Red Hat OpenStack Platform 16.0

Service Telemetry Framework 1.0

Service Telemetry Framework 1.0 のインストールおよびデプロイ

Last Updated: 2022-09-13

Red Hat OpenStack Platform 16.0 Service Telemetry Framework 1.0

Service Telemetry Framework 1.0 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

概要

本ガイドでは、コアコンポーネントのインストールと Service Telemetry Framework 1.0 のデプロイについて説明します。

目次

第1章 SERVICE TELEMETRY フレームワークの概要

Service Telemetry Framework(STF)は、Red Hat OpenStack Platform またはサードパーティーノードから測定およびデータを自動的に収集し、その情報を一元的に送信し、ストレージ、取得、およびモニタリング用の Red Hat OpenShift Container Platform(OCP)デプロイメントを受信します。データは以下の 2 つのタイプのいずれかになります。

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

イベント

システムで不規則な状態や目立った事態の発生

クライアントに必要なコレクションコンポーネントは軽量です。すべてのクライアントによって共有され、デプロイメントによって共有されるマルチキャストメッセージバスは、高速で信頼性の高いデータトランスポートを提供します。データの送受信に関する他のモジュールコンポーネントは、OCP のコンテナーにデプロイされます。

STF は、アラートの生成、ダッシュボードによる可視化、オーケストレーションをサポートする信頼できる Telemetry 分析の単一ソースなどのモニタリング機能へのアクセスを提供します。

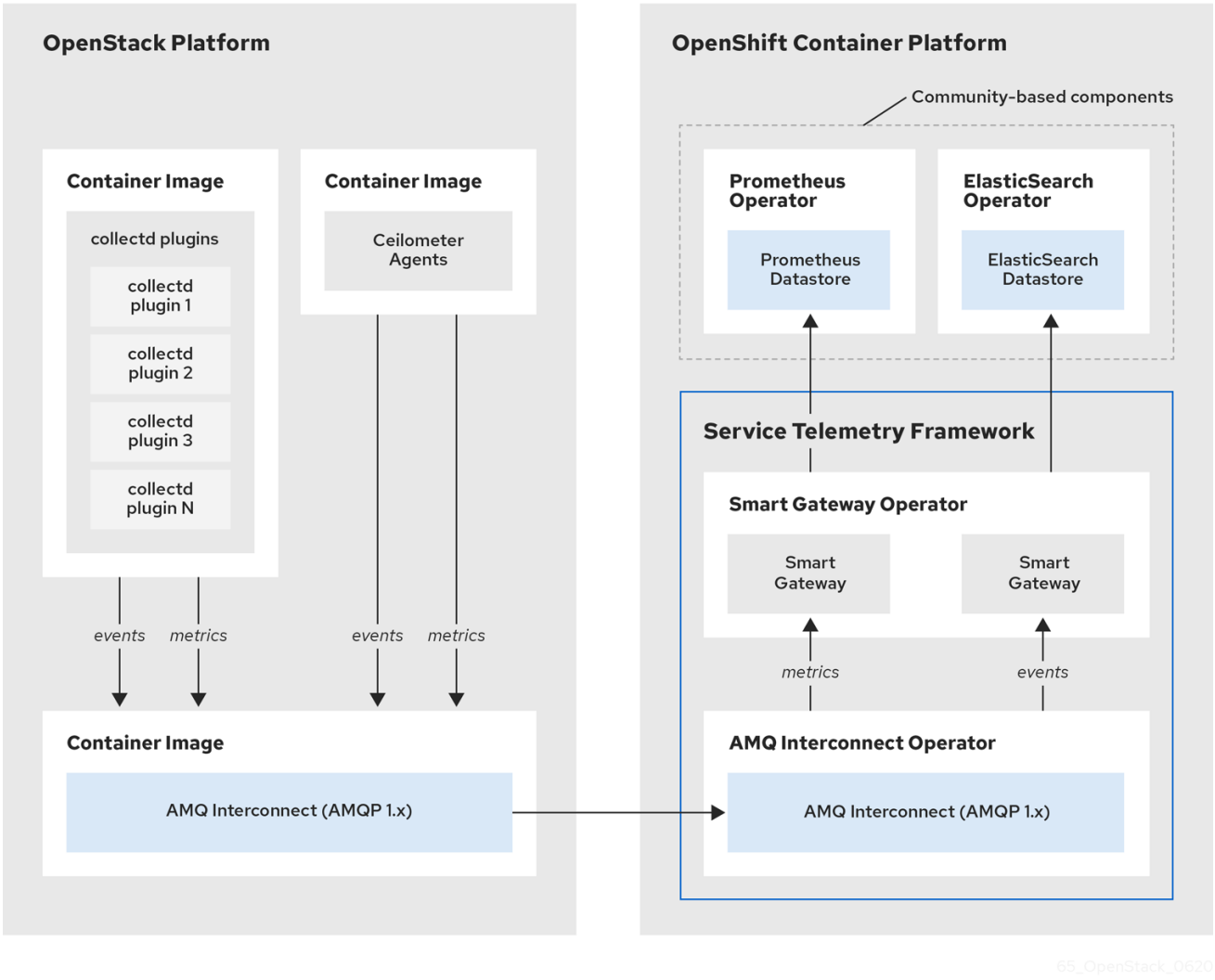
1.1. SERVICE TELEMETRY FRAMEWORK アーキテクチャー

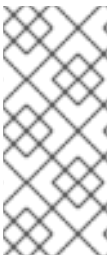
Service Telemetry Framework(STF)は、表1.1「STF コンポーネント」 に記載のコンポーネントを使用します。

表1.1 STF コンポーネント

クライアント	コンポーネント	サーバー (OCP)
はい	Prometheus のストレージ用にメトリクスを STF に移動する AMQP 1.x と互換性のあるメッセージングバス。	はい
いいえ	Smart Gateway: AMQP 1.x バスからメトリクスおよびイベントを選択し、イベントを ElasticSearch に配信したり、メトリクスを Prometheus に提供したりできます。	はい
いいえ	時系列データストレージとしての Prometheus	はい
いいえ	イベントデータストレージとしての Elasticsearch	はい
はい	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する collectd	いいえ
はい	Red Hat OpenStack Platform のメトリクスおよびイベントを収集する ceilometer	いいえ

図1.1 Service Telemetry Framework アーキテクチャー概要





注記

Service Telemetry Framework データ収集コンポーネント (collectd および Ceilometer)、およびトランスポートコンポーネント AMQ Interconnect および Smart Gateway が完全にサポートされます。Operator アーティファクトおよび可視化コンポーネント Grafana を含むデータストレージコンポーネント、Prometheus および ElasticSearch はコミュニティーでサポートされており、正式にはサポートされていません。

メトリクスについては、クライアント側で collectd は高解像度のメトリクスを収集します。collectd は AMQP1 プラグインを使用してデータを Prometheus に配信し、データをメッセージバスに配置します。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のデータストリームを受け取り、それを Prometheus のローカルスクレイプエンドポイントとして公開します。

