Red Hat OpenStack Platform 16.2

Service Telemetry Framework 1.5

Service Telemetry Framework 1.5 のインストールおよびデプロイ

Last Updated: 2024-03-15

Red Hat OpenStack Platform 16.2 Service Telemetry Framework 1.5

Service Telemetry Framework 1.5 のインストールおよびデプロイ

OpenStack Team
rhos-docs@redhat.com

法律上の通知

概要

コアコンポーネントをインストールし、Service Telemetry Framework 1.5 をデプロイします。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに対するご意見をお聞かせください。ドキュメントの改善点があればお知らせください。

Jira でドキュメントのフィードバックを提供する

ドキュメントに関するフィードバックを提供するには、Create Issue フォームを使用します。Red Hat OpenStack Platform Jira プロジェクトで Jira Issue が作成され、フィードバックの進行状況を追跡できます。

1. Jira にログインしていることを確認してください。Jira アカウントをお持ちでない場合は、アカウントを作成してフィードバックを送信してください。
2. Create Issue をクリックして、**Create Issue** ページを開きます。
3. **Summary** フィールドと **Description** フィールドに入力します。**Description** フィールドに、ドキュメントの URL、章またはセクション番号、および問題の詳しい説明を入力します。フォーム内の他のフィールドは変更しないでください。
4. **Create** をクリックします。

第1章 SERVICE TELEMETRY FRAMEWORK 1.5 の概要

Service Telemetry Framework (STF) は、Red Hat OpenStack Platform (RHOSP) 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。

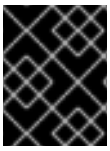
メトリクス

アプリケーションまたはシステムの数値測定。

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。

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 のコンテナーとしてデプロイされます。



重要

STF は、Red Hat OpenShift Container Platform Extended Update Support (EUS) リリースバージョン 4.12 および 4.14 と互換性があります。

関連情報

- Red Hat OpenShift Container Platform 製品ドキュメント
- サービステレメトリーフレームワークのパフォーマンスとスケーリング
- OpenShift Container Platform 4.14 ドキュメント
- Red Hat OpenShift Container Platform のライフサイクルポリシー

1.1. SERVICE TELEMETRY FRAMEWORK のサポート

Red Hat は、AMQ Interconnect、Cluster Observability Operator (Prometheus、Alertmanager)、Service Telemetry Operator、Smart Gateway Operator などのコア Operator とワークロードをサポートしています。Red Hat はコミュニティー Operator またはワークロードコンポーネントをサポートしておらず、そこには Elasticsearch、Grafana、およびそれらの Operator が含まれます。

Service Telemetry Framework (STF) は、完全に接続されたネットワーク環境または Red Hat OpenShift Container Platform の非接続環境にデプロイできます。STF をネットワークプロキシー環境にデプロイすることはできません。

STF のライフサイクルとサポートステータスの詳細については、サービステレメトリーフレームワークのサポート対象バージョンマトリックス を参照してください。

1.2. SERVICE TELEMETRY FRAMEWORK 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 (RHOSP) はクライアントであり、Red Hat OpenShift Container Platform はサーバーです。

デフォルトでは、STF はメトリクス情報を収集、伝送、および保存します。

RHOSP イベントデータを収集し、それをメッセージバスで伝送し、Smart Gateway からユーザー提供の Elasticsearch に転送できますが、このオプションは非推奨です。

STF は、以下のコンポーネントで設定されます。

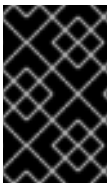
- データ収集
  - collectd: RHOSP 上のインフラストラクチャーメトリクスとイベントを収集します。
  - Ceilometer: RHOSP 上のメトリクスとイベントを収集します。
- 伝送
  - AMQ Interconnect: AMQP 1.x 互換のメッセージングバス。メトリクスを RHOSP から STF に転送して保存または伝送するための、高速で信頼性の高いデータ伝送を提供します。
  - Smart Gateway: AMQP 1.x バスからメトリクスとイベントを取得して、Prometheus または外部 Elasticsearch に配信する Golang 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。
  - 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。
- ユーザー提供のコンポーネント
  - 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。
  - Elasticsearch: Smart Gateway が受信および転送した RHOSP イベントを保存するイベントデータストレージ。

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	Server
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

