hiera ルックアップを実行して collectd AMQP ホストパラメーターに割り当てるネットワークインターフェースを決定するために、追加の設定を指定します。AMQ Interconnect リスナーアドレスパラメーターで同じ設定を実行します。

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

通常、追加のリーフロールは、`_subnet` を `_leafN` に置き換えます。`N` は、リーフの一意識別子です。

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

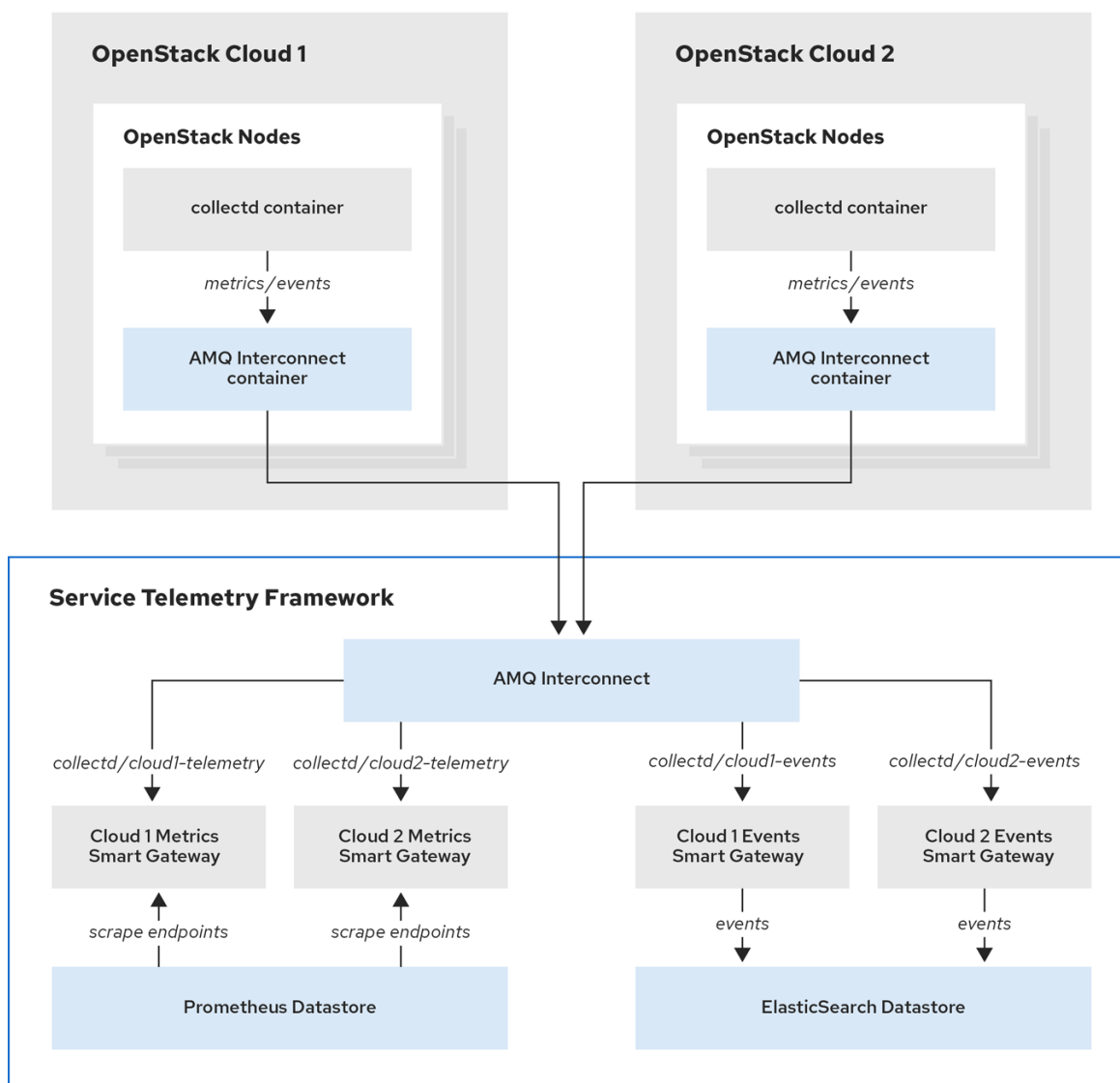
以下の構成例は、CephStorage リーフロール上にあります。

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

## 4.4. 複数クラウドの設定

Service Telemetry Framework (STF) の単一インスタンスをターゲットにするように複数の Red Hat OpenStack Platform (`{OpenStackShort}`) クラウドを設定できます。複数のクラウドを設定する場合には、クラウドはすべて独自で一意的メッセージバストピックでメトリクスおよびイベントを送信する必要があります。STF デプロイメントでは、Smart Gateway インスタンスは、これらのトピックをリッスンして、共通のデータストアに情報を保存します。データストレージドメインの Smart Gateway によって保存されるデータは、各 Smart Gateway が作成するメタデータを使用してフィルターされます。

図4.12 つの {OpenStackShort} クラウドが STF に接続する



65\_OpenStack\_0120

複数のクラウドシナリオ用に {OpenStackShort} オーバークラウドを設定するには、以下のタスクを実行します。

1. 各クラウドに使用する AMQP アドレス接頭辞を計画します。詳細は、「[AMQP アドレス接頭辞の計画](#)」を参照してください。
2. 各クラウドが対応するアドレスの接頭辞をリッスンできるようにメトリクスおよびイベントコンシューマーの Smart Gateway をデプロイします。詳細は、「[Smart Gateway のデプロイ](#)」を参照してください。
3. 各クラウドに一意的ドメイン名を設定します。詳細は、「[一意のクラウドドメインの設定](#)」を参照してください。
4. STF のベース設定を作成します。詳細は、「[STF のベース設定の作成](#)」を参照してください。
5. メトリクスとイベントを正しいアドレスの STF に送信するように各クラウドを設定します。詳細は、「[複数クラウドの Red Hat OpenStack Platform 環境ファイルの作成](#)」を参照してください。

#### 4.4.1. AMQP アドレス接頭辞の計画

デフォルトでは、Red Hat OpenStack Platform (`{OpenStackShort}`) ノードは、`collectd` および `Ceilometer` の 2 つのデータコレクターを使用してデータを受信します。`collectd-sensubility` プラグインには一意のアドレスが必要です。これらのコンポーネントは、Telemetry データまたは通知をそれぞれの AMQP アドレス (例: **collectd/telemetry**) に送信します。STF Smart Gateway は、データのこれらの AMQP アドレスでリッスンします。複数のクラウドをサポートし、監視データを生成するクラウドを特定するには、各クラウドがデータを一意のアドレスに送信するように設定します。クラウド識別子の接頭辞をアドレスの 2 番目の部分に追加します。以下の一覧では、アドレスおよび識別子の例を紹介します。

- **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**

#### 4.4.2. Smart Gateway のデプロイ

各クラウドのデータ収集タイプごとに (`collectd` メトリクスに 1 つ、`collectd` イベントに 1 つ、`Ceilometer` メトリクスに 1 つ、`Ceilometer` イベントに 1 つ、`collectd-sensubility` メトリクスに 1 つ) `Smart Gateway` をデプロイする必要があります。対応するクラウドに定義する AMQP アドレスをリッスンするように、各 `Smart Gateway` を設定します。`Smart Gateway` を定義するには、**ServiceTelemetry** マニフェストに **clouds** パラメーターを設定します。

STF の初回デプロイ時に、単一クラウドの初期 `Smart Gateway` を定義する `Smart Gateway` マニフェストが作成されます。複数のクラウドサポートに `Smart Gateway` をデプロイする場合には、メトリクスおよび各クラウドのイベントデータを処理するデータ収集タイプごとに、複数の `Smart Gateway` をデプロイします。最初の `Smart Gateway` は、以下のサブスクリプションアドレスを使用して **cloud1** で定義されます。

collector	type	デフォルトのサブスクリプションアドレス
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

## 要件

- クラウド命名スキームを決定している。命名スキームの決定の詳細は、「[AMQP アドレス接頭辞の計画](#)」を参照してください。
- クラウドオブジェクトの一覧を作成している。**clouds** パラメーターのコンテンツ作成の詳細は、「[clouds パラメーター](#)」を参照してください。

## 手順

1. Red Hat OpenShift Container Platform にログインします。
2. **service-telemetry** namespace に切り替えます。

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

3. **default** の ServiceTelemetry オブジェクトを編集し、設定で **clouds** パラメーターを追加します。



### 警告

クラウド名が長くなると、Pod 名の最大長 63 文字を超える可能性があります。**ServiceTelemetry** 名の **default** と **clouds.name** の組み合わせが 19 文字を超えないようにしてください。クラウド名には、- などの特殊文字を含めることはできません。クラウド名を英数字 (a-z、0-9) に制限します。

トピックアドレスには文字の制限がなく、**clouds.name** の値とは異なる場合があります。

```
$ 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. ServiceTelemetry オブジェクトを保存します。
5. 各 Smart Gateway が実行されていることを確認します。Smart Gateway の数によっては、数分かかることがあります。

```
$ 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-xmxwn	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