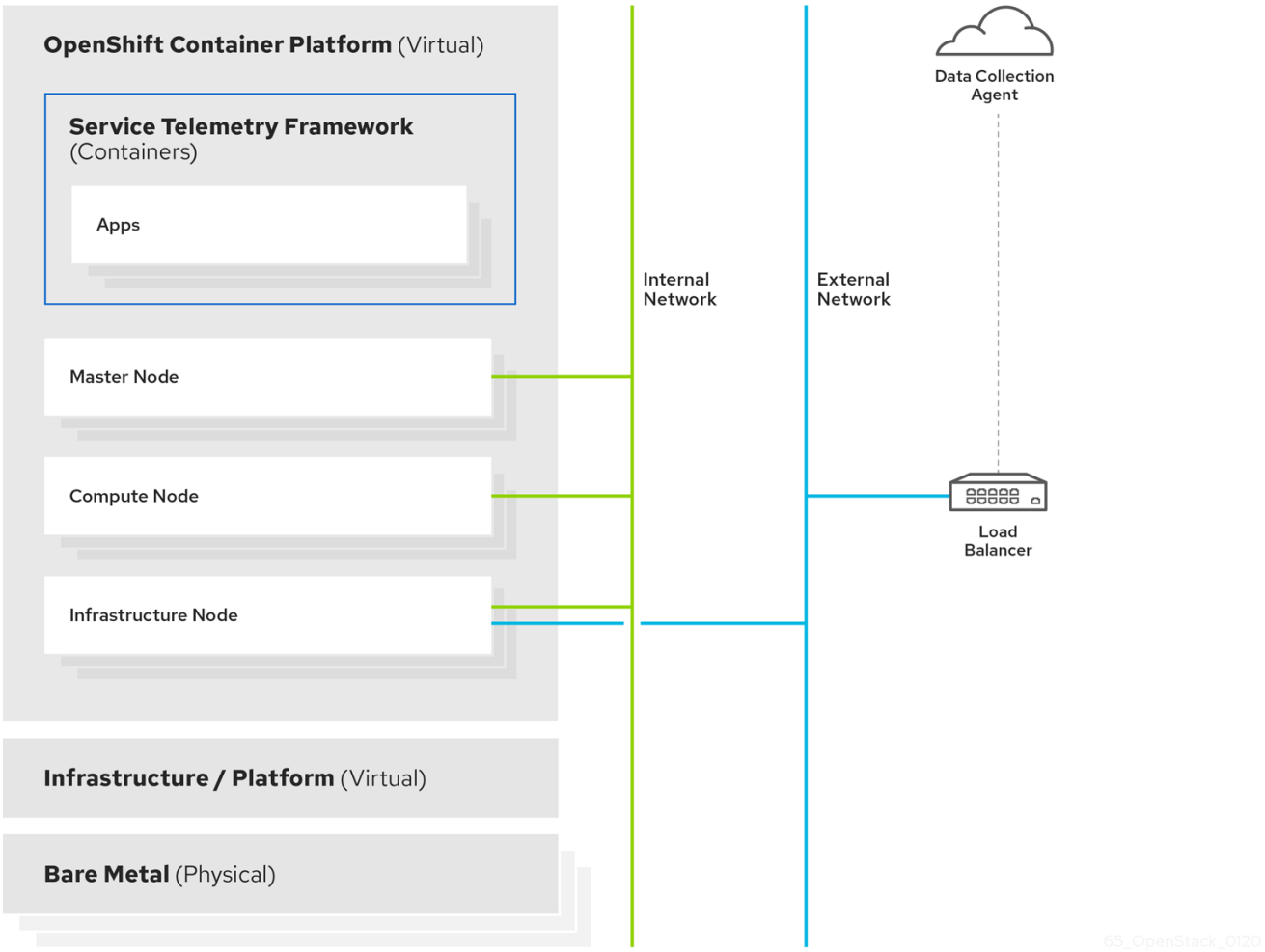
イベントを収集し、保存する予定の場合、collectd または Ceilometer は AMQP1 プラグインを使用してイベントデータをサーバー側に配信し、データをメッセージバスに配置します。別の Smart Gateway が ElasticSearch データストアにデータを書き込みます。

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ています。

- Service Telemetry Framework 1.0(STF)
- Red Hat OpenShift Container Platform(OCP)

- インフラストラクチャープラットフォーム

図1.2 サーバーサイドの STF 監視インフラストラクチャー



Red Hat OpenShift Container Platform のデプロイ方法の詳細は、OCP の製品ドキュメント を参照してください。OCP 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 のパフォーマンスとスケーリングの詳細については、https://access.redhat.com/articles/4907241 を参照してください。

注記

監視するインフラストラクチャーと同じインフラストラクチャーに OCP をインストールしないでください。

1.2. 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。

- 監視するノードの数。
- 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。お使いの Red Hat OpenStack Platform 環境とは別のプラットフォームに、Red Hat OpenStack Platform のモニタリングをインストールするようにしてください。Red Hat OpenShift Container Platform (OCP) 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。OCP のインストールに関する詳細は、OpenShift Container Platform 4.3 ドキュメント を参照してください。

OCP 環境のサイズは、選択するインフラストラクチャーによって異なります。OCP をベアメタルにインストールする際の最小リソース要件の詳細は、『**ベアメタルへのクラスターのインストール**』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 のコアコンポーネントのインストール

Service Telemetry Framework(STF)をインストールする前に、Red Hat OpenShift Container Platform(OCP)バージョン 4.x が実行されており、フレームワークの中核となるコンポーネントを理解するようにしてください。OCP インストール計画プロセスの一環として、管理者が永続ストレージと、OCP 環境の上に STF コンポーネントを実行するための十分なリソースを提供するようにしてください。

警告

現在、STF のインストールには Red Hat OpenShift Container Platform バージョン 4.3 以降が必要です。

2.1. STF のコアコンポーネント

以下の STF コアコンポーネントは Operator によって管理されます。

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

各コンポーネントには、さまざまなアプリケーションコンポーネントおよびオブジェクトのロードに使用できる対応する Operator があります。

関連情報

Operator についての詳細は、『Operator について』ガイドを参照して ください。

2.2. STF 用 OCP 環境の準備

STF 向けの OCP 環境を作成する場合には、永続ストレージ、十分なリソース、およびイベントストレージを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。
- Operators とアプリケーションコンテナーを動かすのに十分なリソースが確保されていることを確認してください。詳細は、「「リソースの割り当て」」を参照してください。
- ElasticSearch をインストールするには、Community Catalog Source を使用する必要があります。コミュニティーカタログを使用しない場合や、イベントを保存しない場合は、「OCP 環境への STF のデプロイ」 を参照してください。
- STF は ElasticSearch を使用してイベントを保存します。イベントには通常の

vm.max_map_count よりも大きく必要になります。Red Hat OpenShift Container Platform では、**vm.max_map_count** の値がデフォルトで設定されます。**vm.max_map_count** の値を編集する方法は、「Node Tuning Operator」 を参照してください。

2.2.1. 永続ボリューム

STF は OCP の永続ストレージを使用して動的にボリュームをインスタンス化し、Prometheus および ElasticSearch がメトリクスおよびイベントを保存できるようにします。

関連情報

OCP の永続ストレージの設定に関する詳細は、「永続ストレージ について」を参照してください。

2.2.1.1. 一時ストレージの使用

警告

STF では一時ストレージを使用できます。ただし、一時ストレージを使用する場合は、Pod が再起動、更新、または別のノードに再スケジュールされると、データの損失が生じる可能性があります。一時ストレージは、本番環境ではなく、開発やテストにのみ使用してください。

手順

- STF の一時ストレージを有効にするには、**ServiceTelemetry** マニフェストで **storageEphemeralEnabled: true** を設定します。

関連情報

STF の一時ストレージを有効にする方法は、「一時ストレージの設定」 を参照してください。

2.2.2. リソースの割り当て

OCP 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。

STF の実行に必要なリソースの量は、お使いの環境と、監視するノードおよびクラウドの数によって異なります。

関連情報

メトリクス収集のサイジングに関する推奨事項は、https://access.redhat.com/articles/4907241 を参照してください。

ElasticSearch のサイジング要件に関する詳細は、https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-managing-compute-resources.html を参照してください。

2.2.3. Node Tuning Operator

STF は ElasticSearch を使用してイベントを保存します。イベントには通常の **vm.max_map_count** よりも大きく必要になります。Red Hat OpenShift Container Platform では、**vm.max_map_count** の値がデフォルトで設定されます。

vm.max_map_count の値を編集する必要がある場合は、Red Hat OpenShift Container Platform がノードを直接管理するため、**sysctl** コマンドを使用してノードのチューニングを手動で適用することはできません。値を設定してインフラストラクチャーに適用するには、Node Tuning Operator を使用する必要があります。詳細は、「ノードチューニング Operator の使用」を 参照してください。

OCP デプロイメントでは、デフォルトの Node Tuning Operator 仕様で、ElasticSearch ワークロードまたはノードでスケジュールされる Pod に必要なプロファイルが提供されます。デフォルトのクラスターノードチューニング仕様を表示するには、以下のコマンドを実行します。

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

デフォルトの仕様の出力は、「クラスターに設定されたデフォルトプロファイル」に記載されています。プロファイルの割り当ては、特定の条件が満たされたときにプロファイルがノードに適用される **recommend** セクションで管理されます。ElasticSearch を STF のノードにスケジュールする時に、以下のプロファイルのいずれかが適用されます。

- **openshift-control-plane-es**
- **openshift-node-es**

ElasticSearch Pod をスケジュールする場合、**tuned.openshift.io/elasticsearch** に一致するラベルがなければなりません。一致するラベルがある場合には、2 つのプロファイルのいずれかが Pod に割り当てられます。ElasticSearch 向けの推奨の Operator を使用する場合に、管理者は特に何もしなくても結構です。カスタムでデプロイされた ElasticSearch を STF で使用する場合、**tuned.openshift.io/elasticsearch** ラベルをすべてのスケジュールされた Pod に追加するようにしてください。

関連情報

ElasticSearch による仮想メモリー使用量の詳細は https://www.elastic.co/guide/en/elasticsearch/reference/current/vm-max-map-count.htmlを参照してください。

プロファイルがノードに適用される方法についての詳細は、「カスタムチューニング仕様」を 参照してください。

2.3. OCP 環境への STF のデプロイ

以下の 2 つの方法で STF を OCP 環境にデプロイできます。

- STF をデプロイし、ElasticSearch でイベントを保存する。詳細は、「「ElasticSearch を使用した OCP 環境への STF のデプロイ」」を参照してください。
- ElasticSearch を使用せずに STF をデプロイし、イベントサポートを無効にする。詳細は、「「ElasticSearch を使用しない OCP 環境への STF のデプロイ」」を参照してください。

2.3.1. ElasticSearch を使用した OCP 環境への STF のデプロイ

以下のタスクを完了します。

1. 「namespace の作成」.