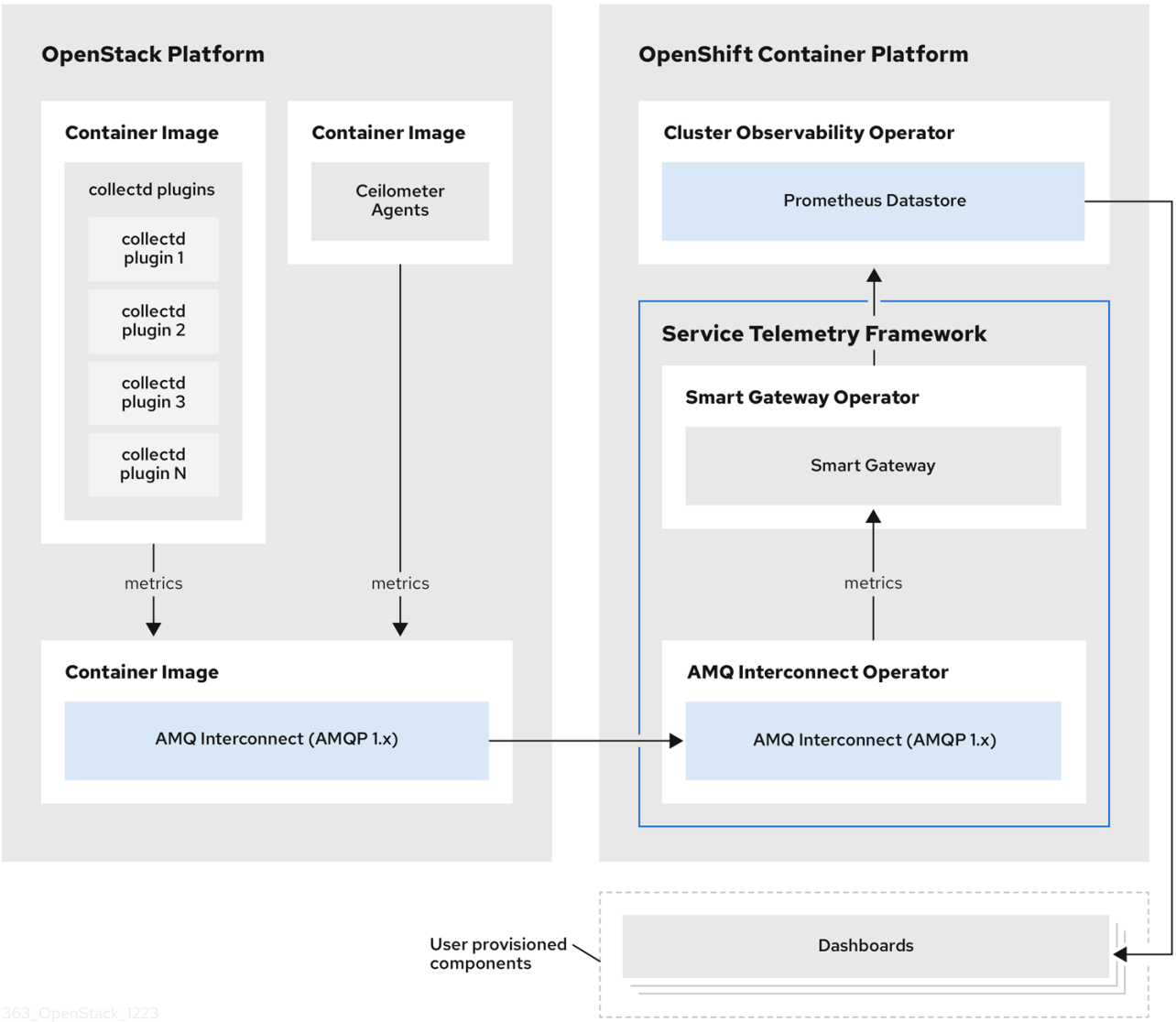
コンポーネント	クライアント	Server
Grafana	いいえ	はい
collectd	はい	いいえ
Ceilometer	はい	いいえ



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 アーキテクチャー概要



クライアント側のメトリクスの場合、collectd はプロジェクトデータなしで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 RHOSP プラットフォームデータを提供します。Ceilometer も collectd も、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 Prometheus に配信します。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。

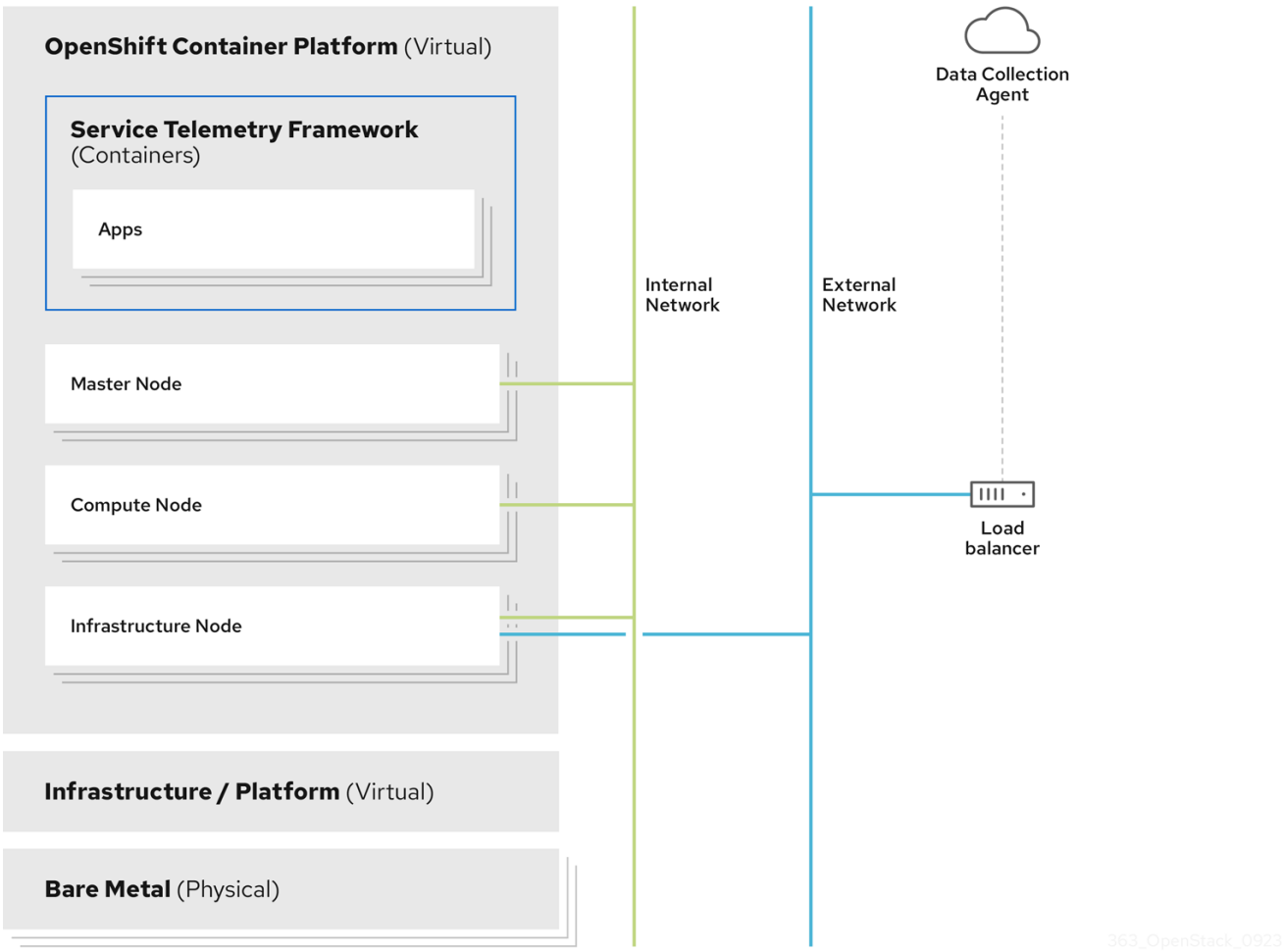
イベントを収集して保存する場合は、collectd および Ceilometer が AMQ Interconnect トランスポートを使用し、イベントデータをサーバー側に渡します。別の Smart Gateway は、ユーザーが提供する Elasticsearch データストアにデータを転送します。