#### 4.4.3. デフォルトの Smart Gateway の削除

複数のクラウドに Service Telemetry Framework (STF) を設定したら、デフォルトの Smart Gateway が使用されなくなった場合に、そのゲートウェイを削除できます。Service Telemetry Operator は、作成済みではあるが、オブジェクトの ServiceTelemetry **clouds** 一覧に表示されなくなった **SmartGateway** オブジェクトを削除できます。**clouds** パラメーターで定義されていない SmartGateway オブジェクトの削除を有効にするには、**ServiceTelemetry** マニフェストで **cloudsRemoveOnMissing** パラメーターを **true** に設定する必要があります。

#### ヒント

Smart Gateway をデプロイしない場合は、**clouds: []** パラメーターを使用して空のクラウド一覧を定義します。



### 警告

**cloudsRemoveOnMissing** パラメーターはデフォルトで無効にされています。**cloudsRemoveOnMissing** パラメーターが有効な場合には、現在の namespace で手動で作成された **SmartGateway** オブジェクトを削除するとそのオブジェクトは復元できません。

### 手順

1. Service Telemetry Operator が管理するクラウドオブジェクトの一覧で **clouds** パラメーターを定義します。詳細は、「[clouds パラメーター](#)」を参照してください。
2. ServiceTelemetry オブジェクトを編集し、**cloudsRemoveOnMissing** パラメーターを追加します。

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

3. 変更を保存します。
4. Operator が Smart Gateways を削除していることを確認します。これは、Operator による変更の調整に数分かかる場合があります。

```
$ oc get smartgateways
```

#### 4.4.4. 一意のクラウドドメインの設定

Red Hat OpenStack Platform (`{OpenStackShort}`) から Service Telemetry Framework (STF) への AMQ Interconnect ルーター接続が一意的で、競合しないようにするには、**CloudDomain** パラメーターを設定します。

### 手順

1. 新しい環境ファイルを作成します (例: **hostnames.yaml**)。
2. 以下の例に示すように、環境ファイルで **CloudDomain** パラメーターを設定します。

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

3. 新しい環境ファイルをデプロイメントに追加します。詳しい情報は、『[Overcloud Parameters](#)』の「[複数クラウドの Red Hat OpenStack Platform 環境ファイルの作成](#)」および「[Core overcloud parameters](#)」を参照してください。

#### 4.4.5. 複数クラウドの Red Hat OpenStack Platform 環境ファイルの作成

元のクラウドに従ってトラフィックにラベルを付けるには、クラウド固有のインスタンス名で設定を作成する必要があります。`stf-connectors.yaml` ファイルを作成

し、`CeilometerQdrEventsConfig`、`CeilometerQdrMetricsConfig`、および `CollectdAmqpInstances` の値を調整して AMQP アドレスのプレフィックススキームと一致するようにします。



#### 注記

コンテナのヘルスおよび API ステータスのモニタリングを有効にしている場合は、`CollectdSensubilityResultsChannel` パラメーターも変更する必要があります。詳細については、「[Red Hat OpenStack Platform API のステータスおよびコンテナ化されたサービスの健全性](#)」を参照してください。

#### 要件

- クラウドオブジェクトの一覧を作成している。clouds パラメーターのコンテンツ作成の詳細は、「[クラウド設定パラメーター](#)」を参照してください。
- AMQ Interconnect ルートアドレスを取得している。詳細は、「[AMQ Interconnect ルートアドレスの取得](#)」を参照してください。
- STF のベース設定を作成済みである。詳細は、「[STF のベース設定の作成](#)」を参照してください。
- 一意のドメイン名の環境ファイルを作成している。詳細は、「[一意のクラウドドメインの設定](#)」を参照してください。

#### 手順

1. Red Hat OpenStack Platform アンダークラウドに `stack` ユーザーとしてログオンします。
2. `/home/stack` ディレクトリーに `stf-connectors.yaml` という設定ファイルを作成します。
3. `stf-connectors.yaml` ファイルで、オーバークラウドデプロイメント上の AMQ Interconnect をに接続するように `MetricsQdrConnectors` アドレスを設定します。`CeilometerQdrEventsConfig`、`CeilometerQdrMetricsConfig`、`CollectdAmqpInstances`、および `CollectdSensubilityResultsChannel` トピックの値を、このクラウドデプロイメントに必要な AMQP アドレスに一致するように設定します。

#### `stf-connectors.yaml`

```
resource_registry:
  OS::TripleO::Services::Collectd: /usr/share/openstack-tripleo-heat-
  templates/deployment/metrics/collectd-container-puppet.yaml ①

parameter_defaults:
  MetricsQdrConnectors:
    - host: stf-default-interconnect-5671-service-telemetry.apps.infra.watch ②
      port: 443
      role: edge
```

```
verifyHostname: false
sslProfile: sslProfile
```

```
MetricsQdrSSLProfiles:
- name: sslProfile
```

```
CeilometerQdrEventsConfig:
driver: amqp
topic: cloud1-event ③
```

```
CeilometerQdrMetricsConfig:
driver: amqp
topic: cloud1-metering ④
```

```
CollectdAmqpInstances:
cloud1-notify: ⑤
  notify: true
  format: JSON
  presettle: false
cloud1-telemetry: ⑥
  format: JSON
  presettle: false
```

```
CollectdSensubilityResultsChannel: sensubility/cloud1-telemetry ⑦
```

- ① 複数のクラウドデプロイメント向けの **collectd-write-qdr.yaml** 環境ファイルが追加されていないので、collectd サービスを直接読み込みます。
  - ② **host** パラメーターは、「[AMQ Interconnect ルートアドレスの取得](#)」で取得した **HOST/PORT** の値に置き換えます。
  - ③ Ceilometer イベントのトピックを定義します。この値は、**anycast/ceilometer/cloud1-event.sample** のアドレス形式です。
  - ④ Ceilometer メトリクスのトピックを定義します。この値は、**anycast/ceilometer/cloud1-metering.sample** のアドレス形式です。
  - ⑤ collectd イベントのトピックを定義します。この値は、**collectd/cloud1-notify** の形式になります。
  - ⑥ collectd メトリクスのトピックを定義します。この値は、**collectd/cloud1-telemetry** の形式になります。
  - ⑦ collectd-sensubility イベントのトピックを定義します。この値は、正確な文字列形式 (**sensubility/cloud1-telemetry**) であることを確認します。
4. **stf-connectors.yaml** ファイルの命名規則が、Smart Gateway 設定の **spec.bridge.amqpUrl** フィールドと一致していることを確認します。たとえば、**CeilometerQdrEventsConfig.topic** フィールドを **cloud1-event** の値に設定します。
  5. source コマンドで認証ファイルを読み込みます。

```
[stack@undercloud-0 ~]$ source stackrc
```

```
(undercloud) [stack@undercloud-0 ~]$
```

6. 実際の環境に該当するその他の環境ファイルと共に、**openstack overcloud deploy** コマンドに **stf-connectors.yaml** ファイルおよび一意のドメイン名環境ファイル **hostnames.yaml** を追加します。



### 警告

カスタム **CollectdAmqpInstances** パラメーターで **collectd-write-qdr.yaml** ファイルを使用する場合、データはカスタムおよびデフォルトのトピックに公開されます。複数のクラウド環境では、**stf-connectors.yaml** ファイルの **resource\_registry** パラメーターの設定では collectd サービスを読み込みます。

```
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud deploy <other_arguments>
--templates /usr/share/openstack-tripleo-heat-templates \
--environment-file <...other_environment_files...> \
--environment-file /usr/share/openstack-tripleo-heat-templates/environments/metrics/ceilometer-write-qdr.yaml \
--environment-file /usr/share/openstack-tripleo-heat-templates/environments/metrics/qdr-edge-only.yaml \
--environment-file /home/stack/hostnames.yaml \
--environment-file /home/stack/enable-stf.yaml \
--environment-file /home/stack/stf-connectors.yaml
```

7. Red Hat OpenStack Platform オーバークラウドをデプロイします。

### 追加リソース

- デプロイメントの検証方法は、「[クライアント側のインストールの検証](#)」を参照してください。

#### 4.4.6. 複数クラウドからのメトリクスデータのクエリー

Prometheus に保存されるデータには、収集元の Smart Gateway に従って **service** ラベルが割り当てられます。このラベルを使用して、特定のクラウドからデータをクエリーできます。

特定のクラウドからデータをクエリーするには、関連する **service** ラベルに一致する Prometheus `promql` クエリーを使用します (例: `collectd_uptime{service="default-cloud1-coll-meter"}`)。

## 第5章 SERVICE TELEMETRY FRAMEWORK の運用機能の使用

以下の操作機能を使用して、Service Telemetry Framework (STF)に追加機能を提供できます。

- [ダッシュボードの設定](#)
- [メトリクスの保持期間の設定](#)
- [アラートの設定](#)
- [SNMP トラップの設定](#)
- [高可用性の設定](#)
- [一時ストレージの設定](#)
- [Red Hat OpenShift Container Platform でのルートの作成](#)
- [OpenStack サービスのリソース使用状況の監視](#)
- [コンテナヘルスおよび API ステータスのモニタリング](#)