2. 「OperatorGroup の作成」.
3. 「OperatorHub.io Community Catalog Source の有効化」.
4. 「Red Hat STF Operator ソースの有効化」.
5. 「AMQ Certificate Manager Operator のサブスクライブ」.
6. 「Kubernetes Operator での Elastic Cloud のサブスクライブ」.
7. 「Service Telemetry Operator へのサブスクライブ」.
8. 「OCP での ServiceTelemetry オブジェクトの作成」.

2.3.2. ElasticSearch を使用しない OCP 環境への STF のデプロイ

以下のタスクを完了します。

1. 「namespace の作成」.
2. 「OperatorGroup の作成」.
3. 「Red Hat STF Operator ソースの有効化」.
4. 「AMQ Certificate Manager Operator のサブスクライブ」.
5. 「Service Telemetry Operator へのサブスクライブ」.
6. 「OCP での ServiceTelemetry オブジェクトの作成」.

2.3.3. namespace の作成

STF コンポーネントを保持する namespace を作成します。本書では、**service-telemetry** namespace が使用されます。

手順

- 以下のコマンドを入力します。

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

2.3.4. OperatorGroup の作成

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

手順

- 以下のコマンドを入力します。

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

```
spec:
  targetNamespaces:
  - service-telemetry
EOF
```

関連情報

詳細は、「 OperatorGroups 」を参照してください。

2.3.5. OperatorHub.io Community Catalog Source の有効化

ElasticSearch をインストールする前に、OperatorHub.io Community Catalog Source のリソースにアクセスできる必要があります。

手順

- 以下のコマンドを入力します。

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

2.3.6. Red Hat STF Operator ソースの有効化

Red Hat OpenShift Container Platform に STF をデプロイする前に、Operator ソースを有効にする必要があります。

手順

1. Service Telemetry Operator および Smart Gateway Operator が含まれる OperatorSource をインストールします。

```
oc apply -f - <<EOF
apiVersion: operators.coreos.com/v1
kind: OperatorSource
metadata:
  labels:
    opsrc-provider: redhat-operators-stf
  name: redhat-operators-stf
  namespace: openshift-marketplace
spec:
  authorizationToken: {}
  displayName: Red Hat STF Operators
  endpoint: https://quay.io/cnr
  publisher: Red Hat
```

```
  registryNamespace: redhat-operators-stf
  type: appregistry
EOF
```

2. OperatorSource の作成を検証するには、**oc get operatorsources** コマンドを使用します。インポートに成功すると、**MESSAGE** フィールド **が The オブジェクトの結果を返すようになります**。

```
$ oc get -nopenshift-marketplace operatorsource redhat-operators-stf

NAME                  TYPE          ENDPOINT              REGISTRY              DISPLAYNAME
PUBLISHER   STATUS      MESSAGE
redhat-operators-stf   appregistry   https://quay.io/cnr   redhat-operators-stf   Red Hat STF
Operators   Red Hat     Succeeded   The object has been successfully reconciled
```

3. Operator がカタログから利用できることを確認するには、**oc get packagemanifest** コマンドを使用します。

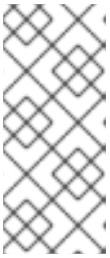
```
$ oc get packagemanifests | grep "Red Hat STF"

smartgateway-operator                Red Hat STF Operators     2m50s
servicetelemetry-operator            Red Hat STF Operators     2m50s
```

2.3.7. AMQ Certificate Manager Operator のサブスクライブ

AMQ Certificate Manager Operator はグローバルスコープで実行され、他の namespace スコープの Operator と使用する場合に Operator Lifecycle Manager の依存関係管理との互換性がないため、他の STF コンポーネントをデプロイする前に AMQ Certificate Manager Operator にサブスクライブする必要があります。

手順

1. AMQ Certificate Manager Operator をサブスクライブし、サブスクリプションを作成し、AMQ7 Certificate Manager を確認します。



注記

AMQ Certificate Manager はすべての namespace に対してグローバルにインストールされているため、指定した **namespace** の値は **openshift-operators** になります。**amq7-cert-manager.v1.0.0** ClusterServiceVersion は、namespace に対して処理が実行されるまで数分間、**service-telemetry** namespace に表示されないことがあります。

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

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

2. **ClusterServiceVersion** を検証するには、**oc get csv** コマンドを使用します。amq7-cert-manager.v1.0.0 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.v1.0.0   Red Hat Integration - AMQ Certificate Manager   1.0.0
Succeeded
```

2.3.8. Kubernetes Operator での Elastic Cloud のサブスクライブ

Service Telemetry Operator をインストールし、イベントを ElasticSearch に保存する予定の場合には、Elastic Cloud Kubernetes Operator を有効にする必要があります。

手順

1. 以下のマニフェストを OCP 環境に適用し、Kubernetes Operator で Elastic Cloud を有効にします。

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

2. Kubernetes の ElasticSearch Cloud の **ClusterServiceVersion** が **成功** したことを確認するには、**oc get csv** コマンドを入力します。

```
$ oc get csv

NAME                    DISPLAY                                VERSION   REPLACES
PHASE
elastic-cloud-eck.v1.1.0   Elastic Cloud on Kubernetes                1.1.0     elastic-cloud-
eck.v1.0.1   Succeeded
```

2.3.9. Service Telemetry Operator へのサブスクライブ

STF インスタンスをインスタンス化するには、**ServiceTelemetry** オブジェクトを作成し、Service Telemetry Operator が環境を作成できるようにします。

手順

1. Service Telemetry Operator サブスクリプションを作成するには、**oc apply -f** コマンドを入力します。

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

2. Service Telemetry Operator および依存する Operator を検証するには、以下のコマンドを入力します。

```
$ 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.v1.2.0   Red Hat Integration - AMQ Interconnect       1.2.0
Succeeded
elastic-cloud-eck.v1.1.0        Elastic Cloud on Kubernetes            1.1.0     elastic-
cloud-eck.v1.0.1        Succeeded
prometheusoperator.0.37.0       Prometheus Operator                    0.37.0
prometheusoperator.0.32.0       Succeeded
service-telemetry-operator.v1.0.2   Service Telemetry Operator             1.0.2     service-
telemetry-operator.v1.0.1   Succeeded
smart-gateway-operator.v1.0.1       Smart Gateway Operator                 1.0.1     smart-
gateway-operator.v1.0.0       Succeeded
```

2.3.10. OCP での ServiceTelemetry オブジェクトの作成

Service Telemetry Framework をデプロイするには、OCP で **ServiceTelemetry** のインスタンスを作成する必要があります。デフォルトでは、**eventEnabled** は false に設定されます。イベントを ElasticSearch に保存しない場合は、event **Enabled** が false に設定されていることを確認します。詳細は、「「ElasticSearch を使用しない OCP 環境への STF のデプロイ」」を参照してください。

ServiceTelemetry マニフェストには、以下のコアパラメーターを利用できます。

表2.1 ServiceTelemetry マニフェストのコアパラメーター

パラメーター	説明	デフォルト値

パラメーター	説明	デフォルト値
eventsEnabled	STF でのイベントサポートを有効にします。ElasticSearch を起動できるようにするには、前提条件の手順が必要です。詳細は、「「Kubernetes Operator での Elastic Cloud のサブスクライブ」」を参照してください。	**false**
metricsEnabled	STF でのメトリクスサポートの有効化。	**true**
highAvailabilityEnabled	STF で高可用性を有効にします。詳細は、「「高可用性」」を参照してください。	**false**
storageEphemeralEnabled	STF での一時ストレージのサポートを有効にします。詳細は、「「一時ストレージ」」を参照してください。	**false**

手順

1. ElasticSearch にイベントを保存するには、デプロイメント時に **eventsEnabled** を true に設定します。

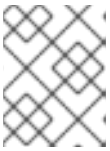
```
oc apply -f - <<EOF
apiVersion: infra.watch/v1alpha1
kind: ServiceTelemetry
metadata:
  name: stf-default
  namespace: service-telemetry
spec:
  eventsEnabled: true
  metricsEnabled: true
EOF
```

2. Service Telemetry Operator で STF デプロイメントログを表示するには、**oc logs** コマンドを使用します。

```
oc logs $(oc get pod --selector='name=service-telemetry-operator' -oname) -c ansible
```

```
PLAY RECAP ***
localhost                  : ok=37   changed=0    unreachable=0    failed=0    skipped=1
rescued=0    ignored=0
```

3. **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します。**



注記

eventsEnabled: true を設定すると、通知 Smart Gateways は ElasticSearch の開始前に一定期間 Error および CrashLoopBackOff になります。