サーバー側の STF 監視インフラストラクチャーは、以下のレイヤーで設定されています。

- Service Telemetry Framework 1.5
- Red Hat OpenShift Container Platform Extended Update Support (EUS) リリース 4.12 および 4.14
- インフラストラクチャープラットフォーム

Red Hat OpenShift Container Platform EUS リリースの詳細は、 Red Hat OpenShift Container Platform ライフサイクルポリシー を参照してください。

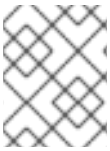
図1.2 サーバーサイドの STF 監視インフラストラクチャー



1.2.1. STF アーキテクチャーの変更

1.5.3 より前の STF のリリースでは、Service Telemetry Operator は Elastic Cloud on Kubernetes (ECK) Operator に Elasticsearch のインスタンスを要求しました。現在、STF は転送モデルを使用するようになり、イベントが Smart Gateway インスタンスからユーザー提供の Elasticsearch インスタンスに転送

されます。



注記

Service Telemetry Operator による Elasticsearch インスタンスの管理は、非推奨になりました。

新しい **ServiceTelemetry** デプロイメントでは、**observabilityStrategy** パラメーターの値は **use_redhat** であり、ECK から Elasticsearch インスタンスを要求しません。STF バージョン 1.5.2 以前を使用し、1.5.3 に更新された **ServiceTelemetry** のデプロイメントでは、**observabilityStrategy** パラメーターが **use_community** に設定されます。これは、以前のアーキテクチャーと一致します。

ユーザーが以前に STF を使用して Elasticsearch インスタンスをデプロイしていた場合、Service Telemetry Operator は **observabilityStrategy** パラメーターが **use_community** になるように **ServiceTelemetry** カスタムリソースオブジェクトを更新し、以前のリリースと同様に機能するようにします。可観測性ストラテジーの詳細は、「Service Telemetry Framework の可観測性ストラテジー」を参照してください。

STF のユーザーには、**use_redhat** 可観測性ストラテジーへの移行が推奨されます。**use_redhat** 可観測性ストラテジーへの移行について、詳細は Red Hat ナレッジベースの Migrating Service Telemetry Framework to fully supported operators を参照してください。

1.3. RED HAT OPENSHIFT 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。
- メトリクスの解決。
- データを保存する期間です。

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 RED HAT OPEN SHIFT CONTAINER PLATFORM の環境を SERVICE TELEMETRY FRAMEWORK 用に準備

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、イベントストレージ、およびネットワークに関する考慮事項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。

2.1. SERVICE TELEMETRY FRAMEWORK の可観測性ストラテジー

Service Telemetry Framework (STF) に、イベントストレージバックエンドやダッシュボードツールは含まれません。STF は、オプションでコミュニティー Operator を使用して Grafana のデータソース設定を作成し、ダッシュボードインターフェイスを提供できます。

Service Telemetry Operator がカスタムリソース要求を作成する代わりに、これらのアプリケーションまたは他の互換性のあるアプリケーションの独自のデプロイメントを使用し、Telemetry ストレージ用の独自の Prometheus 互換システムに配信するためにメトリクス Smart Gateways を収集できます。**observabilityStrategy** を **none** に設定すると、ストレージバックエンドはデプロイされないため、STF は永続ストレージを必要としません。