### 5.1. SERVICE TELEMETRY FRAMEWORK でのダッシュボード

サードパーティーのアプリケーション Grafana を使用して、collectd および Ceilometer が各ホストノードについて収集するシステムレベルのメトリクスを可視化します。

collectd の設定の詳細は、[「Service Telemetry Framework 向けの Red Hat OpenStack Platform オペラクラウドのデプロイ」](#) を参照してください。

2つのダッシュボードを使用してクラウドを監視できます。

#### インフラストラクチャーダッシュボード

インフラストラクチャーダッシュボードを使用して、一度にノードのメトリクスを1つ表示します。ダッシュボードの左上隅からノードを選択します。

#### Cloud view ダッシュボード

クラウドビューダッシュボードを使用してパネルを表示し、サービスリソースの使用状況、API 統計、およびクラウドイベントを監視します。このダッシュボードにデータを提供するには、API ヘルスマモニタリングおよびサービスのモニタリングを有効にする必要があります。API ヘルスマモニタリングは、STF ベース設定でデフォルトで有効にされます。詳細は、[「STF のベース設定の作成」](#) を参照してください。

- API ヘルスマモニタリングに関する詳細は、[「Red Hat OpenStack Platform API のステータスおよびコンテナ化されたサービスの健全性」](#) を参照してください。
- {OpenStackShort} サービスのモニタリングの詳細は、[「Red Hat OpenStack Platform サービスのリソース使用状況」](#) を参照してください。

#### 5.1.1. ダッシュボードをホストするための Grafana の設定

Grafana はデフォルトの Service Telemetry Framework (STF) デプロイメントに含まれないので、OperatorHub.io から Grafana Operator をデプロイする必要があります。Service Telemetry Operator を使用して Grafana をデプロイすると、Grafana インスタンスとローカル STF デプロイメントのデフォルトデータソースの設定が作成されます。

## 手順

1. Red Hat OpenShift Container Platform にログインします。
2. **service-telemetry** namespace に切り替えます。

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

3. Grafana Operator をデプロイします。

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

4. Operator が正常に起動したことを確認します。コマンド出力で、**PHASE** 列の値が **Succeeded** の場合には、Operator は正常に起動されています。

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

NAME	DISPLAY	VERSION	REPLACES	PHASE
grafana-operator.v3.10.3	Grafana Operator	3.10.3	grafana-operator.v3.10.2	Succeeded

5. Grafana インスタンスを起動するには、**ServiceTelemetry** オブジェクトを作成または変更します。 **graphing.enabled** および **graphing.grafana.ingressEnabled** を **true** に設定します。

```
$ oc edit stf default
```

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
...
spec:
  ...
  graphing:
    enabled: true
  grafana:
    ingressEnabled: true
```

6. Grafana インスタンスがデプロイされていることを確認します。

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

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



- Grafana データソースが正しくインストールされていることを確認します。

```
$ oc get grafanadatasources
```

```
NAME          AGE
default-datasources 20h
```

- Grafana ルートが存在することを確認します。

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

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

### 5.1.2. デフォルトの Grafana コンテナイメージの上書き

Service Telemetry Framework (STF) のダッシュボードには、Grafana バージョン 8.1.0 以降でのみ利用できる機能が必要です。デフォルトで、Service Telemetry Operator は互換性のあるバージョンをインストールします。**graphing.grafana.baselmage** でイメージレジストリーへのイメージパスを指定して、ベース Grafana イメージを上書きできます。

#### 手順

- Grafana の正しいバージョンがあることを確認します。

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

- 実行中のイメージが 8.1.0 よりも古い場合は、ServiceTelemetry オブジェクトにパッチを適用してイメージを更新します。Service Telemetry Operator は、Grafana デプロイメントを再起動する Grafana マニフェストを更新します。

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

- 新しい Grafana Pod が存在し、**Running** の値が **STATUS** であることを確認します。

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

- 新規インスタンスが更新されたイメージを実行していることを確認します。

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

### 5.1.3. ダッシュボードのインポート

Grafana Operator は、**GrafanaDashboard** オブジェクトを作成してダッシュボードをインポートおよび管理できます。ダッシュボードのサンプルは、<https://github.com/infrawatch/dashboards> で表示できます。

## 手順

1. インフラストラクチャーダッシュボードをインポートします。

```
$ oc apply -f https://raw.githubusercontent.com/infrawatch/dashboards/master/deploy/stf-1.3/rhos-dashboard.yaml
```

```
grafanadashboard.integreatly.org/rhos-dashboard-1.3 created
```

2. クラウドダッシュボードをインポートします。



### 警告

クラウドダッシュボードの一部のパネルについては、**stf-connectors.yaml** ファイルで `collectd virt` プラグインパラメーター `hostname_format` の値を **name uuid hostname** に設定する必要があります。このパラメーターを設定しない場合には、影響を受けるダッシュボードは空のままになります。Virt プラグインの詳細は、「[collectd plugins](#)」を参照してください。

```
$ oc apply -f https://raw.githubusercontent.com/infrawatch/dashboards/master/deploy/stf-1.3/rhos-cloud-dashboard.yaml
```

```
grafanadashboard.integreatly.org/rhos-cloud-dashboard-1.3 created
```

3. クラウドイベントダッシュボードをインポートします。

```
$ oc apply -f https://raw.githubusercontent.com/infrawatch/dashboards/master/deploy/stf-1.3/rhos-cloudevents-dashboard.yaml
```

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

4. 仮想マシンダッシュボードをインポートします。

```
$ oc apply -f https://raw.githubusercontent.com/infrawatch/dashboards/master/deploy/stf-1.3/virtual-machine-view.yaml
```

```
grafanadashboard.integreatly.org/virtual-machine-view-1.3 configured
```

5. memcached ダッシュボードをインポートします。

```
$ oc apply -f https://raw.githubusercontent.com/infrawatch/dashboards/master/deploy/stf-1.3/memcached-dashboard.yaml
```

```
grafanadashboard.integreatly.org/memcached-dashboard-1.3 created
```

6. ダッシュボードが利用可能であることを確認します。

```
$ oc get grafanadashboards

NAME                AGE
memcached-dashboard-1.3    115s
rhos-cloud-dashboard-1.3   2m12s
rhos-cloudevents-dashboard 2m6s
rhos-dashboard-1.3        2m17s
virtual-machine-view-1.3   2m
```

7. Grafana ルートアドレスを取得します。

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

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

8. Web ブラウザーで、[https://<grafana\\_route\\_address>](https://<grafana_route_address>) に移動します。<grafana\_route\_address> は、直前の手順で取得した値に置き換えます。
9. ダッシュボードを表示するには、**Dashboards** と **Manage** をクリックします。

#### 5.1.4. Grafana ログイン認証情報の取得および設定

Service Telemetry Framework (STF) は、Grafana が有効な場合にデフォルトのログイン認証情報を設定します。**ServiceTelemetry** オブジェクトで認証情報を上書きできます。

##### 手順

1. Red Hat OpenShift Container Platform にログインします。
2. **service-telemetry** namespace に切り替えます。

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

3. デフォルトのユーザー名とパスワードを取得するには、Grafana オブジェクトを記述します。

```
$ oc describe grafana default
```

4. ServiceTelemetry オブジェクトを介して Grafana 管理者のユーザー名およびパスワードのデフォルト値を変更するには、**graphing.grafana.adminUser** および **graphing.grafana.adminPassword** パラメーターを使用します。

## 5.2. SERVICE TELEMETRY FRAMEWORK でのメトリクスの保持期間

Service Telemetry Framework (STF) に保存されるメトリクスのデフォルトの保持期間は 24 時間となっており、アラート目的で傾向を把握するのに十分なデータが提供されます。

長期ストレージの場合は、Thanos など、長期のデータ保持用に設計されたシステムを使用します。

##### 追加リソース

- 追加のメトリクスの保持時間に合わせて STF を調整するには、「[Service Telemetry Framework でのメトリクスの保持期間の編集](#)」を参照してください。
- Prometheus データストレージおよびストレージ領域の予測に関する推奨事項は、<https://prometheus.io/docs/prometheus/latest/storage/#operational-aspects>を参照してください。
- Thanos の詳細は、<https://thanos.io/> を参照してください。

### 5.2.1. Service Telemetry Framework でのメトリクスの保持期間の編集

追加のメトリクスの保持時間に対応するために、Service Telemetry Framework (STF) を調整できます。

#### 手順

1. Red Hat OpenShift Container Platform にログインします。
2. service-telemetry namespace に切り替えます。

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

3. ServiceTelemetry オブジェクトを編集します。