```
$ oc get pods

NAME                                                              READY   STATUS             RESTARTS   AGE
alertmanager-stf-default-0                                        2/2     Running            0          26m
elastic-operator-645dc8b8ff-jwnzt                                 1/1     Running            0          88m
elasticsearch-es-default-0                                        1/1     Running            0          26m
interconnect-operator-6fd49d9fb9-4bl92                            1/1     Running            0
46m
prometheus-operator-bf7d97fb9-kwnlx                               1/1     Running            0
46m
prometheus-stf-default-0                                          3/3     Running            0          26m
service-telemetry-operator-54f4c99d9b-k7ll6                       2/2     Running            0
46m
smart-gateway-operator-7ff58bcf94-66rvx                           2/2     Running            0
46m
stf-default-ceilometer-notification-smartgateway-6675df547q4lbj   1/1     Running
0          26m
stf-default-collectd-notification-smartgateway-698c87fbb7-xj528   1/1     Running
0          26m
stf-default-collectd-telemetry-smartgateway-79c967c8f7-9hsqn      1/1     Running
0          26m
stf-default-interconnect-7458fd4d69-nqbfs                         1/1     Running            0
26m
```

2.4. OCP 環境からの STF の削除

STF 機能が必要なくなったら、OCP 環境から STF を削除します。

以下のタスクを完了します。

1. **「namespace の削除」**
2. **「OperatorSource の削除」.**

2.4.1. namespace の削除

OCP から STF で動作するリソースを削除するには、対象の 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. **ネームスペースからリソースが削除されたことを確認します。**

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

2.4.2. OperatorSource の削除

Service Telemetry Framework を再度インストールすることが予想される場合は、OperatorSource を削除します。OperatorSource を削除する場合、STF に関連する PackageManifest は Operator Lifecycle Manager カタログから削除されます。

手順

1. OperatorSource を削除します。

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

2. STF PackageManifest がプラットフォームから削除されていることを確認します。成功すると、以下のコマンドは結果を返しません。

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

3. インストール時に OperatorHub.io Community Catalog Source を有効にしていて、このカタログソースが不要になった場合は、削除してください。

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

関連情報

OperatorHub.io Community Catalog Source の詳細は、「OCP 環境への STF のデプロイ」を参照してください。

第3章 SERVICE TELEMETRY FRAMEWORK 設定の完了

3.1. RED HAT OPENSTACK PLATFORM の SERVICE TELEMETRY FRAMEWORK への接続

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 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。

分散コンピュートノード(DCN)やスパイン/リーフ等のルーティング対応 L3 ドメインを使用する Red Hat OpenStack Platform クラウドノード上の STF にデータ収集と転送をデプロイするには、「標準外のネットワークトポロジーへのデプロイメント」 を参照してください。

3.2. 標準外のネットワークトポロジーへのデプロイメント

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。このシナリオはスピンリーフや DCN トポロジーで典型的です。DCN 設定についての詳しい情報は、『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 16.0 で STF を使用し、Ceph、Block、または Object Storage 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 ネットワーク構成に加える設定変更と同様の設定を変更する必要があります。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のデプロイメントには、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ます。Red Hat OpenStack Platform 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、Red Hat OpenStack Platform 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

1. Leaf ロールごとにどのネットワークが利用できるかを記録します。ネットワーク名の定義例は、『スパイン/リーフ型ネットワーク』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 グループ（ロール）の作成およびそれらのグループへのネットワークの割り当てに関する詳細は、『スパイン/リーフ型ネットワーク』の「ロールデータファイルの作成」を参照して ください。

2. 各 Leaf 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、ネットワーク識別の 0 は、Leaf 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ています。
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 のリスナーアドレスパラメーターについても同様の設定を行います。

```
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 はリーフの一意のインデントを表します。

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

この設定例は、Ceph Storage の Leaf ロール上のものです。

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

3.3. SERVICE TELEMETRY FRAMEWORK 向けの RED HAT OPENSTACK PLATFORM オーバークラウドの設定

Red Hat OpenStack Platform オーバークラウドを設定するには、データ収集アプリケーションおよび STF へのデータトランスポートを設定して、オーバークラウドをデプロイする必要があります。

Red Hat OpenStack Platform の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. 「AMQ Interconnect ルートアドレスの取得」
2. 「オーバークラウドの STF 接続の設定」
3. 「クライアント側のインストールの検証」

3.3.1. AMQ Interconnect ルートアドレスの取得

STF 向けに Red Hat OpenStack Platform オーバー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 (OCP) 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 -ogo-template='{{ range .items }}{{printf "%s\n" .spec.host }}{{ end }}' |
grep "\-5671"
stf-default-interconnect-5671-service-telemetry.apps.infra.watch
```

注記

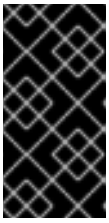
STF インストールがドキュメントと異なる場合は、正しい AMQ Interconnect ルートアドレスが取得されていることを確認します。

3.3.2. オーバークラウドの STF 接続の設定

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。イベントの収集と STF への保存を有効にし、オーバークラウドを展開します。

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。



重要

Service Telemetry Operator は、単一クラウドデプロイメント用の全データ取得およびデータストレージコンポーネントのデプロイメントを単純化します。データストレージドメインを複数のクラウドと共有するには、「複数のクラウドの設定」 を参照してください。

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。
   - **CeilometerQdrPublishEvents: true** パラメーターを追加して、Ceilometer イベントの STF への収集および転送を有効にします。
   - **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:
  EventPipelinePublishers: []
  CeilometerQdrPublishEvents: true
  MetricsQdrConnectors:
  - host: stf-default-interconnect-5671-service-telemetry.apps.infra.watch
    port: 443
    role: edge
    sslProfile: sslProfile
    verifyHostname: false
```

4. collectd および AMQ Interconnect を設定にするには、以下のファイルを Red Hat OpenStack Platform director デプロイメントに追加します。
   - **stf-connectors.yaml** 環境ファイル
   - オーバークラウドのデプロイメント中に環境が使用されるようにする **enable-stf.yaml** ファイル
   - Ceilometer Telemetry が STF に送信されることを確認する **ceilometer-write-qdr.yaml** ファイル

```
openstack overcloud deploy <other arguments>
```

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

5. Red Hat OpenStack Platform オーバークラウドのデプロイ

3.3.3. クライアント側のインストールの検証

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 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。

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 コンテナー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. ローカルの AMQ インターコネクトへの接続のリストを返します。

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

Connections
  id   host                                                container
role    dir  security                        authentication  tenant

=========================================================================
=========================================================================
=========================================================================
=========================================================
  1    stf-default-interconnect-5671-service-telemetry.apps.infra.watch:443  stf-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
```

接続は 4 つあります。

- STF へのアウトバウンド接続
- collectd からのインバウンド接続
- ceilometer からのインバウンド接続
- 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 インターコネクトへの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
2979924  0      0    0    2    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.lSY6Mcicol4J2Kp      250  0    0     0      0
0       0      0    0    0    0    0     0
 endpoint  in   16     41                                    250  0    0     0      2979924  2979924
0      0    0    0    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     0
```

6. Red Hat OpenStack Platform ノードから STF へのアドレスを一覧表示するには、OCP 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=stf-default-interconnect

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

7. Pod に接続し、qdstat --connections コマンドを実行して既知の接続を一覧表示します。

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

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

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

========================================================================================
========================================================================================
=============================================
  1   10.129.2.21:43062  rcv[stf-default-collectd-telemetry-smartgateway-79c967c8f7-
kq4qv]     normal  in   no-security                    anonymous-user          000:00:00:00
006:21:50:25
  2   10.130.0.52:55754  rcv[stf-default-ceilometer-notification-smartgateway-
6675df547mbjk5]  normal  in   no-security                    anonymous-user
000:21:25:57  006:21:49:36
  3   10.130.0.51:43110  rcv[stf-default-collectd-notification-smartgateway-698c87fbb7-
f28v6]  normal  in   no-security                    anonymous-user          000:21:36:53
006:21:49:09
  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
```

この例では、Red Hat OpenStack Platform 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。

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 stf-default-interconnect-7458fd4d69-bgzfb -- qdstat --address

2020-04-21 18:20:10.293258 UTC
stf-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       1,553     1,553
0     0
  mobile  collectd/notify                0    multicast  -    1      0       10        10        0     0
  mobile  collectd/telemetry             0    multicast  -    1      0       7,798,049 7,798,049 0
0
```

第4章 高度な機能

以下のオプションの機能は、Service Telemetry Framework (STF) に他の機能を追加できます。

- デプロイメントのカスタマイズ詳細は、「「デプロイメントのカスタマイズ」」を参照してください。
- Alerts詳細は、「「アラート」」を参照してください。
- 高可用性。詳細は、「「高可用性」」を参照してください。
- ダッシュボード。詳細は、「「ダッシュボード」」を参照してください。
- 複数のクラウド。詳細は、「「複数のクラウドの設定」」を参照してください。
- 一時ストレージ。詳細は、「「一時ストレージ」」を参照してください。

4.1. デプロイメントのカスタマイズ

Service Telemetry Operator は、Red Hat OpenShift Container Platform (OCP) に読み込む **ServiceTelemetry** マニフェストを監視します。その後、Operator が他のオブジェクトをメモリーに作成することで、依存する Operator は、管理対象のワークロードを作成します。