STF オブジェクトの observabilityStrategy プロパティーを使用して、デプロイされる可観測性コンポーネントのタイプを指定します。

以下の値を使用できます。

値	意味
use_redhat	STF は、Red Hat のサポート対象コンポーネントを要求します。これには、Cluster Observability Operator からの Prometheus と Alertmanager が含まれますが、Elastic Cloud on Kubernetes (ECK) Operator へのリソース要求は含まれません。有効にすると、Grafana Operator (コミュニティーコンポーネント) からもリソースが要求されます。
use_hybrid	Red Hat のサポート対象コンポーネントに加えて、Elasticsearch および Grafana リソースも要求されます (ServiceTelemetry オブジェクトで指定されている場合)。
use_community	Cluster Observability Operator の代わりに、Prometheus Operator のコミュニティーバージョンが使用されます。Elasticsearch および Grafana リソースも要求されます (ServiceTelemetry オブジェクトで指定されている場合)。

値	意味
none	ストレージまたはアラートコンポーネントはデプロイメントされていません。

注記

1.5.3 の時点で新しくデプロイされた STF 環境は、デフォルトで **use_redhat** になります。1.5.3 より前に作成された既存の STF デプロイメントは、デフォルトで **use_community** になります。

既存の STF デプロイメントを **use_redhat** に移行するには、Red Hat ナレッジベース Migrating Service Telemetry Framework to fully supported operators を参照してください。

2.2. 永続ボリューム

Service Telemetry Framework (STF) は、Prometheus がメトリクスを保存できるように、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。

関連情報

- 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テクノロジーの詳細は、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 における Prometheus の永続的ストレージの設定については、「Prometheus に永続ストレージの設定」を参照してください。

2.3. 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。

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については、Red Hat ナレッジベースの記事 Service Telemetry Framework Performance and Scaling を参照してください。

2.4. SERVICE TELEMETRY FRAMEWORK のネットワークに関する考慮事項

Service Telemetry Framework (STF) は、完全に接続されたネットワーク環境または Red Hat OpenShift Container Platform の非接続環境にデプロイできます。STF をネットワークプロキシー環境にデプロイすることはできません。

2.5. RED HAT OPENSHIFT CONTAINER PLATFORM の非接続環境への STF のデプロイ

Service Telemetry Framework (STF) バージョン 1.5.4 以降、Red Hat OpenShift Container Platform の非接続環境に STF をデプロイできるようになりました。

前提条件

- 制限されたネットワークにデプロイされた Red Hat OpenShift Container Platform Extended Update Support (EUS) バージョン 4.12 または 4.14。
- Red Hat OpenShift Container Platform クラスターが必要なイメージにアクセスできるようにするミラーレジストリー。ミラーレジストリーの詳細は、Red Hat OpenShift Container Platform **インストール** ガイドの 非接続インストールのミラーリング を参照してください。
- すべての STF 依存関係は、Red Hat OpenShift Container Platform クラスターミラーレジストリーで利用できます。

ミラーレジストリーへの STF 依存関係の追加

oc-mirror プラグインを使用して STF 依存関係を取得し、それらを Red Hat OpenShift Container Platform クラスターミラーレジストリーに追加できます。**oc-mirror** プラグインのインストールに関する詳細は、Red Hat OpenShift Container Platform **インストール** ガイドの oc-mirror プラグインを使用した非接続インストールのイメージのミラーリング を参照してください。

手順

1. ローカル作業ディレクトリーに **imagesetconfig.yaml** ファイルを作成します。

imagesetconfig.yaml

```
apiVersion: mirror.openshift.io/v1alpha2
kind: ImageSetConfiguration
storageConfig:
  local:
    path: ./
mirror:
  operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.14
    packages:
      - name: service-telemetry-operator
        channels:
          - name: stable-1.5
      - name: openshift-cert-manager-operator
        channels:
          - name: stable-v1
      - name: amq7-interconnect-operator
```

```
    channels:
      - name: 1.10.x
  - name: smart-gateway-operator
    channels:
      - name: stable-1.5
  - name: cluster-observability-operator
    channels:
      - name: development
```

2. (オプション) ミラーレジストリーにアクセスできない場合は、**oc-mirror** で取得したマニフェストとイメージを保存し、それらをミラーレジストリーと Red Hat OpenShift Container Platform クラスターに物理的に転送できます。それ以外の場合は、**oc-mirror** を実行してミラーレジストリーをポイントすることができます。
oc-mirror プラグインは、次のように環境に応じてさまざまな方法で使用できます。

   - ミラー間のミラーリング。
   - ミラーからディスクへのミラーリング。
   - ディスクからミラーへのミラーリング。
   さまざまな **oc-mirror** シナリオの詳細は、Red Hat OpenShift Container Platform **インストール** ガイドの 完全な非接続環境でのイメージセットのミラーリング を参照してください。

3. STF Operator とその依存関係をミラーレジストリーからプッシュし、Red Hat OpenShift Container Platform クラスターのマニフェストを生成します。

```
$ oc-mirror --config imagesetconfig.yaml <mirror_registry_location>
```

   - <mirror_registry_location> を、使用するミラーレジストリーへのファイルパスに置き換えます。

4. 生成されたマニフェストを見つけて、ターゲットの Red Hat OpenShift Container Platform クラスターに適用します。詳細は、Red Hat OpenShift Container Platform **インストール** ガイドの oc-mirror によって生成されたリソースを使用するためのクラスターの設定 を参照してください。