```
$ oc edit stf default
```

4. **retention: 7d** を backends.metrics.prometheus.storage の storage セクションに追加し、保持期間を7日間増やします。



#### 注記

保持期間を長く設定すると、設定された Prometheus システムからデータを取得すると、クエリーで結果の速度が遅くなる可能性があります。

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

5. 変更を保存して、オブジェクトを閉じます。

#### 追加リソース

- メトリクスの保持期間の詳細は、「[Service Telemetry Framework でのメトリクスの保持期間](#)」を参照してください。

### 5.3. SERVICE TELEMETRY FRAMEWORK でのアラート

Prometheus でアラートルールを、Alertmanager でアラートルートを作成します。Prometheus サーバーのアラートルールは、アラートを管理する Alertmanager にアラートを送信します。Alertmanager は通知をオフにしたり、アラートを集約してメール (on-call 通知システムまたはチャットプラットフォーム) で通知を送信できます。

アラートを作成するには、以下のタスクを実行します。

1. Prometheus でアラートルールを作成します。詳細は、「[Prometheus でのアラートルールの作成](#)」を参照してください。
2. Alertmanager でアラートルートを作成します。詳細は、「[Alertmanager でのアラートルートの作成](#)」を参照してください。

#### 追加リソース

Prometheus および Alertmanager のアラートまたは通知の詳細は <https://prometheus.io/docs/alerting/overview/> を参照してください。

Service Telemetry Framework (STF) で使用できるアラートのセットのサンプルを表示するには、<https://github.com/infrawatch/service-telemetry-operator/tree/master/deploy/alerts> を参照してください。

#### 5.3.1. Prometheus でのアラートルールの作成

Prometheus はアラートルールを評価し、通知をトリガーします。ルール条件が返す結果が空の場合には、条件は false になります。それ以外の場合は、ルールは true であるため、アラートがトリガーされます。

#### 手順

1. Red Hat OpenShift Container Platform にログインします。
2. **service-telemetry** namespace に切り替えます。

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

3. アラートルールを含む **PrometheusRule** オブジェクトを作成します。Prometheus Operator はルールを Prometheus に読み込みます。

```
$ oc apply -f - <<EOF
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  creationTimestamp: null
labels:
  prometheus: default
  role: alert-rules
name: prometheus-alarm-rules
namespace: service-telemetry
spec:
```

```
groups:
- name: ./openstack.rules
  rules:
- alert: Metric Listener down
  expr: collectd_qpuid_router_status < 1 1
EOF
```

**1** ルールを変更するには、`expr` パラメーターの値を編集します。

- Operator がルールを Prometheus に読み込んだことを確認するには、`curl` にアクセスできる Pod を作成します。

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

- `curl` コマンドを実行して **prometheus-operated** サービスにアクセスし、メモリーに読み込まれるルールを返します。

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

- 出力に Pod から定義された **./openstack.rules** が含まれるかなど、出力に **PrometheusRule** オブジェクトに読み込まれるルールが表示されることを確認するには、Pod を終了します。

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

- `curl` Pod を削除して環境を消去します。

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

## 追加リソース

- アラートについての詳細は、<https://github.com/coreos/prometheus-operator/blob/master/Documentation/user-guides/alerting.md>を参照してください。

### 5.3.2. カスタムアラートの設定

カスタムアラートは、「[Prometheus でのアラートルールの作成](#)」で作成した **PrometheusRule** オブジェクトに追加できます。

#### 手順

- `oc edit` コマンドを使用します。

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

- PrometheusRules** マニフェストを編集します。

3. マニフェストを保存し、終了します。

### 追加リソース

- アラートルールの設定方法は、[https://prometheus.io/docs/prometheus/latest/configuration/alerting\\_rules/](https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/) を参照してください。
- PrometheusRules オブジェクトの詳細は <https://github.com/coreos/prometheus-operator/blob/master/Documentation/user-guides/alerting.md> を参照してください。

### 5.3.3. Alertmanager でのアラートルートの作成

Alertmanager を使用して、メール、IRC、またはその他の通知チャネルなどの外部システムにアラートを送信します。Prometheus Operator は、Alertmanager 設定を Red Hat OpenShift Container Platform シークレットとして管理します。デフォルトで、Service Telemetry Framework (STF) は、受信側を持たない基本的な設定をデプロイします。

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

STF を使用してカスタム Alertmanager ルートをデプロイするには、**alertmanagerConfigManifest** パラメーターを Service Telemetry Operator に渡す必要があります。これにより、更新されたシークレットが作成され、Prometheus Operator の管理対象となります。

### 手順

1. Red Hat OpenShift Container Platform にログインします。
2. **service-telemetry** namespace に切り替えます。

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

3. STF デプロイメントの **ServiceTelemetry** オブジェクトを編集します。

```
$ oc edit stf default
```

4. 新規パラメーター **alertmanagerConfigManifest** および **Secret** オブジェクトの内容を追加し、Alertmanager の **alertmanager.yaml** 設定を定義します。



## 注記

この手順では、Service Telemetry Operator が管理するデフォルトのテンプレートを読み込みます。変更が正しく入力されていることを確認するには、値を変更して **alertmanager-default** シークレットを返し、新しい値がメモリーに読み込まれていることを確認します。たとえば、パラメーター **global.resolve\_timeout** の値を **5m** から **10m** に変更します。

```
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. 設定がシークレットに適用されていることを確認します。

```
$ 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'
```



- 設定が Alertmanager にロードされたことを確認するには、**curl** にアクセスできる Pod を作成します。

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

- alertmanager-operated** サービスに対して **curl** を実行し、ステータスと **configYAML** の内容を取得し、提供された設定が Alertmanager の設定と一致することを確認します。

```
[ root@curl:/ ]$ curl alertmanager-operated:9093/api/v1/status

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

- configYAML** フィールドに予想される変更が含まれることを確認します。

- Pod を終了します。

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

- 環境を消去するには、**curl** Pod を削除します。

```
$ oc delete pod curl

pod "curl" deleted
```

## 追加リソース

- Red Hat OpenShift Container Platform シークレットおよび Prometheus Operator の詳細は、「[Alerting](#)」を参照してください。

## 5.4. SNMP トラップの設定

Service Telemetry Framework (STF) は、SNMP トラップで通知を受信する既存のインフラストラクチャーモニタリングプラットフォームと統合できます。SNMP トラップを有効にするには、**ServiceTelemetry** オブジェクトを変更し、**snmpTraps** パラメーターを設定します。

アラートの設定の詳細は、「[Service Telemetry Framework でのアラート](#)」を参照してください。

### 要件

- アラートを送信する SNMP トラップレシーバーの IP アドレスまたはホスト名が分かっている。

### 手順

- SNMP トラップを有効にするには、**ServiceTelemetry** オブジェクトを変更します。

```
$ oc edit stf default
```

2. **alerting.alertmanager.receivers.snmpTraps** パラメーターを設定します。

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

3. **target** の値は、SNMP トラップレシーバーの IP アドレスまたはホスト名に設定するようにしてください。

## 5.5. 高可用性

高可用性により、Service Telemetry Framework (STF) はコンポーネントサービスの障害から迅速に復旧できます。Red Hat OpenShift Container Platform は、ワークロードをスケジュールするノードが利用可能な場合に、障害のある Pod を再起動しますが、この復旧プロセスではイベントとメトリクスが失われる可能性があります。高可用性設定には、複数の STF コンポーネントのコピーが含まれており、復旧時間が約 2 秒に短縮されます。Red Hat OpenShift Container Platform ノードの障害から保護するには、3 つ以上のノードで STF を Red Hat OpenShift Container Platform クラスタにデプロイします。



### 警告

STF はまだ完全な耐障害性システムではありません。リカバリー期間中のメトリクスおよびイベントの配信は保証されません。

高可用性を有効にすると、以下のような影響があります。

- デフォルトの ElasticSearch Pod の代わりに 3 つの ElasticSearch Pod が実行されます。
- 以下のコンポーネントは、デフォルトの Pod ではなく 2 つの Pod を実行します。
  - AMQ Interconnect
  - Alertmanager
  - Prometheus
  - Events Smart Gateway
  - Metrics Smart Gateway

- これらのサービスのいずれの場合でも、失われた Pod からのリカバリ一時間が 2 秒に削減されます。

### 5.5.1. 高可用性の設定

高可用性のために Service Telemetry Framework (STF) を設定するには、Red Hat OpenShift Container Platform の ServiceTelemetry オブジェクトに **highAvailability.enabled: true** を追加します。このパラメーターはインストール時に設定できます。またはすでに STF をデプロイしている場合には、以下の手順を実行します。

#### 手順

1. Red Hat OpenShift Container Platform にログインします。
2. **service-telemetry** namespace に切り替えます。

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

3. oc コマンドを使用して ServiceTelemetry オブジェクトを編集します。

```
$ oc edit stf default
```

4. **highAvailability.enabled: true** を **spec** セクションに追加します。

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

5. 変更を保存して、オブジェクトを閉じます。

## 5.6. 一時ストレージ

一時ストレージを使用して、Red Hat OpenShift Container Platform クラスターにデータを永続的に保存せずに Service Telemetry Framework (STF) を実行できます。



### 警告

一時ストレージを使用する場合は、Pod の再起動、更新、別のノードでの再スケジューリングが発生した場合に、データが損失する可能性があります。一時ストレージは、開発またはテスト環境にのみ使用してください。

### 5.6.1. 一時ストレージの設定

一時ストレージ用に STF コンポーネントを設定するには、対応するパラメーターに **...storage.strategy: ephemeral** を追加します。たとえば、Prometheus バックエンドの一時ストレージ

を有効にするには、**backends.metrics.prometheus.storage.strategy: ephemeral** を設定します。一時ストレージの設定をサポートするコンポーネントには、**alerting.alertmanager**、**backends.metrics.prometheus**、および **backends.events.elasticsearch** が含まれます。インストール時に一時ストレージ設定を追加できます。STF をデプロイ済みの場合には、以下の手順を実行します。

## 手順

1. Red Hat OpenShift Container Platform にログインします。
2. **service-telemetry** namespace に切り替えます。

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

3. ServiceTelemetry オブジェクトを編集します。