警告

マニフェストを上書きする時点で、オブジェクト名または名前空間を含むマニフェスト全体を指定する必要があります。マニフェストを上書きする時に、パラメーターは動的に置き換えられません。

Service Telemetry Framework (STF) でマニフェストを正常に上書きするには、コアオプションのみを使用してデフォルトの環境をデプロイします。コアオプションの詳細は、「OCP での ServiceTelemetry オブジェクトの作成」 を参照してください。STF をデプロイする場合には、**oc get** コマンドを使用してデフォルトのデプロイされたマニフェストを取得します。Service Telemetry Operator で最初に生成されたマニフェストを使用する場合には、このマニフェストは Operator が管理する他のオブジェクトと互換性があります。

たとえば、**metricsEnabled: true** パラメーターが ServiceTelemetry マニフェストで設定されている場合、**Service Telemetry** Operator はデフォルトマニフェストを使用してメトリクスの取得およびストレージのコンポーネントを要求します。デフォルトのマニフェストを上書きする場合があります。詳細は、「「マニフェストの上書きパラメーター」」を参照してください。

4.1.1. マニフェストの上書きパラメーター

以下の表には、マニフェストの上書きに使用できるパラメーターと、対応する取得コマンドが記載されています。

表4.1 マニフェストの上書きパラメーター

パラメーターの上書き	説明	取得コマンド
alertmanagerManifest	**Alertmanager** オブジェクトの作成を上書きします。Prometheus Operator は **Alertmanager** オブジェクトを監視します。	**oc get alertmanager stf-default -oyaml**
alertmanagerConfigManifest	Alertmanager 設定が含まれる **Secret** を上書きします。Prometheus Operator は **alertmanager-{{ alertmanager-name }}** という名前のシークレット（例： **stf-default** ）を使用して **alertmanager.yaml** 設定を Alertmanager に提供します。	**oc get secret alertmanager-stf-default -oyaml**
elasticsearchManifest	**ElasticSearch** オブジェクトの作成を上書きします。Elastic Cloud on Kuberneters Operator は **ElasticSearch** オブジェクトを監視します。	**oc get elasticsearch elasticsearch -oyaml**
interconnectManifest	**Interconnect** オブジェクトの作成を上書きします。AMQ Interconnect Operator は **Interconnect** オブジェクトを監視します。	**oc get interconnect stf-default-interconnect -oyaml**
prometheusManifest	**Prometheus** オブジェクトの作成を上書きします。Prometheus Operator は **Prometheus** オブジェクトを監視します。	**oc get prometheus stf-default -oyaml**
servicemonitorManifest	**ServiceMonitor** オブジェクトの作成を上書きします。Prometheus Operator は **ServiceMonitor** オブジェクトを監視します。	**oc get servicemonitor stf-default -oyaml**
smartgatewayCollectdMetricsManifest	collectd メトリクスの **SmartGateway** オブジェクト作成を上書きします。Smart Gateway Operator は **SmartGateway** オブジェクトを監視します。	**oc get smartgateway stf-default-collectd-telemetry -oyaml**

パラメーターの上書き	説明	取得コマンド
smartgatewayCollectdEventsManifest	collectd イベントの **SmartGateway** オブジェクト作成を上書きします。Smart Gateway Operator は **SmartGateway** オブジェクトを監視します。	**oc get smartgateway stf-default-collectd-notification -oyaml**
smartgatewayCeilometerEventsManifest	Ceilometer イベントの **SmartGateway** オブジェクト作成を上書きします。Smart Gateway Operator は **SmartGateway** オブジェクトを監視します。	**oc get smartgateway stf-default-ceilometer-notification -oyaml**

4.1.2. 管理マニフェストの上書き

ServiceTelemetry オブジェクトを編集し、パラメーターとマニフェストを指定します。利用可能なマニフェストオーバーライドパラメーターの一覧は、「デプロイメントのカスタマイズ」 を参照してください。デフォルトの ServiceTelemetry オブジェクトは stf-default です。oc get servicetelemetry を使用して、利用可能な STF デプロイメントを一覧表示します。

ヒント

oc edit コマンドは、デフォルトのシステムエディターを読み込みます。デフォルトのエディターを上書きするには、環境変数 EDITOR を優先エディターに指定するか、または設定します。たとえば、EDITOR=nano oc edit servicetelemetry stf-default 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. **ServiceTelemetry オブジェクトをエディターに読み込みます。**

```
oc edit servicetelemetry stf-default
```

4. **ServiceTelemetry オブジェクトを変更するには、マニフェストの上書きパラメーターと、STF が提供するデフォルトではなく、OCP に書き込むマニフェストの内容を指定します。**

注記

マニフェストの上書きパラメーターの入力した後に表示される末尾のパイプ (|) は、指定した値が複数行であることを示します。

```
$ oc edit servicetelemetry stf-default

apiVersion: infra.watch/v1alpha1
kind: ServiceTelemetry
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"infra.watch/v1alpha1","kind":"ServiceTelemetry","metadata":
{"annotations":{},"name":"stf-default","namespace":"service-telemetry"},"spec":
{metricsEnabled":true}}
  creationTimestamp: "2020-04-14T20:29:42Z"
  generation: 1
  name: stf-default
  namespace: service-telemetry
  resourceVersion: "1949423"
  selfLink: /apis/infra.watch/v1alpha1/namespaces/service-
telemetry/servicetelemetrys/stf-default
  uid: d058bc41-1bb0-49f5-9a8b-642f4b8adb95
spec:
  metricsEnabled: true
  smartgatewayCollectdMetricsManifest: | (1)
    apiVersion: smartgateway.infra.watch/v2alpha1
    kind: SmartGateway
    metadata:
      name: stf-default-collectd-telemetry
      namespace: service-telemetry
    spec:
      amqpUrl: stf-default-interconnect.service-
telemetry.svc.cluster.local:5672/collectd/telemetry
      debug: true
      prefetch: 15000
      serviceType: metrics
      size: 1
      useTimestamp: true (2)
status:
  conditions:
  - ansibleResult:
      changed: 0
      completion: 2020-04-14T20:32:19.079508
      failures: 0
      ok: 52
      skipped: 1
    lastTransitionTime: "2020-04-14T20:29:59Z"
    message: Awaiting next reconciliation
    reason: Successful
    status: "True"
    type: Running
```

(1) マニフェストの上書きパラメーターは、ServiceTelemetry オブジェクトの spec で定義されます。

(2) マニフェストの上書き内容の最後

5. 保存して閉じます。

4.2. 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。

関連情報
Prometheus と Alertmanager によるアラートまたは通知の詳細については、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例を見るには、https://github.com/infrawatch/service-telemetry-operator/tree/master/deploy/alerts を参照してください。

4.2.1. 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て通知を行います。ルール条件が空の結果セットを返す場合は、条件は偽となります。それ以外の場合は、ルールが真となり、アラートが発生します。

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. 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: stf-default
    role: alert-rules
  name: prometheus-alarm-rules
  namespace: service-telemetry
spec:
  groups:
    - name: ./openstack.rules
      rules:
        - alert: Metric Listener down
```

```
        expr: collectd_qpid_router_status < 1 # To change the rule, edit the value of the
expr parameter.
EOF
```

4. Operator でルールが Prometheus にロードされている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
```

5. curl を実行して prometheus-operated サービスにアクセスし、メモリーにロードされたルールを返します。

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

6. 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
```

7. curl Pod を削除して環境を消去します。

```
$ oc delete pod curl

pod "curl" deleted
```

関連情報
アラートの詳細については、https://github.com/coreos/prometheus-operator/blob/master/Documentation/user-guides/alerting.md を参照してください。

4.2.2. カスタムアラートの設定

カスタムアラートを、「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

1. oc edit コマンドを使用します。

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

2. PrometheusRules マニフェストを編集します。

3. 保存して閉じます。

関連情報
アラートルールの設定についての詳細は、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。

4.2.3. 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 (OCP) シークレットとして管理します。デフォルトでは、STF は、レシーバーを持たない基本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。詳細は、「「管理マニフェストの上書き」」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. STF デプロイメントの **ServiceTelemetry** オブジェクトを編集します。

```
oc edit servicetelemetry stf-default
```

4. 新規パラメーター **alertmanagerConfigManifest** および **Secret** オブジェクトコンテンツを追加して、Alertmanager の **alertmanager.yaml** 設定を定義します。

注記

これにより、Service Telemetry Operator によってすでに管理されているデフォルトのテンプレートが読み込まれます。変更が正しく設定されることを確認するには、値を変更して **alertmanager-stf-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、**global.resolve_timeout** の値を **5m** から **10m** に変更します。

```
apiVersion: infra.watch/v1alpha1
kind: ServiceTelemetry
metadata:
  name: stf-default
```

```
  namespace: service-telemetry
spec:
  metricsEnabled: true
  alertmanagerConfigManifest: |
    apiVersion: v1
    kind: Secret
    metadata:
      name: 'alertmanager-stf-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-stf-default -o go-template='{{index .data
"alertmanager.yaml" | base64decode }}'

global:
  resolve_timeout: 10m
route:
  group_by: ['job']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: 'null'
receivers:
- name: 'null'
```

6. 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
```