注記

oc-mirror で生成したマニフェストは、**CatalogSource** の **redhat-operators** ではなく **redhat-operator-index** など、完全なインデックス名を持つカタログを生成します。STF サブスクリプションに正しいインデックス名を使用していることを確認してください。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。oc ミラーを使用した Operators のカスタマイズに関する詳細は、Red Hat ナレッジベースのソリューション記事 How to customize the catalog name and tags of Operators mirrored to the mirror registry using the oc mirror plugin. を参照してください。

検証

- カタログソースが適用されていることを確認します。STF Operator とその依存関係を参照する新しいカタログのエントリーを返すことができます。

```
$ oc get catalogsources
```

- 非接続の Red Hat OpenShift Container Platform クラスターに STF をデプロイしたため、外部ネットワークにアクセスできません。

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、次の各 STF コアコンポーネントを管理します。

- 証明書の管理
- AMQ Interconnect
- Smart Gateway
- Prometheus と Alertmanager

Service Telemetry Framework (STF) は、デプロイメントの一部として他のサポート Operator を使用します。STF はほとんどの依存関係を自動的に解決できますが、Prometheus と Alertmanager のインスタンスを提供する Cluster Observability Operator や、証明書の管理に使用する Red Hat OpenShift の cert-manager など、一部の Operator を事前にインストールする必要があります。

前提条件

- Red Hat OpenShift Container Platform Extended Update Support (EUS) リリースバージョン 4.12 または 4.14 が実行されている。
- 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。STF のパフォーマンスについて、詳しくは Red Hat ナレッジベースの記事 Service Telemetry Framework Performance and Scaling を参照してください。
- 完全に接続された環境、または Red Hat OpenShift Container Platform の非接続環境に STF をデプロイしている。STF はネットワークプロキシー環境では使用できません。

重要

STF は、Red Hat OpenShift Container Platform バージョン 4.12 および 4.14 と互換性があります。

関連情報

- Operator の詳細は、**Operator について** を参照してください。
- Operator カタログの詳細は、**Red Hat 提供の Operator カタログ**を参照してください。
- Red Hat の cert-manager Operator の詳細は、**cert-manager Operator for Red Hat OpenShift の概要** を参照してください。
- Cluster Observability Operator の詳細は、**Cluster Observability Operator の概要** を参照してください。
- OpenShift ライフサイクルポリシーと Extended Update Support (EUS) の詳細は、**Red Hat OpenShift Container Platform ライフサイクルポリシー** を参照してください。

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

Service Telemetry Framework (STF) をデプロイして、Red Hat OpenStack Platform (RHOSP) テレメトリーを収集および保存します。

3.1.1. Cluster Observability Operator のデプロイ

ServiceTelemetry オブジェクトで **observabilityStrategy** が **use_redhat** に、**backends.metrics.prometheus.enabled** が **true** に設定されている場合、Service Telemetry Framework (STF) のインスタンスを作成する前に、Cluster Observability Operator (COO) をインストールする必要があります。COO の詳細は、**OpenShift Container Platform ドキュメント** の Cluster Observability Operator の概要 を参照してください。

手順

1. STF がホストされている Red Hat OpenShift Container Platform 環境にログインします。
2. Prometheus にメトリクスを保存するには、**redhat-operators** CatalogSource を使用して Cluster Observability Operator を有効にします。

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

3. Cluster Observability Operator の **ClusterServiceVersion の** ステータスが **Succeeded** であることを確認します。

```
$ oc wait --for jsonpath="{.status.phase}"=Succeeded csv --namespace=openshift-operators
-l operators.coreos.com/cluster-observability-operator.openshift-operators

clusterserviceversion.operators.coreos.com/observability-operator.v0.0.26 condition met
```

3.1.2. cert-manager for Red Hat OpenShift のデプロイ

Service Telemetry Framework (STF) のインスタンスを作成する前に、cert-manager for Red Hat OpenShift (cert-manager) Operator の事前インストールを行う必要があります。cert-manager の詳細は、cert-manager for Red Hat OpenShift の概要 を参照してください。

STF の以前のバージョンでは、唯一使用できる cert-manager チャネルが **tech-preview** で、これは Red Hat OpenShift Container Platform v4.12 まで使用できました。Red Hat OpenShift Container Platform v4.14 以降のバージョンでは、**stable-v1** チャネルから cert-manager インストールをインストールする必要があります。STF を新規インストールする場合は、**stable-v1** チャネルから cert-manager をインストールすることが推奨されます。

警告

Red Hat OpenShift Container Platform クラスターごとにインストールできる cert-manager デプロイメントは 1 つだけです。複数のプロジェクトで cert-manager をサブスクライブすると、デプロイメントが相互に競合します。

手順

1. STF がホストされている Red Hat OpenShift Container Platform 環境にログインします。
2. Red Hat OpenShift Container Platform クラスターに cert-manager がインストールされていないことを確認します。結果が返された場合、それ以上は cert-manager インスタンスをインストールしないでください。

```
$ oc get sub --all-namespaces -o json | jq '.items[] | select(.metadata.name | match("cert-manager")) | .metadata.name'
```