```
$ oc edit stf default
```

4. **...storage.strategy: ephemeral** パラメーターを関連コンポーネントの **spec** セクションに追加します。

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

5. 変更を保存して、オブジェクトを閉じます。

## 5.7. RED HAT OPENSIFT CONTAINER PLATFORM でのルートの作成

Red Hat OpenShift Container Platform では、アプリケーションをルートで外部ネットワークに公開できます。詳細は、[Ingress クラスタートラフィックの設定について](#)を参照してください。

Service Telemetry Framework (STF) では、ルートはデフォルトで公開されず、STF デプロイメントの攻撃対象領域を制限します。STF にデプロイされたサービスにアクセスするには、アクセスできるように Red Hat OpenShift Container Platform でサービスを公開する必要があります。

以下の例のように、STF に公開する一般的なサービスは Prometheus です。

## 手順

1. Red Hat OpenShift Container Platform にログインします。
2. **service-telemetry** namespace に切り替えます。

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

3. **service-telemetry** プロジェクトで利用可能なサービスを一覧表示します。

```
$ oc get services
NAME                                TYPE          CLUSTER-IP      EXTERNAL-IP  PORT(S)
AGE
alertmanager-operated              ClusterIP      None             <none>
9093/TCP,9094/TCP,9094/UDP          93m
default-cloud1-ceil-meter-smartgateway ClusterIP      172.30.114.195  <none>      8081/TCP
93m
default-cloud1-coll-meter-smartgateway ClusterIP      172.30.133.180  <none>      8081/TCP
93m
default-interconnect               ClusterIP      172.30.3.241    <none>
5672/TCP,8672/TCP,55671/TCP,5671/TCP,5673/TCP 93m
ibm-auditlogging-operator-metrics   ClusterIP      172.30.216.249  <none>
8383/TCP,8686/TCP                  11h
prometheus-operated                ClusterIP      None             <none>      9090/TCP
93m
service-telemetry-operator-metrics   ClusterIP      172.30.11.66    <none>
8383/TCP,8686/TCP                  11h
smart-gateway-operator-metrics       ClusterIP      172.30.145.199  <none>
8383/TCP,8686/TCP                  11h
```

4. ポートとサービス名をメモしてルートとして公開します (例: サービスの **prometheus-operated** およびポート **9090**)。
5. **prometheus-operated** サービスを edge ルートとして公開し、セキュアでないトラフィックをポート **9090** のセキュアなエンドポイントにリダイレクトします。

```
$ oc create route edge metrics-store --service=prometheus-operated --insecure-policy="Redirect" --port=9090
route.route.openshift.io/metrics-store created
```

6. ルート用に公開される外部 DNS を検証および検索するには、**oc get route** コマンドを使用します。

```
$ oc get route metrics-store -o go-template='{{.spec.host}}'
metrics-store-service-telemetry.apps.infra.watch
```

**prometheus-operated** サービスが公開される DNS アドレスで利用可能になりました (例: <https://metrics-store-service-telemetry.apps.infra.watch>)。



## 注記

ルートのアドレスは解決可能であり、設定は環境固有である必要があります。

## 追加リソース

- Red Hat OpenShift Container Platform ネットワークの詳細は、「[ネットワークについて](#)」を参照してください。
- ルート設定の詳細は、「[ルート設定](#)」を参照してください。
- Ingress クラスタートラフィックの詳細は、「[Ingress クラスタートラフィックの設定](#)」を参照してください。

## 5.8. RED HAT OPENSTACK PLATFORM サービスのリソース使用状況

API や他のインフラストラクチャープロセスなどの Red Hat OpenStack Platform (`{OpenStackShort}`) サービスのリソース使用状況を監視して、コンピュータの電源が不足するサービスを表示し、オーバークラウドのボトルネックを特定できます。リソース使用状況の監視はデフォルトで有効にされています。

### 追加リソース

- リソース使用状況の監視を無効にするには、「[Red Hat OpenStack Platform サービスのリソース使用状況の監視の無効化](#)」を参照してください。

### 5.8.1. Red Hat OpenStack Platform サービスのリソース使用状況の監視の無効化

`{OpenStackShort}` コンテナ化されたサービスのリソース使用状況のモニタリングを無効にするには、**CollectdEnableLibpodstats** パラメーターを **false** に設定する必要があります。

#### 要件

- **stf-connectors.yaml** ファイルを作成している。詳細は、「[Service Telemetry Framework 向けの Red Hat OpenStack Platform オーバークラウドのデプロイ](#)」を参照してください。
- 最新バージョンの Red Hat OpenStack Platform (`{OpenStackShort}`) 16.1 を使用しています。

#### 手順

1. **stf-connectors.yaml** ファイルを開き、**CollectdEnableLibpodstats** パラメーターを追加して **enable-stf.yaml** の設定を上書きします。**stf-primaries.yaml** が、**enable-stf.yaml** の後に **openstack overcloud deploy** コマンドから呼び出されていることを確認します。

```
CollectdEnableLibpodstats: false
```

2. オーバークラウドのデプロイメント手順を続けます。詳細は、「[オーバークラウドのデプロイ](#)」を参照してください。

## 5.9. RED HAT OPENSTACK PLATFORM API のステータスおよびコンテナ化されたサービスの健全性

OCI (Open Container Initiative) 標準を使用して、ヘルスチェックスクリプトを定期的に行って、各 Red Hat OpenStack Platform (`{OpenStackShort}`) サービスのコンテナの正常性ステータスを評価することができます。ほとんどの `{OpenStackShort}` サービスは、問題を記録し、バイナリーのステータスを返すヘルスチェックを実装します。`{OpenStackShort}` API の場合には、ヘルスチェックはルートエンドポイントをクエリーし、応答時間に基づいてその正常性を判別します。

{OpenStackShort} コンテナの正常性および API ステータスのモニタリングは、デフォルトで有効にされます。

## 追加リソース

- {OpenStackShort} コンテナの正常性および API ステータスのモニタリングを無効にするには、「[コンテナの正常性および API ステータスモニタリングの無効化](#)」を参照してください。

### 5.9.1. コンテナの正常性および API ステータスモニタリングの無効化

{OpenStackShort} コンテナ化されたサービスの正常性および API ステータスのモニタリングを無効にするには、**CollectdEnableSensubility** パラメーターを **false** に設定する必要があります。

## 要件

- templates ディレクトリーに **stf-connectors.yaml** ファイルを作成している。詳細は、「[Service Telemetry Framework 向けの Red Hat OpenStack Platform オーバークラウドのデプロイ](#)」を参照してください。
- 最新バージョンの Red Hat OpenStack Platform({OpenStackShort})16.1 を使用しています。

## 手順

1. **stf-connectors.yaml** を開き、**CollectdEnableLibpodstats** パラメーターを追加して **enable-stf.yaml** の設定を上書きします。**stf-primaries.yaml** が、**enable-stf.yaml** の後に **openstack overcloud deploy** コマンドから呼び出されていることを確認します。

```
CollectdEnableSensubility: false
```

2. オーバークラウドのデプロイメント手順を続けます。詳細は、「[オーバークラウドのデプロイ](#)」を参照してください。

## 追加リソース

- 複数のクラウドアドレスの詳細は、「[複数クラウドの設定](#)」を参照してください。

## 第6章 SERVICE TELEMETRY FRAMEWORK のバージョン 1.3 へのアップグレード

Service Telemetry Framework (STF) 1.2 から STF 1.3 に移行するには、Red Hat OpenShift Container Platform 環境の **service-telemetry** 名前空間の **ClusterServiceVersion** および **Subscription** オブジェクトを置き換える必要があります。

### 要件

- Red Hat OpenShift Container Platform 環境を 4.6 にアップグレードしている。STF 1.3 は、Red Hat OpenShift Container Platform 4.5 以前で実行されません。STF 1.2 は、Red Hat OpenShift Container Platform 4.7 以降では実行されません。
- 環境をアップグレードする前にデータのバックアップを作成している。STF 1.2 を 1.3 にアップグレードすると、Smart Gateway のアップグレード中に短時間停止が生じます。さらに、Operator の置き換え中には、**ServiceTelemetry** および **SmartGateway** オブジェクトへの変更は加えられません。

STF 1.2 から 1.3 にアップグレードするには、以下の手順を実行します。

### 手順

1. [STF 1.2 Operator](#) を削除します。
2. [Service Telemetry Operator](#) にサブスクライブします。

## 6.1. SERVICE TELEMETRY FRAMEWORK 1.2 OPERATOR の削除

STF 1.2、Smart Gateway Operator、および Service Telemetry Operator から Operator を削除します。



### 警告

API インターフェースに変更が加えられたため、**clouds** パラメーターを一時的に削除する必要があります。これにより、アップグレードの完了後にすべての Smart Gateway が削除され、アップグレード中にメトリクスおよびイベントを配信できなくなります。

### 手順

1. Operator をアップグレードする前にこのパラメーターを削除する必要があるため、現在の **ServiceTelemetry** オブジェクトを取得して **clouds** パラメーターなどの内容をメモします。