7. 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 にロードされた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: []\n",...}}
```

8. configYAML フィールドに想定の変更が追加されていることを確認します。Pod を終了します。

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

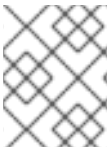
9. 環境を消去するには、curl Pod を削除します。

```
$ oc delete pod curl

pod "curl" deleted
```

関連情報
Red Hat OpenShift Container Platform シークレットおよび Prometheus Operator 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。

4.3. 高可用性

高可用性とは、コンポーネントサービスで障害から迅速に回復する Service Telemetry Framework (STF) の機能です。Red Hat OpenShift Container Platform (OCP) 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。OCP ノードを障害から保護するには、3 つ以上のノードが含まれる OCP クラスターに STF をデプロイします。



注記

STF はまだ完全な耐障害性システムではありません。復旧期間中のメトリクスや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、以下のような効果があります。

- 2 つの AMQ Interconnect Pod がデフォルト 1 ではなく実行されます。
- 3 つの ElasticSearch Pod はデフォルト 1 ではなく実行されます。
- これらのサービスのいずれかで失われた Pod からの復旧時間は、約 2 秒減らします。

4.3.1. 高可用性の設定

高可用性に STF を設定するには、OCP の ServiceTelemetry オブジェクトに **highAvailabilityEnabled: 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 コマンドで ServiceTelemetry オブジェクトを編集します。

```
$ oc edit ServiceTelemetry
```

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

```
spec:
  eventsEnabled: true
  metricsEnabled: true
  highAvailabilityEnabled: true
```

5. 変更内容を保存し、オブジェクトを閉じます。

4.4. ダッシュボード

サードパーティーアプリケーション Grafana を使用して、collectd が収集した個別のホストノードに関するシステムレベルのメトリクスを可視化します。collectd の設定に関する詳細は、「Service Telemetry Framework 向けの Red Hat OpenStack Platform オーバークラウドの設定」を参照してください。

4.4.1. ダッシュボードをホストする Grafana の設定

Grafana はデフォルトの Service Telemetry Framework (STF) のデプロイメントに含まれていないため、Operator Hub.io から Grafana Operator をデプロイする必要があります。

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. ダッシュボードリポジトリーのクローンを作成します。

```
git clone https://github.com/infrawatch/dashboards
cd dashboards
```

4. Grafana オペレーターをデプロイします。

```
oc create -f deploy/subscription.yaml
```

5. Operator が正常に起動したことを確認するには、oc get csv コマンドを実行します。PHASE 列の値が Succeeded の場合、Operator は正常に起動します。

```
$ oc get csv

NAME                    DISPLAY                         VERSION   REPLACES
PHASE
```

```
grafana-operator.v3.2.0        Grafana Operator                3.2.0
Succeeded
...
```

6. Grafana インスタンスを起動します。

```
$ oc create -f deploy/grafana.yaml
```

7. Grafana インスタンスがデプロイされたことを確認します。

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

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

8. データソースおよびダッシュボードリソースを作成します。

```
oc create -f deploy/datasource.yaml \
   -f deploy/rhos-dashboard.yaml
```

9. リソースが正しくインストールされたことを確認します。

```
$ oc get grafanadashboards

NAME            AGE
rhos-dashboard   7d21h

$ oc get grafanadatasources
NAME                                AGE
service-telemetry-grafanadatasource   1m
```

10. Web ブラウザーで、https://<grafana-route-address> に移動します。**oc get routes** コマンドを使用して、Grafana ルートアドレスを取得します。

```
oc get routes
```

11. ダッシュボードを表示するには、Dashboards および Manage をクリックします。

関連情報

- OperatorHub.io カタログソースの有効化に関する詳細は、「OperatorHub.io Community Catalog Source の有効化」 を参照してください。

4.4.1.1. クエリーの表示および編集

手順

1. Red Hat OpenShift Container Platform にログインします。クエリーを表示し、編集するには、admin ユーザーとしてログインします。

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、oc describe を使用して Grafana オブジェクトを記述します。**

```
oc describe grafana service-telemetry-grafana
```

4.4.2. Grafana 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には、一度に単一ノードのメトリクスが表示されます。ダッシュボードの左上からノードを選択します。

4.4.2.1. トップパネル

Title	単位	説明
現在のグローバルアラート	-	Prometheus によって発生する現在のアラート
最近のグローバルアラート	-	最近実行されたアラートの 5m 時間手順
ステータスパネル	-	ノードのステータス： up、down、Unavailable
アップタイム	s/m/h/d/M/Y	ノードの合計稼働時間
CPU コア	cores	コアの合計数
メモリー	bytes	メモリー合計
ディスクサイズ	bytes	ストレージ合計サイズ
プロセス	processes	タイプ別に一覧表示されるプロセスの合計数
負荷平均	processes	負荷平均は、カーネル実行キューにある実行中および割り込み不可能なプロセスの平均数を表します。

4.4.2.2. ネットワークパネル

ノードのネットワークインターフェースを表示するパネル。

パネル	単位	説明
物理インターフェースの Ingress エラー	errors	受信データを含む合計エラー

物理インターフェースの Egress エラー	errors	送信データを含む合計エラー
物理インターフェース Ingress エラーレート	errors/s	受信データエラーの割合
物理インターフェースの egress エラー率	errors/s	送信データエラーの割合
物理インターフェースのパケット送信の 1 秒あたりの着信パケット	物理インターフェースのパケット (Egress)	pps
1 秒あたりの送信パケット	物理インターフェースデータ Ingress	bytes/s
受信データレート	物理インターフェースデータ Egress	bytes/s
送信データレート	物理インターフェースドロップレート Ingress	pps
着信パケットのドロップレート	物理インターフェースドロップレート Egress	pps

4.4.2.3. CPU パネル

ノードの CPU 使用率を表示するパネル。

パネル	単位	説明
現在の CPU 使用率	percent	最後のクエリー時の即時使用。
CPU 使用率の集約	percent	ノード上のすべてのコアの非アイドル CPU アクティビティーの平均。
Aggr.タイプ別の CPU 使用率	percent	すべてのコアで平均するスレッドの各タイプに費やされた時間を表示します。

4.4.2.4. メモリーパネル

ノードのメモリー使用量を表示するパネル。

パネル	単位	説明
使用されているメモリー	percent	最後のクエリー時に使用されているメモリーの量。

パネル	単位	説明
使用される Huge Page	hugepages	使用されている hugepages の数。 メモリー

4.4.2.5. ディスク/ファイルシステム

ディスクに使用されている領域を表示するパネル。

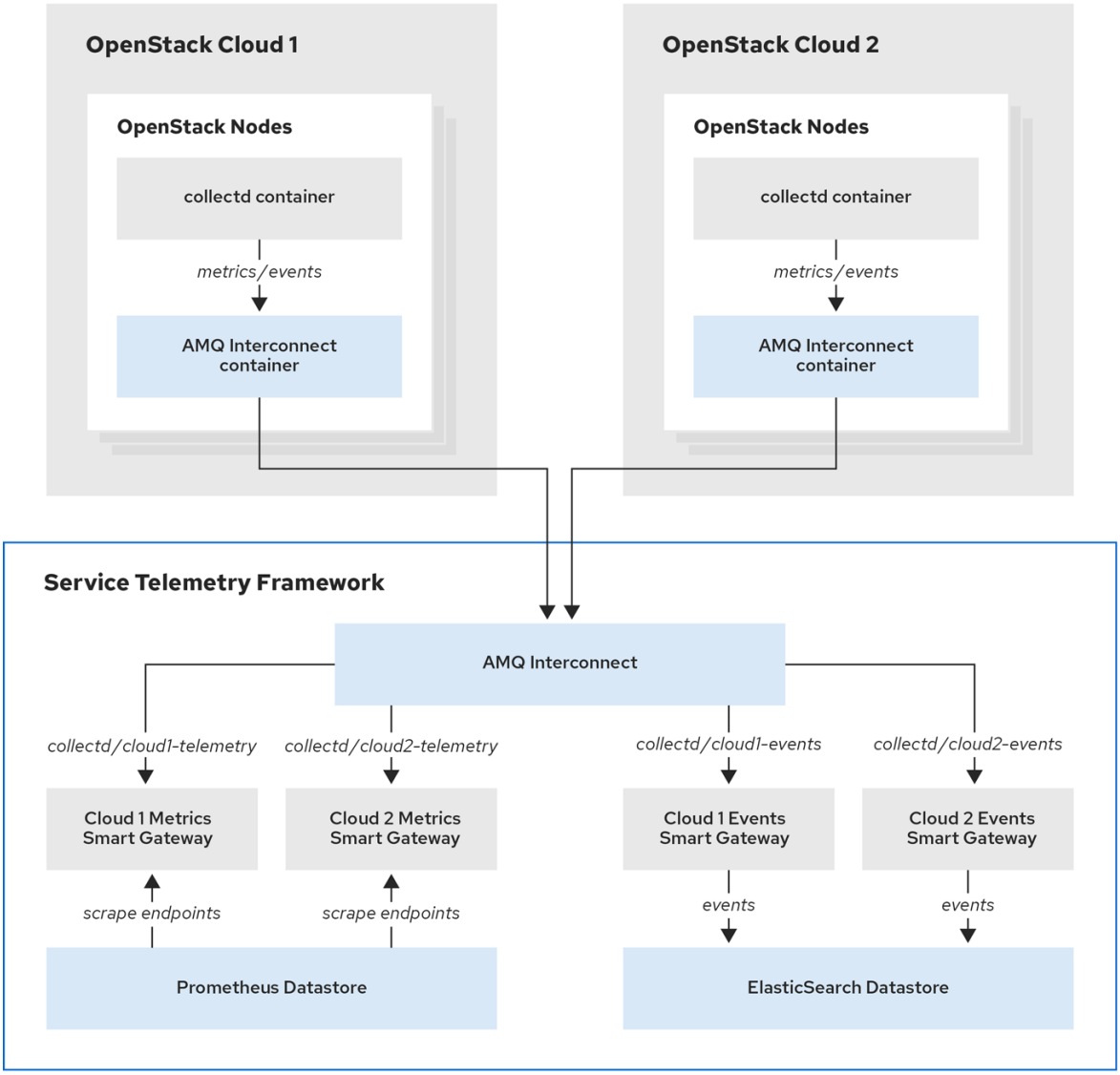
パネル	単位	説明	注記
ディスク領域の使用量	percent	最後のクエリー時のディスク使用量の合計。	
inode の使用方法	percent	最後のクエリー時の inode の合計使用	
容量の使用量の集約	bytes	使用量および予約済みディスク容量の合計。	このクエリーは **df** プラグインに依存するため、ディスク領域を使用しない一時ファイルシステムは結果に含まれます。クエリーはこれらの大半を除外しようとしますが、網羅的なものではない場合があります。
ディスクトラフィック	bytes/s	は、読み取りおよび書き込みの両方のレートを示します。	
ディスクの読み込み	percent	使用されているディスク合計帯域幅の概算率。加重 I/O 時系列には、累積される可能性のあるバックログが含まれます。詳細は、collectd ディスクプラグインのドキュメントを参照してください。	
操作/s	ops/s	1 秒あたりの操作	
平均 I/O 操作時間	秒	各 I/O 操作の完了にかかった平均時間。この平均は正確ではありません。collectd ディスクプラグインのドキュメントを参照してください。	

4.5. 複数のクラウドの設定

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 クラウドを設定できます。

1. 各クラウドで使用する AMQP アドレスのプレフィックスを計画します。詳細は、「「AMQP アドレスプレフィックスの計画」」を参照してください。
2. メトリクスとイベントのコンシューマーである Smart Gateway を各クラウドに配備し、対応するアドレスプレフィックスをリッスンします。詳細は、「「Smart Gateway の導入」」を参照してください。
3. 各クラウドがメトリクスやイベントを正しいアドレスで STF に送信するように設定します。詳細は、「「OpenStack 環境ファイルの作成」」を参照してください。

図4.1 STF に接続する Red Hat OpenStack Platform クラウド 2 つ



65_OpenStack_0120

4.5.1. AMQP アドレスプレフィックスの計画

デフォルトでは、Red Hat OpenStack Platform 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。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。ここで、STF Smart Gateway はこれらのアドレスをリッスンしてデータを監視します。

複数のクラウドをサポートし、どのクラウドが監視データを生成したかを識別するために、各クラウドがデータを固有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。以下のリストは、アドレスと識別子の例です。

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

4.5.2. Smart Gateway の導入

各クラウドの各データ収集タイプに Smart Gateway をデプロイする必要があります。1つは collectd メトリクス用、もう 1つは collectd イベント用、もう 1つは Ceilometer イベントに 1つです。各 Smart Gateway は、対応するクラウドに定義された AMQP アドレスをリッスンするように設定します。

STF を初めてデプロイする際は、1つのクラウドに対する初期の 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。初期 Smart Gateway は、追加の Smart Gateway を作成するためのテンプレートとして機能し、データストアへの接続に必要な情報も作成します。

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. 追加の Smart Gateways のテンプレートとして、最初にデプロイされた Smart Gateways を使用します。oc get smartgateways コマンドを使用して、現在デプロイされている Smart Gateway を一覧表示します。たとえば、metricsEnabled: true および eventsEnabled: true で STF をデプロイした場合、以下の Smart Gateways が出力に表示されます。