3. cert-manager Operator の namespace を作成します。

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

4. cert-manager Operator の OperatorGroup を作成します。

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

5. redhat-operators CatalogSource を使用して、cert-manager Operator にサブスクライブします。

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

```
  name: openshift-cert-manager-operator
  namespace: cert-manager-operator
  labels:
    operators.coreos.com/openshift-cert-manager-operator.cert-manager-operator: ""
spec:
  channel: stable-v1
  installPlanApproval: Automatic
  name: openshift-cert-manager-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

6. ClusterServiceVersion を検証します。cert-manager Operator が **Succeeded** のフェーズを表示していることを確認します。

```
oc wait --for jsonpath="{.status.phase}"=Succeeded csv --namespace=cert-manager-operator --selector=operators.coreos.com/openshift-cert-manager-operator.cert-manager-operator

clusterserviceversion.operators.coreos.com/cert-manager-operator.v1.12.1 condition met
```

3.1.3. Service Telemetry Operator のデプロイ

Service Telemetry Operator を Red Hat OpenShift Container Platform にデプロイして、Red Hat OpenStack Platform (RHOSP) クラウドプラットフォームの監視用 Service Telemetry Framework (STF) インスタンスを作成するためのサポート Operator とインターフェイスを提供します。

前提条件

- Cluster Observability Operator をインストールして、メトリクスの保存を許可している。詳細は、「Cluster Observability Operator のデプロイ」 を参照してください。
- 証明書管理のために cert-manager for Red Hat OpenShift をインストールしている。詳細は、「cert-manager for Red Hat OpenShift のデプロイ」 を参照してください。

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. 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-1.5
  installPlanApproval: Automatic
  name: service-telemetry-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. Service Telemetry Operator および依存する Operator のフェースが Succeeded になるかどうかを検証します。

```
$ oc wait --for jsonpath="{.status.phase}"=Succeeded csv --namespace=service-telemetry -l operators.coreos.com/service-telemetry-operator.service-telemetry ; oc get csv --namespace service-telemetry

clusterserviceversion.operators.coreos.com/service-telemetry-operator.v1.5.1700688542 condition met

NAME                                         DISPLAY                          VERSION          REPLACES                                    PHASE
amq7-interconnect-operator.v1.10.17          Red Hat Integration - AMQ Interconnect   1.10.17                                     amq7-interconnect-operator.v1.10.4   Succeeded
observability-operator.v0.0.26               Cluster Observability Operator           0.1.0                                       Succeeded
service-telemetry-operator.v1.5.1700688542   Service Telemetry Operator
1.5.1700688542                                 Succeeded
smart-gateway-operator.v5.0.1700688539       Smart Gateway Operator
5.0.1700688539                                 Succeeded
```

3.2. RED HAT OPENSHIFT 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。

前提条件

- STF とサポート Operator をデプロイしている。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。
- Cluster Observability Operator をインストールして、メトリクスの保存を許可している。詳細は、「Cluster Observability Operator のデプロイ」 を参照してください。

- 証明書管理のために cert-manager for Red Hat OpenShift をインストールしている。詳細は、「cert-manager for Red Hat OpenShift のデプロイ」 を参照してください。

手順

1. STF がホストされている Red Hat OpenShift Container Platform 環境にログインします。
2. メトリクス配信用のコアコンポーネントを設定する STF をデプロイするには、**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:
  alerting:
    alertmanager:
      storage:
        persistent:
          pvcStorageRequest: 20G
        strategy: persistent
    enabled: true
  backends:
    metrics:
      prometheus:
        enabled: true
        scrapeInterval: 30s
        storage:
          persistent:
            pvcStorageRequest: 20G
          retention: 24h
          strategy: persistent
  clouds:
  - metrics:
      collectors:
      - bridge:
          ringBufferCount: 15000
          ringBufferSize: 16384
          verbose: false
        collectorType: collectd
        debugEnabled: false
        subscriptionAddress: collectd/cloud1-telemetry
      - bridge:
          ringBufferCount: 15000
          ringBufferSize: 16384
          verbose: false
        collectorType: ceilometer
        debugEnabled: false
        subscriptionAddress: anycast/ceilometer/cloud1-metering.sample
      - bridge:
          ringBufferCount: 15000
          ringBufferSize: 65535
          verbose: false
        collectorType: sensubility
```

```
      debugEnabled: false
      subscriptionAddress: sensubility/cloud1-telemetry
  name: cloud1
 observabilityStrategy: use_redhat
 transports:
  qdr:
   auth: basic
   certificates:
    caCertDuration: 70080h
    endpointCertDuration: 70080h
   enabled: true
   web:
    enabled: false
EOF
```

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。

3. 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=90   changed=0    unreachable=0    failed=0    skipped=26
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。

```
$ oc get pods

NAME                                               READY   STATUS    RESTARTS   AGE
alertmanager-default-0                             3/3     Running   0          123m
default-cloud1-ceil-meter-smartgateway-7dfb95fcb6-bs6jl     3/3     Running   0          122m
default-cloud1-coll-meter-smartgateway-674d88d8fc-858jk     3/3     Running   0          122m
default-cloud1-sens-meter-smartgateway-9b869695d-xcssf      3/3     Running   0          122m
default-interconnect-6cbf65d797-hk7l6              1/1     Running   0          123m
interconnect-operator-7bb99c5ff4-l6xc2             1/1     Running   0          138m
prometheus-default-0                               3/3     Running   0          122m
service-telemetry-operator-7966cf57f-g4tx4         1/1     Running   0          138m
smart-gateway-operator-7d557cb7b7-9ppls            1/1     Running   0          138m
```