```
$ oc get stf default -oyaml
```

2. ServiceTelemetry オブジェクトを変更して、**clouds** パラメーターを消去し、空の一覧に設定します。**cloudsRemoveOnMissing** を **true** に設定して、すべての Smart Gateways を削除します。



**警告**

このコマンドは、アップグレードが完了して **clouds** オブジェクトが再定義されるまで、すべての監視機能を停止します。デフォルトのクラウド設定を使用する場合には、ServiceTelemetry オブジェクトに定義されません。

```
$ oc patch stf default --patch '$spec:\n clouds: []\n cloudsRemoveOnMissing: true' --type=merge
```

- Smart Gateway Pod が完全に終了して削除されるまで、Smart Gateway Pod をモニターします。

```
$ oc get pods --selector app=smart-gateway --watch
```

NAME	READY	STATUS	RESTARTS	AGE
default-cloud1-ceil-meter-smartgateway-58cc854f4-hgk92	1/1	Running	0	2m42s
default-cloud1-coll-meter-smartgateway-6c76f9786d-crn9b	2/2	Running	0	2m55s
default-cloud1-coll-meter-smartgateway-6c76f9786d-crn9b	2/2	Terminating	0	3m12s
default-cloud1-ceil-meter-smartgateway-58cc854f4-hgk92	1/1	Terminating	0	3m
...				

- Smart Gateway Operator の **Subscription** 名を取得します。

```
$ oc get sub smart-gateway-operator-stable-1.2-redhat-operators-openshift-marketplace
```

NAME	PACKAGE	SOURCE
smart-gateway-operator-stable-1.2-redhat-operators-openshift-marketplace	smart-gateway-operator	redhat-operators
channel	stable-1.2	

- Smart Gateway Operator サブスクリプションを削除します。

```
$ oc delete sub smart-gateway-operator-stable-1.2-redhat-operators-openshift-marketplace
```

```
subscription.operators.coreos.com "smart-gateway-operator-stable-1.2-redhat-operators-openshift-marketplace" deleted
```

- Smart Gateway Operator ClusterServiceVersion を取得します。

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

```
clusterserviceversion.operators.coreos.com/smart-gateway-operator.v2.2.1623675667
```

- Smart Gateway Operator ClusterServiceVersion を削除します。

```
$ oc delete clusterserviceversion.operators.coreos.com/smart-gateway-operator.v2.2.1623675667
```

```
clusterserviceversion.operators.coreos.com "smart-gateway-operator.v2.2.1623675667"
deleted
```

8. SmartGateway カスタムリソース定義を削除します。

```
$ oc delete crd smartgateways.smartgateway.infra.watch
```

```
customresourcedefinition.apiextensions.k8s.io "smartgateways.smartgateway.infra.watch"
deleted
```

9. Service Telemetry Operator Subscription にパッチを適用し、stable-1.3 チャンネルを使用します。

```
$ oc patch sub service-telemetry-operator --patch '$spec:\n channel: stable-1.3' --
type=merge
```

```
subscription.operators.coreos.com/service-telemetry-operator patched
```

10. Smart Gateway Operator がインストールされ、Service Telemetry Operator がバージョン 1.2 および 1.3 の **Pending** になるまで **oc get csv** コマンドの出力を監視します。

```
$ oc get csv
```

NAME	DISPLAY	VERSION
REPLACES	PHASE	
amq7-cert-manager.v1.0.0	Red Hat Integration - AMQ Certificate Manager	1.0.0
Succeeded		
amq7-interconnect-operator.v1.2.4	Red Hat Integration - AMQ Interconnect	1.2.4
amq7-interconnect-operator.v1.2.3	Succeeded	
elastic-cloud-eck.v1.6.0	Elasticsearch (ECK) Operator	1.6.0
elastic-cloud-eck.v1.5.0	Succeeded	
prometheusoperator.0.47.0	Prometheus Operator	0.47.0
prometheusoperator.0.37.0	Succeeded	
service-telemetry-operator.v1.2.1623675667	Service Telemetry Operator	
1.2.1623675667	Pending	
service-telemetry-operator.v1.3.1622734200	Service Telemetry Operator	
1.3.1622734200	service-telemetry-operator.v1.2.1623675667	Pending
smart-gateway-operator.v3.0.1622734308	Smart Gateway Operator	
3.0.1622734308	Succeeded	

11. Service Telemetry Operator v1.2 ClusterServiceVersion を削除します。

```
$ oc delete csv service-telemetry-operator.v1.2.1623675667
```

```
clusterserviceversion.operators.coreos.com "service-telemetry-operator.v1.2.1623675667"
deleted
```

12. ServiceTelemetry オブジェクトを編集し、先ほどメモした **clouds** パラメーターの内容を挿入します。デフォルトの Smart Gateway インスタンスを使用しているため、**clouds** パラメーターが定義されていない場合は、**clouds: []** パラメーターを削除します。

```
$ oc edit stf default
```

13. Smart Gateway が復元されていることを確認します。

```
$ oc get pods --selector app=smart-gateway

NAME                                READY STATUS RESTARTS AGE
default-cloud1-ceil-meter-smartgateway-6484b98b68-sl7mb 2/2 Running 0      5m56s
default-cloud1-coll-meter-smartgateway-799f687658-nfzr6 2/2 Running 0      6m6s
```

## 6.2. SERVICE TELEMETRY OPERATOR のサブスクリプション

STF インスタンスを管理する Service Telemetry Operator にサブスクリプションする必要があります。

### 手順

1. Service Telemetry Operator サブスクリプションを作成します。

```
$ oc create -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: service-telemetry-operator
  namespace: service-telemetry
spec:
  channel: stable-1.3
  installPlanApproval: Automatic
  name: service-telemetry-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

2. Service Telemetry Operator および依存する Operator を検証します。

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

NAME                                DISPLAY          VERSION
REPLACES                            PHASE
amq7-cert-manager.v1.0.0             Red Hat Integration - AMQ Certificate Manager 1.0.0
Succeeded
amq7-interconnect-operator.v1.2.3    Red Hat Integration - AMQ Interconnect      1.2.3
amq7-interconnect-operator.v1.2.2    Succeeded
elastic-cloud-eck.v1.6.0             Elasticsearch (ECK) Operator                1.6.0
elastic-cloud-eck.v1.5.0             Succeeded
prometheusoperator.0.47.0            Prometheus Operator                         0.47.0
prometheusoperator.0.37.0            Succeeded
service-telemetry-operator.v1.3.1622734200 Service Telemetry Operator
1.3.1622734200                       Succeeded
smart-gateway-operator.v3.0.1622734308 Smart Gateway Operator
3.0.1622734308                       Succeeded
```

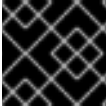
新規 Operator が起動すると、既存の **ServiceTelemetry** および **SmartGateway** オブジェクトを調整し、Smart Gateway コンテナを再起動します。

- Smart Gateway コンテナの状態を確認します。

```
oc get pods
```

```
NAME                                READY STATUS    RESTARTS AGE
...
default-cloud1-ceil-meter-smartgateway-5849c4cdb5-xgl42 1/1 Running    0      35s
default-cloud1-coll-meter-smartgateway-749674f75c-k7pm7 2/2 Terminating 0
56m
default-cloud1-coll-meter-smartgateway-868476456b-ksh9b 2/2 Running    0      26s
...
```

## 第7章 COLLECTD プラグイン



### 重要

Red Hat は本書で現在、今回のリリース向けのプラグイン情報を更新しています。

Red Hat OpenStack Platform 16.1 環境に合わせて、複数の collectd プラグインを設定できます。以下の例のように、特定のプラグインで利用可能なパラメーターの表を参照してください。

ExtraConfig:

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

Prometheus または Grafana クエリーの特定プラグインのメトリクステーブルを参照します。

### collectd::plugin::aggregation

複数の値を **aggregation** プラグインで集約できます。メトリクスを算出するには、**sum**、**average**、**min**、**max** などの集約関数を使用します (例: 平均および合計の CPU 統計)。

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

### collectd::plugin::ampq

#### collectd::plugin::amqp1

**amqp1** プラグインを使用して、AMQ Interconnect などの amqp1 メッセージバスに値を書き込みます。

表7.1 amqp1 パラメーター

パラメーター	タイプ
manage_package	ブール値
transport	文字列
host	string
port	整数
user	文字列
password	文字列
address	文字列
instances	ハッシュ
retry_delay	整数
send_queue_limit	整数

パラメーター	タイプ
interval	整数

### 設定例