```
$ oc get smartgateways

NAME                                AGE
stf-default-ceilometer-notification   14d
stf-default-collectd-notification     14d
stf-default-collectd-telemetry        14d
```

4. 各 Smart Gateway のマニフェストを取得して、コンテンツを一時ファイルに保存します。このファイルは後で変更し、これを使用して新しい Smart Gateways を作成できます。

```
truncate --size 0 /tmp/cloud1-smartgateways.yaml && \
for sg in $(oc get smartgateways -oname)
do
  echo "---" >> /tmp/cloud1-smartgateways.yaml
  oc get ${sg} -oyaml --export >> /tmp/cloud1-smartgateways.yaml
done
```

5. **/tmp/cloud1-smartgateways.yaml** ファイルの Smart Gateway マニフェストを変更します。**metadata.name** および **spec.amqpUrl** フィールドを調整し、スキーマからクラウド識別子を追加します。詳細は、「「AMQP アドレスプレフィックスの計画」」を参照してください。Smart Gateway マニフェストのサンプルを表示するには、<<example-manifests_advanced-features[] を参照してください。

6. 新しい Smart Gateways をデプロイします。

```
oc apply -f /tmp/cloud1-smartgateways.yaml
```

7. 各 Smart Gateway が起動していることを確認します。この作業は、Smart Gateway の台数によっては数分かかることがあります。

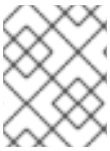
```
oc get po -l app=smart-gateway
```

4.5.2.1. マニフェストの例

重要

以下の例の内容は、デプロイメントのファイルの内容とは異なる場合があります。デプロイメントのマニフェストをコピーします。

各 Smart Gateway の **name** および **amqpUrl** パラメーターが、クラウドに使用する名前と一致していることを確認します。詳細は、「「AMQP アドレスプレフィックスの計画」」を参照してください。



注記

出力には、OCP に読み込むマニフェストから削除できる追加の メタデータ パラメーターが含まれる場合があります。

```
apiVersion: smartgateway.infra.watch/v2alpha1
kind: SmartGateway
metadata:
  name: stf-default-ceilometer-notification-cloud1 (1)
spec:
  amqpDataSource: ceilometer
  amqpUrl: stf-default-interconnect.service-
telemetry.svc.cluster.local:5672/anycast/ceilometer/cloud1-event.sample (2)
  debug: false
  elasticPass: fkzfhghw.....
  elasticUrl: https://elasticsearch-es-http.service-telemetry.svc.cluster.local:9200
  elasticUser: elastic
```