3.2.1. ServiceTelemetry オブジェクトのパラメーター

ServiceTelemetry オブジェクトの次の主要な設定パラメーターを設定して、STF デプロイメントを設定できます。

- **alerting**
- **バックエンド**

- **clouds**
- **graphing**
- **highAvailability**
- **transports**

バックエンドパラメーター
backends パラメーターの値を設定すると、メトリクスとイベントにストレージバックエンドを割り当て、**clouds** パラメーターで定義されている Smart Gateway を有効にできます。詳細は、「clouds パラメーター」 を参照してください。

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。Service Telemetry Operator は、Prometheus Operator が Prometheus ワークロードを作成するために監視するカスタムリソースオブジェクトを作成できます。イベントを保存するには、Elasticsearch の外部デプロイメントが必要です。

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

1. **Service Telemetry** オブジェクトを編集します。

```
$ oc edit stf default
```

2. backends.metrics.prometheus.enabled パラメーターの値を **true** に設定します。

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

Prometheus に永続ストレージの設定
backends.metrics.prometheus.storage.persistent で追加のパラメーターを設定すると、Prometheus のストレージクラスやボリュームサイズなどの永続的ストレージオプションを設定できます。

storageClass パラメーターを使用して、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 Open Shift 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。

pvcStorageRequest パラメーターを使用して、ストレージ要求に必要な最小ボリュームサイズを定義します。デフォルトでは、Service Telemetry Operator は **20G** (20 ギガバイト) のボリュームサイズを要求します。

手順

1. 利用可能なストレージクラスをリスト表示します。

```
$ 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
```

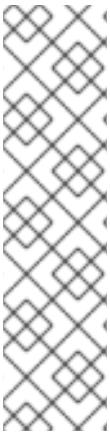
2. **Service Telemetry** オブジェクトを編集します。

```
$ oc edit stf default
```

3. **backends.metrics.prometheus.enabled** パラメーターの値を **true** に**backends.metrics.prometheus.storage.strategy** の値を **persistent** に設定します。

```
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



注記

STF の以前のバージョンでは、コミュニティーがサポートする Elastic Cloud on Kubernetes Operator (ECK) の Elasticsearch オブジェクトが管理されていました。Elasticsearch 管理機能は、STF 1.5.3 で非推奨になりました。引き続き ECK を使用してデプロイおよび管理する既存の Elasticsearch インスタンスに転送することもできますが、Elasticsearch オブジェクトの作成を管理することはできません。STF デプロイメントをアップグレードすると、既存の Elasticsearch オブジェクトとデプロイメントは残りますが、STF によって管理されなくなります。

STF で Elasticsearch を使用する際の詳細は、Red Hat ナレッジベースの記事 Using Service Telemetry Framework with Elasticsearch を参照してください。

ストレージバックエンドとして Elasticsearch にイベントを転送できるようにするには、**ServiceTelemetry** オブジェクトを設定する必要があります。

手順

1. **Service Telemetry** オブジェクトを編集します。

```
$ oc edit stf default
```

2. **backends.events.elasticsearch.enabled** パラメーターの値を **true** に設定し、関連する Elasticsearch インスタンスを使用して **hostUrl** を設定します。

```
apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
metadata:
  name: default
  namespace: service-telemetry
spec:
  [...]
  backends:
    events:
      elasticsearch:
        enabled: true
        forwarding:
          hostUrl: https://external-elastic-http.domain:9200
          tlsServerName: ""
          tlsSecretName: elasticsearch-es-cert
          userSecretName: elasticsearch-es-elastic-user
          useBasicAuth: true
          useTls: true
```

3. **userSecretName** パラメーターで指定されたシークレットを作成して、basic **auth** の認証情報を保存します。

```
$ oc create secret generic elasticsearch-es-elastic-user --from-
literal=elastic='<PASSWORD>'
```

4. CA 証明書を **EXTERNAL-ES-CA.pem** という名前のファイルにコピーし、**tlsSecretName** パラメーターで指定されたシークレットを作成して STF で使用できるようにします。

```
$ cat EXTERNAL-ES-CA.pem
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----

$ oc create secret generic elasticsearch-es-cert --from-file=ca.crt=EXTERNAL-ES-CA.pem
```

clouds パラメーター

clouds パラメーターを設定して、デプロイされる Smart Gateway オブジェクトを定義し、監視対象のクラウド環境に STF のインスタンスに接続するためのインターフェイスを提供します。サポートするバックエンドが利用可能な場合は、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リストを作成して、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設定されます。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、コレクターのリスト、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設定されます。メトリクス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サブスクリプションアドレスは、クラウド、データタイプ、コレクターの組み合わせごとに一意であることを確認してください。

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: default
  namespace: service-telemetry
spec:
  clouds:
    - name: cloud1
      metrics:
        collectors:
          - collectorType: collectd
            subscriptionAddress: collectd/cloud1-telemetry
          - collectorType: ceilometer
            subscriptionAddress: anycast/ceilometer/cloud1-metering.sample
          - collectorType: sensubility
            subscriptionAddress: sensubility/cloud1-telemetry
            debugEnabled: false
      events:
        collectors:
          - collectorType: collectd
            subscriptionAddress: collectd/cloud1-notify
          - collectorType: ceilometer
            subscriptionAddress: anycast/ceilometer/cloud1-event.sample
```