```
Parameter_defaults:
CollectdExtraPlugins:
- amqp1
ExtraConfig:
collectd::plugin::amqp1::send_queue_limit: 50
```

### collectd::plugin::apache

**apache** プラグインを使用して Apache データを収集します。

表7.2 Apache パラメーター

パラメーター	タイプ
instances	ハッシュ
interval	整数
manage-package	ブール値
package_install_options	リスト

### 設定例

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

### 追加リソース

**apache** プラグインの設定の詳細は、[apache](#) を参照してください。

### collectd::plugin::battery

**battery** プラグインを使用して、ラップトップのバッテリーの残量、電源、または電圧を報告します。

表7.3 バッテリーパラメーター

パラメーター	タイプ
values_percentage	ブール値
report_degraded	ブール値

パラメーター	タイプ
query_state_fs	ブール値
interval	整数

## 追加リソース

**battery** プラグインの設定の詳細は、「[バッテリー](#)」を参照してください。

### collectd::plugin::bind

**bind** プラグインを使用して、DNS サーバーからクエリーおよび応答に関するエンコードされた統計を取得します。プラグインは、値を collectd に送信します。

### collectd::plugin::ceph

**ceph** プラグインを使用して、ceph デーモンからデータを収集します。

表7.4 Ceph パラメーター

パラメーター	タイプ
daemons	配列
longrunavglatency	ブール値
convertspecialmetrictypes	ブール値
manage_package	ブール値
package_name	文字列

## 設定例

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



### 注記

Object Storage Daemon (OSD) がすべてのノードにない場合には、OSD を一覧表示する必要があります。



## 注記

collectd をデプロイする時に、**ceph** プラグインを ceph ノードに追加します。デプロイメントが失敗するので、Ceph ノードの **ceph** プラグインを **CollectdExtraPlugins** に追加しないでください。

## 追加リソース

**ceph** プラグインの設定の詳細は、「[ceph](#)」を参照してください。

### collectd::plugins::cgroups

**cgroups** プラグインを使用して、cgroup 内のプロセスの情報を収集します。

表7.5 cgroups パラメーター

パラメーター	タイプ
ignore_selected	ブール値
interval	整数
cgroups	リスト

## 追加リソース

**cgroups** プラグインの設定の詳細は、「[cgroups](#)」を参照してください。

### collectd::plugin::connectivity

**connectivity** プラグインを使用して、ネットワークインターフェースの状態を監視します。



## 注記

インターフェースが一覧にない場合は、すべてのインターフェースがデフォルトで監視されます。

表7.6 接続性のパラメーター

パラメーター	タイプ
インターフェース	配列

## 設定例

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

## 追加リソース

**connectivity** プラグインの設定の詳細は、「[接続性](#)」を参照してください。



**collectd::plugin::connttrack**

**connttrack** プラグインを使用して、Linux 接続追跡テーブルのエントリー数を追跡します。このプラグインのパラメーターはありません。

**collectd::plugin::contextswitch**

**ContextSwitch** プラグインを使用して、システムで処理されるコンテキストスイッチの数を収集します。

**追加リソース**

**contextswitch** プラグインの設定の詳細は、「[contextswitch](#)」を参照してください。

**collectd::plugin::cpu**

**cpu** プラグインを使用して、CPU がさまざまな状態に費やした時間 (例: idle、ユーザーコードの実行中、システムコードの実行中、IO 操作の待機中、その他の状態など) を監視します。

**cpu** プラグインは、パーセンテージの値ではなく **\_jiffies\_** を収集します。jiffy の値は、ハードウェアプラットフォームのクロック周波数により異なるため、絶対的な間隔単位ではありません。

パーセンテージの値を報告するには、ブール値パラメーター **reportbycpu** および **reportbystate** を **true** に設定し、ブール値のパラメーター値 **percentage** を true に設定します。

表7.7 CPU メトリック

名前	説明	クエリー
idle	アイドル時間	collectd_cpu_total{...,type_instance=idle}
interrupt	割り込みでブロックされる CPU	collectd_cpu_total{...,type_instance=interrupt}
nice	優先度の低いプロセスを実行する時間	collectd_cpu_total{...,type_instance=nice}
softirq	割り込み要求の処理に費やされたサイクル数	collectd_cpu_total{...,type_instance=waitirq}
steal	ハイパーバイザーが別の仮想プロセッサに対応している間、仮想 CPU が実際の CPU を待機する時間の割合	collectd_cpu_total{...,type_instance=steal}
system	システムレベル (カーネル) で費やした時間	collectd_cpu_total{...,type_instance=system}
user	ユーザープロセスによって使用される jiffies	collectd_cpu_total{...,type_instance=user}
wait	未処理の I/O 要求で待機中の CPU	collectd_cpu_total{...,type_instance=wait}

表7.8 CPU パラメーター

パラメーター	タイプ
reportbystate	ブール値
valuespercentage	ブール値
reportbycpu	ブール値
reportnumcpu	ブール値
reportgueststate	ブール値
subtractgueststate	ブール値
interval	整数

## 設定例

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

## 追加リソース

**cpu** プラグインの設定の詳細は、[cpu](#) を参照してください。

### collectd::plugin::cpufreq

- なし

### collectd::plugin::cpusleep

### collectd::plugin::csv

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

### collectd::plugin::curl\_json

### collectd::plugin::curl

### collectd::plugin::curl\_xml

### collectd::plugin::dbi

### collectd::plugin::df

**df** プラグインを使用して、ファイルシステムのディスク領域の使用状況に関する情報を収集します。

## 表7.9 df メトリクス

名前	説明	クエリー
free	空きディスク容量	collectd_df_df_complex{..., type_instance="free"}
reserved	予約済みディスク容量	collectd_df_df_complex{..., type_instance="reserved"}
used	使用済みディスク容量	collectd_df_df_complex{..., type_instance="used"}

表7.10 df パラメーター

パラメーター	タイプ
devices	配列
fstypes	配列
ignoreselected	ブール値
mountpoints	配列
reportbydevice	ブール値
reportinodes	ブール値
reportreserved	ブール値
valuesabsolute	ブール値
valuespercentage	ブール値

## 設定例

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

## 追加リソース

**df** プラグインの設定の詳細は、[df](#) を参照してください。

### collectd::plugin::disk

**disk** プラグインを使用してハードディスクのパフォーマンス統計と (サポートされている場合には) パーティションの情報を収集します。このプラグインはデフォルトで有効です。

表7.11 ディスクパラメーター

パラメーター	タイプ
disks	配列
ignoreselected	ブール値
udevnameattr	文字列

表7.12 ディスクメトリクス

名前	説明
merged	すでにキューに入っている操作 (例: 2 つ以上の論理操作に対応する物理ディスクアクセス) を一緒にマージできる操作の数。
time	I/O 操作が完了するまでの平均時間。この計算は煩雑であるため、実際の値は鵜呑みにしないようにしてください。
io_time	I/Os (ms) の処理に費やした時間。このメトリックをデバイスの負荷率として利用できます。1 秒の値は、負荷の 100% に一致します。
weighted_io_time	I/O の完了時間と、累積する可能性のあるバックログを測定します。
pending_operations	保留中の I/O 操作のキューサイズを表示します。

## 設定例

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

## 追加リソース

**disk** プラグインの設定の詳細は、[disk](#) を参照してください。

**collectd::plugin::dns**  
**collectd::plugin::dpdk\_telemetry**  
**collectd::plugin::entropy**

- collectd::plugin::entropy::interval

### collectd::plugin::ethstat

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

- `collectd::plugin::ethstat::interval`

### `collectd::plugin::exec`

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

### `collectd::plugin::fhcount`

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

### `collectd::plugin::filecount`

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

### `collectd::plugin::fscache`

- なし

### `collectd-hddtemp`

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

### `collectd::plugin::hugepages`

`hugepages` プラグインを使用して `hugepages` 情報を収集します。このプラグインはデフォルトで有効です。

表7.13 `hugepages` パラメーター

パラメーター	タイプ
デフォルト値	<code>report_per_node_hp</code>
ブール値	<code>true</code>
<code>report_root_hp</code>	ブール値
<code>true</code>	<code>values_pages</code>
ブール値	<code>true</code>

パラメーター	タイプ
values_bytes	ブール値
false	values_percentage
ブール値	false

## 設定例

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

## 追加リソース

- **hugepages** プラグインの設定の詳細は、「[ヒュージページ](#)」を参照してください。

### collectd::plugin::intel\_rdt

### collectd::plugin::interface

**interface** プラグインを使用して、オクテットごとのパケット数、秒ごとのパケットレート、およびエラーレートでインターフェーストラフィックを測定します。このプラグインはデフォルトで有効です。

表7.14 インターフェースパラメーター

パラメーター	タイプ
デフォルト	interfaces
配列	[]
ignoreselected	Boolean
false	reportinactive
Boolean	true

## 設定例

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

## 追加リソース

- **interfaces** プラグインの設定の詳細は、「[インターフェース](#)」を参照してください。

## collectd::plugin::ipc

- なし

## collectd::plugin::ipmi

- collectd::plugin::ipmi::ignore\_selected
- collectd::plugin::ipmi::notify\_sensor\_add
- collectd::plugin::ipmi::notify\_sensor\_remove
- collectd::plugin::ipmi::notify\_sensor\_not\_present
- collectd::plugin::ipmi::sensors
- collectd::plugin::ipmi::interval

## collectd::plugin::iptables

## collectd::plugin::irq

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

## collectd::plugin::load

**load** プラグインを使用してシステム負荷を収集し、システムの使用に関する概要を取得します。このプラグインはデフォルトで有効です。

表7.15 プラグインパラメーター

パラメーター	タイプ
report_relative	Boolean

## 設定例

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

## 追加リソース

- **load** プラグインの設定の詳細は、「[ロード](#)」を参照してください。

## collectd::plugin::logfile

- collectd::plugin::logfile::log\_level
- collectd::plugin::logfile::log\_file
- collectd::plugin::logfile::log\_timestamp

- `collectd::plugin::logfile::print_severity`
- `collectd::plugin::logfile::interval`

`collectd::plugin::log_logstash`  
`collectd::plugin::madwifi`  
`collectd::plugin::match_empty_counter`  
`collectd::plugin::match_hashed`  
`collectd::plugin::match_regex`  
`collectd::plugin::match_timediff`  
`collectd::plugin::match_value`  
`collectd::plugin::mbmon`  
`collectd::plugin::mcelog`