```
  resetIndex: false
  serviceType: events
  size: 1
  tlsCaCert: /config/certs/ca.crt
  tlsClientCert: /config/certs/tls.crt
  tlsClientKey: /config/certs/tls.key
  tlsServerName: elasticsearch-es-http.service-telemetry.svc.cluster.local
  useBasicAuth: true
  useTls: true
---
apiVersion: smartgateway.infra.watch/v2alpha1
kind: SmartGateway
metadata:
  name: stf-default-collectd-notification-cloud1 3
spec:
  amqpDataSource: collectd
  amqpUrl: stf-default-interconnect.service-telemetry.svc.cluster.local:5672/collectd/cloud1-
notify 4
  debug: false
  elasticPass: fkzfhghw.....
  elasticUrl: https://elasticsearch-es-http.service-telemetry.svc.cluster.local:9200
  elasticUser: elastic
  resetIndex: false
  serviceType: events
  size: 1
  tlsCaCert: /config/certs/ca.crt
  tlsClientCert: /config/certs/tls.crt
  tlsClientKey: /config/certs/tls.key
  tlsServerName: elasticsearch-es-http.service-telemetry.svc.cluster.local
  useBasicAuth: true
  useTls: true
---
apiVersion: smartgateway.infra.watch/v2alpha1
kind: SmartGateway
metadata:
  name: stf-default-collectd-telemetry-cloud1 5
spec:
  amqpUrl: stf-default-interconnect.service-telemetry.svc.cluster.local:5672/collectd/cloud1-
telemetry 6
  debug: false
  prefetch: 15000
  serviceType: metrics
  size: 1
  useTimestamp: true
```

1. cloud1の Ceilometer 通知の名前
2. cloud1の Ceilometer 通知用の AMQP アドレス
3. cloud1の collectd Telemetry の名前
4. cloud1の collectd Telemetry の AMQP アドレス
5. cloud1の collectd 通知の名前
6. cloud1の collectd 通知用の AMQP アドレス

4.5.3. OpenStack 環境ファイルの作成

発信元のクラウドに応じてトラフィックをラベリングするには、クラウド固有のインスタンス名を持つ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、AMQP アドレスプレフィックススキームと一致するように CeilometerQdrEventsConfig および CollectdAmqpInstances の値を調整します。詳細は、「「AMQP アドレスプレフィックスの計画」」を参照してください。

警告

オーバークラウドのデプロイメントから、enable-stf.yaml および ceilometer-write-qdr.yaml 環境ファイルの参照を削除します。この設定は冗長であるため、各クラウドノードから重複した情報が送信されます。

手順

1. stf-connectors.yaml ファイルを作成し、このクラウドデプロイメントの AMQP アドレスに一致するように変更します。

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

parameter_defaults:
  EnableSTF: true

  EventPipelinePublishers: []
  CeilometerEnablePanko: false
  CeilometerQdrPublishEvents: true
  CeilometerQdrEventsConfig:
    driver: amqp
    topic: cloud1-event   (1)

  CollectdConnectionType: amqp1
  CollectdAmqpInterval: 5
  CollectdDefaultPollingInterval: 5

  CollectdAmqpInstances:
    cloud1-notify:   (2)
      notify: true
```

```
      format: JSON
      presettle: false
    cloud1-telemetry:    3
      format: JSON
      presettle: true

MetricsQdrAddresses:
  - prefix: collectd
    distribution: multicast
  - prefix: anycast/ceilometer
    distribution: multicast

MetricsQdrSSLProfiles:
  - name: sslProfile

MetricsQdrConnectors:
  - host: stf-default-interconnect-5671-service-telemetry.apps.infra.watch    4
    port: 443
    role: edge
    verifyHostname: false
    sslProfile: sslProfile
```

+ <1> Ceilometer イベントのトピックを定義します。この値は、エニーキャスト/ceilometer/cloud1-event.sample のアドレス形式です。 <2> collectd イベント用のトピックを定義します。この値は、collectd/cloud1-notify の形式です。<3> collectd メトリクスのトピックを定義します。この値は、collectd/cloud1-telemetry の形式です。<4> MetricsQdrConnectors ホストを STF ルートのアドレスに調整します。

1. stf-connectors.yaml ファイルの命名規則が、Smart Gateway 設定の spec.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。

2. カスタム環境ファイル（例: /home/stack/custom_templates/）にファイルを保存します。

3. 認証ファイルのソース

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

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

4. 実際の環境に該当するその他の環境ファイルと共に、stf-connectors.yaml ファイルを overcloud deployment コマンドに含めます。

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

関連情報
デプロイメントの検証に関する詳細は、「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。

4.5.4. 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、特定のクラウドのデータを照会することができます。

特定のクラウドからのデータをクエリーするには、関連付けられた サービス ラベルと一致する Prometheus **promql** クエリーを使用します（例： **collectd_uptime{service="stf-default-collectd-telemetry-cloud1-smartgateway"}**）。

4.6. 一時ストレージ

一時ストレージを使用して、データを Red Hat OpenShift Container Platform(OCP)クラスターに永続的に保存することなく Service Telemetry Framework(STF)を実行します。適切で設計通りに動作している場合に、プラットフォームのデータの自発性により、一時ストレージは本番環境では推奨されません。たとえば、Pod を再起動するか、またはワークロードを別のノードに再スケジュールすると、Pod の起動後に書き込まれたローカルデータが失われます。

STF で一時ストレージを有効にする場合、Service Telemetry Operator は関連する ストレージセクションをデータストレージ コンポーネントマニフェストに追加しません。

4.6.1. 一時ストレージの設定

一時ストレージ用に STF を設定するには、ストレージ**EphemeralEnabled: true** を OCP の ServiceTelemetry オブジェクトに追加します。インストール時に **storageEphemeralEnabled: true** を追加するか、または STF をすでにデプロイしている場合は、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. Service Telemetry オブジェクトを編集します。

```
$ oc edit ServiceTelemetry stf-default
```

4. **storageEphemeralEnabled: true** パラメーターを **spec** セクションに追加します。

```
spec:
  eventsEnabled: true
  metricsEnabled: true
  storageEphemeralEnabled: true
```

5. 変更内容を保存し、オブジェクトを閉じます。

付録A COLLECTD プラグイン

本項では、Red Hat OpenStack Platform 16.0 用の collectd プラグインおよび設定の完全な一覧について説明します。

collectd-aggregation

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

collectd-amqp1

collectd-apache

- collectd::plugin::apache::instances（例： {localhost TEMPLATES {url gitops http://localhost/mod_status?auto}}）
- collectd::plugin::apache::interval

collectd-apcups

collectd-battery

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

collectd-ceph

- collectd::plugin::ceph::daemons
- collectd::plugin::ceph::longrunavglatency
- collectd::plugin::ceph::convertspecialmetrictypes

collectd-cgroups

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

collectd-conntrack

- None

collectd-contextswitch

- collectd::plugin::contextswitch::interval

collectd-cpu

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

collectd-cpufreq

- None

collectd-cpusleep

collectd-csv

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

collectd-df

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

collectd-disk

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

- collectd::plugin::disk::interval

collectd-entropy

- collectd::plugin::entropy::interval

collectd-ethstat

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

collectd-exec

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

collectd-fhcount

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

collectd-filecount

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

collectd-fscache

- None

collectd-hddtemp

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

collectd-hugepages

- collectd::plugin::hugepages::report_per_node_hp

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

collectd-intel_rdt

collectd-interface

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

collectd-ipc

- None

collectd-ipmi

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

collectd-irq

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

collectd-load

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

collectd-logfile

- collectd::plugin::logfile::log_level

- collectd::plugin::logfile::log_file
- collectd::plugin::logfile::log_timestamp
- collectd::plugin::logfile::print_severity
- collectd::plugin::logfile::interval

collectd-madwifi

collectd-mbmon

collectd-md

collectd-memcached

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

collectd-memory

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

collectd-multimeter

collectd-mysql

- collectd::plugin::mysql::interval

collectd-netlink

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

collectd-network

- collectd::plugin::network::timetolive
- collectd::plugin::network::maxpacketsize

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

collectd-nfs

- collectd::plugin::nfs::interval

collectd-ntpd

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

collectd-numa

- None

collectd-olsrd

collectd-openvpn

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

collectd-ovs_events

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

- collectd::plugin::ovs_events::$port
- collectd::plugin::ovs_events::socket

collectd-ovs_stats

- collectd::plugin::ovs_stats::address
- collectd::plugin::ovs_stats::bridges
- collectd::plugin::ovs_stats::port
- collectd::plugin::ovs_stats::socket

collectd-ping

- collectd::plugin::ping::hosts
- collectd::plugin::ping::timeout
- collectd::plugin::ping::ttl
- collectd::plugin::ping::source_address
- collectd::plugin::ping::device
- collectd::plugin::ping::max_missed
- collectd::plugin::ping::size
- collectd::plugin::ping::interval

collectd-powerdns

- collectd::plugin::powerdns::interval
- collectd::plugin::powerdns::servers
- collectd::plugin::powerdns::recursors
- collectd::plugin::powerdns::local_socket
- collectd::plugin::powerdns::interval

collectd-processes

- collectd::plugin::processes::processes
- collectd::plugin::processes::process_matches
- collectd::plugin::processes::collect_context_switch
- collectd::plugin::processes::collect_file_descriptor
- collectd::plugin::processes::collect_memory_maps
- collectd::plugin::powerdns::interval

collectd-protocols

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

collectd-python

collectd-serial

collectd-smart

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

collectd-snmp_agent

collectd-statsd

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

collectd-swap

- collectd::plugin::swap::reportbydevice
- collectd::plugin::swap::reportbytes
- collectd::plugin::swap::valuesabsolute

- collectd::plugin::swap::valuespercentage
- collectd::plugin::swap::reportio
- collectd::plugin::swap::interval

collectd-syslog

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

collectd-table

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

collectd-tail

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

collectd-tail_csv

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

collectd-tcpconns

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

collectd-ted

collectd-thermal

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

collectd-threshold

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

collectd-turbostat

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

collectd-unixsock

collectd-uptime

- collectd::plugin::uptime::interval

collectd-users

- collectd::plugin::users::interval

collectd-uuid

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

collectd-virt

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

- collectd::plugin::virt::interface_format
- collectd::plugin::virt::extra_stats
- collectd::plugin::virt::interval

collectd-vmem

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

collectd-vserver

collectd-wireless

collectd-write_graphite

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

collectd-write_kafka

- collectd::plugin::write_kafka::kafka_host
- collectd::plugin::write_kafka::kafka_port
- collectd::plugin::write_kafka::kafka_hosts
- collectd::plugin::write_kafka::topics

collectd-write_log

- collectd::plugin::write_log::format

collectd-zfs_arc

- None