**mcelog** プラグインを使用して、マシンチェック例外 (MCE: Machine Check Exceptions) の発生時に関連する通知および統計を送信します。**mcelog** をデーモンモードで実行し、ロギング機能が有効化されていることを確認します。

表7.16 mcelog パラメーター

パラメーター	タイプ
Mcelogfile	文字列
メモリー	hash { mcelogclientsocket[string], persistentnotification[boolean] }

## 設定例

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

## 追加リソース

- **mcelog** プラグインの設定の詳細は、[celog](#) を参照してください。

`collectd::plugin::md`  
`collectd::plugin::memcachec`  
`collectd::plugin::memcached`

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

## `collectd::plugin::memory`

**memory** プラグインは、システムのメモリーに関する情報を提供します。このプラグインはデフォルトで有効です。

表7.17 メモリーパラメーター



パラメーター	タイプ
valuesabsolute	ブール値
valuespercentage	ブール値

## 設定例

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

## 追加リソース

- **memory** プラグインの設定の詳細は、「[メモリー](#)」を参照してください。

### collectd::plugin::multimeter

#### collectd::plugin::mysql

- collectd::plugin::mysql::interval

### collectd::plugin::netlink

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

### collectd::plugin::network

- collectd::plugin::network::timetolive
- collectd::plugin::network::maxpacketize
- collectd::plugin::network::forward
- collectd::plugin::network::reportstats
- collectd::plugin::network::listeners
- collectd::plugin::network::servers
- collectd::plugin::network::interval

**collectd::plugin::nfs**

- collectd::plugin::nfs::interval

**collectd::plugin::notify\_nagios**  
**collectd::plugin::ntpd**

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

**collectd::plugin::numa**

- なし

**collectd::plugin::olsrd**  
**collectd::plugin::openldap**  
**collectd::plugin::openvpn**

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

**collectd::plugin::ovs\_stats**

OVS に接続されたインターフェースの統計値を収集するには、**ovs\_stats** プラグインを使用します。**ovs\_stats** プラグインは、OVSDB 管理プロトコル (RFC7047) モニターメカニズムを使用して OVSDB から統計値を取得します。

表7.18 ovs\_stats パラメーター

パラメーター	タイプ
address	文字列
bridges	リスト
port	整数
socket	文字列

## 設定例

以下の例は、**ovs\_stats** プラグインを有効にする方法を示しています。オーバークラウドを OVS でデプロイする場合には、**ovs\_stats** プラグインを有効にする必要はありません。

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

## 追加リソース

- **ovs\_stats** プラグインの設定の詳細は、[ovs\\_stats](#) を参照してください。

## collectd::plugin::pcie\_errors

**pcie\_errors** プラグインを使用して、ベースラインおよび Advanced Error Reporting (AER) エラーの PCI 設定領域をポーリングし、AER イベントの syslog を解析します。エラーは通知により報告されません。

表7.19 pcie\_errors パラメーター

パラメーター	タイプ
source	列挙 (sysfs, proc)
access	文字列
reportmasked	ブール値
persistent_notifications	ブール値

## 設定例

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

## 追加リソース

- **pcie\_errors** プラグインの設定に関する詳細は、[pcie\\_errors](#) を参照してください。

## collectd::plugin::ping

- collectd::plugin::ping::hosts
- collectd::plugin::ping::timeout
- collectd::plugin::ping::ttl
- collectd::plugin::ping::source\_address
- collectd::plugin::ping::device

- collectd::plugin::ping::max\_missed
- collectd::plugin::ping::size
- collectd::plugin::ping::interval

### collectd::plugin::powerdns

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

### collectd::plugin::processes

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

### collectd::plugin::processes

**processes** プラグインは、システムのプロセスに関する情報を提供します。このプラグインはデフォルトで有効です。

表7.20 プラグインパラメーター

パラメーター	タイプ
processes	配列
process_matches	配列
collect_context_switch	Boolean
collect_file_descriptor	Boolean
collect_memory_maps	Boolean

### 追加リソース

- **processes** プラグインの設定の詳細は「[プロセス](#)」を参照してください。

## collectd::plugin::protocols

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

## collectd::plugin::python

## collectd::plugin::sensors

## collectd::plugin::serial

## collectd::plugin::smart

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

## collectd::plugin::snmp

## collectd::plugin::snmp\_agent

**snmp\_agent** プラグインを SNMP サブエージェントとして使用し、collectd メトリクスを関連する OID にマッピングします。snmp エージェントには、実行中の snmpd サービスも必要です。

### 設定例

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

### その他のリソース:

**snmp\_agent** の設定方法の詳細は、[snmp\\_agent](#) を参照してください。

## collectd::plugin::statsd

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

- `collectd::plugin::statsd::timersum`
- `collectd::plugin::statsd::timercount`
- `collectd::plugin::statsd::interval`

### **collectd::plugin::swap**

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

### **collectd::plugin::sysevent**

### **collectd::plugin::syslog**

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

### **collectd::plugin::table**

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

### **collectd::plugin::tail**

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

### **collectd::plugin::tail\_csv**

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

### **collectd::plugin::target\_notification**

### **collectd::plugin::target\_replace**

### **collectd::plugin::target\_scale**

### **collectd::plugin::target\_set**

### **collectd::plugin::target\_v5upgrade**

### **collectd::plugin::tcpconns**

- `collectd::plugin::tcpconns::localports`
- `collectd::plugin::tcpconns::remoteports`

- collectd::plugin::tcpconns::listening
- collectd::plugin::tcpconns::allportssummary
- collectd::plugin::tcpconns::interval

### **collectd::plugin::ted**

### **collectd::plugin::thermal**

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

### **collectd::plugin::threshold**

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

### **collectd::plugin::turbostat**

- collectd::plugin::turbostat::core\_c\_states
- collectd::plugin::turbostat::package\_c\_states
- collectd::plugin::turbostat::system\_management\_interrupt
- collectd::plugin::turbostat::digital\_temperature\_sensor
- collectd::plugin::turbostat::tcc\_activation\_temp
- collectd::plugin::turbostat::running\_average\_power\_limit
- collectd::plugin::turbostat::logical\_core\_names

### **collectd::plugin::unixsock**

### **collectd::plugin::uptime**

- collectd::plugin::uptime::interval

### **collectd::plugin::users**

- collectd::plugin::users::interval

### **collectd::plugin::uuid**

- collectd::plugin::uuid::uuid\_file
- collectd::plugin::uuid::interval

### **collectd::plugin::virt**

**virt** プラグインを使用して、ホスト上の仮想マシンの **libvirt** API で CPU、ディスク、ネットワーク負荷、およびその他のメトリックを収集します。

表7.21 virt パラメーター

パラメーター	タイプ
connection	文字列
refresh_interval	ハッシュ
domain	文字列
block_device	文字列
interface_device	文字列
ignore_selected	ブール値
plugin_instance_format	文字列
hostname_format	文字列
interface_format	文字列
extra_stats	文字列

## 設定例

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

## 追加リソース

**virt** プラグインの設定の詳細は、「[virt](#)」を参照してください。

### collectd::plugin::vmem

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

### collectd::plugin::vserver

### collectd::plugin::wireless

### collectd::plugin::write\_graphite

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



**collectd::plugin::write\_http**

**write\_http** 出力プラグインを使用して、POST リクエストを使用し JSON でメトリックをエンコードして、または **PUTVAL** コマンドを使用して、HTTP サーバーに値を送信します。

表7.22 write\_http パラメーター

パラメーター	タイプ
ensure	列挙 [present,absent]
nodes	ハッシュ[String, Hash[String, Scalar]]
urls	ハッシュ[String, Hash[String, Scalar]]
manage_package	ブール値

## 設定例

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

## 追加リソース

- **write\_http** プラグインの設定に関する詳細は、[write\\_http](#) を参照してください。

**collectd::plugin::write\_kafka**

**write\_kafka** プラグインを使用して、値を Kafka トピックに送信します。**write\_kafka** プラグインを1つ以上のトピックブロックで設定します。トピックブロックごとに、一意の名前と1つの Kafka プロデューサーを指定する必要があります。topic ブロックでは、以下の per-topic パラメーターを使用できません。

表7.23 write\_kafka パラメーター

パラメーター	タイプ
kafka_hosts	配列[String]
kafka_port	整数
topics	ハッシュ
properties	ハッシュ

パラメーター	タイプ
meta	ハッシュ

## 設定例

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

### その他のリソース:

**write\_kafka** プラグインの設定方法は「[write\\_kafka](#)」を参照してください。

### collectd::plugin::write\_log

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

### collectd::plugin::zfs\_arc

- なし