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設定されます。**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** で設定されるオブジェクトのリストを指定し、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。

関連情報

- 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 を削除」 を参照してください。
- 複数のクラウドを設定する方法は、「複数のクラウドの設定」 を参照してください。

alerting パラメーター

alerting パラメーターを設定して、Alertmanager インスタンスとストレージバックエンドを作成します。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 パラメーターを設定して、Grafana インスタンスを作成します。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

警告

STF 高可用性 (HA) モードは非推奨であり、実稼働環境ではサポートされていません。Red Hat OpenShift Container Platform は高可用性プラットフォームであるため、HA モードを有効にすると問題が発生し、STF でのデバッグが複雑になる可能性があります。

highAvailability パラメーターを設定して STF コンポーネントの複数のコピーをインスタンス化し、失敗したか再スケジュールされたコンポーネントの回復時間を短縮します。デフォルトで、**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
transports パラメーターを設定して、STF デプロイメントのメッセージバスを有効にします。現在サポートされているトランスポートは AMQ Interconnect のみです。デフォルトでは、**qdr** トランスポートが有効です。

3.3. STF コンポーネントのユーザーインターフェイスへのアクセス

Red Hat OpenShift Container Platform では、アプリケーションはルートを介して外部ネットワークに公開されます。ルートについての詳細は、Configuring ingress cluster traffic を参照してください。

Service Telemetry Framework (STF) では、Web ベースのインターフェイスを持つサービスごとに HTTPS ルートが公開され、Red Hat OpenShift Container Platform のロールベースのアクセス制御 (RBAC) によって保護されます。

対応するコンポーネント UI にアクセスするには、次の権限が必要です。

```
{"namespace":"service-telemetry", "resource":"grafana", "group":"grafana.integreatly.org",
"verb":"get"}
{"namespace":"service-telemetry", "resource":"prometheus", "group":"monitoring.rhobs", "verb":"get"}
{"namespace":"service-telemetry", "resource":"alertmanager", "group":"monitoring.rhobs",
"verb":"get"}
```

RBAC の詳細は、Using RBAC to define and apply permissions 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. **service-telemetry** プロジェクトで利用可能な Web UI ルートをリスト表示します。

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

4. Web ブラウザーで https://<route_address> に移動し、対応するサービスの Web インターフェイスにアクセスします。

3.4. 代替可観測性ストラテジーの設定

ストレージ、可視化、およびアラートバックエンドのデプロイメントを省略するには、ServiceTelemetry 仕様に **observabilityStrategy: none** を追加します。このモードでは、AMQ Interconnect ルーターと Smart Gateway のみをデプロイし、STF Smart Gateway からメトリクスを収集する外部 Prometheus 互換システムと、転送されたイベントを受信する外部 Elasticsearch を設定する必要があります。

手順

1. **spec** パラメーターに **observabilityStrategy: none** プロパティーを指定して **ServiceTelemetry** オブジェクトを作成します。マニフェストは、すべてのメトリクスコレクタータイプを持つ単一のクラウドから Telemetry を受け取るのに適した STF のデフォルトデプロイメントを示しています。

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

2. コミュニティー Operator が管理する残りのオブジェクトを削除します。

```
$ for o in alertmanagers.monitoring.rhobs/default prometheuses.monitoring.rhobs/default elasticsearch/elasticsearch grafana/default-grafana; do oc delete $o; done
```

3. すべてのワークロードが適切に動作していることを確認するには、Pod および各 Pod のステータスを確認します。

```
$ oc get pods
NAME                                                READY   STATUS    RESTARTS   AGE
default-cloud1-ceil-event-smartgateway-6f8547df6c-p2db5   3/3     Running   0          132m
default-cloud1-ceil-meter-smartgateway-59c845d65b-gzhcs   3/3     Running   0          132m
default-cloud1-coll-event-smartgateway-bf859f8d77-tzb66   3/3     Running   0          132m
default-cloud1-coll-meter-smartgateway-75bbd948b9-d5phm   3/3     Running   0          132m
default-cloud1-sens-meter-smartgateway-7fdbb57b6d-dh2g9   3/3     Running   0          132m
default-interconnect-668d5bbcd6-57b2l                     1/1     Running   0          132m
```

```
interconnect-operator-b8f5bb647-tlp5t                1/1     Running   0          47h
service-telemetry-operator-566b9dd695-wkvjq          1/1     Running   0          156m
smart-gateway-operator-58d77dcf7-6xsq7               1/1     Running   0          47h
```

関連情報

- 追加のクラウドの設定、またはサポート対象のコレクターのセットの変更に関する詳細は、「Smart Gateway の導入」 を参照してください。
- 既存の STF デプロイメントを **use_redhat** に移行するには、Red Hat ナレッジベース Migrating Service Telemetry Framework to fully supported operators を参照してください。

第4章 SERVICE TELEMETRY FRAMEWORK 向けの RED HAT OPENSTACK PLATFORM ディレクターの設定

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 (RHOSP) 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。RHOSP と STF のデフォルトの設定は、単一のクラウドのインストールのために設定されています。

- デフォルトの設定での単一の RHOSP オーバークラウドのデプロイメントについては、「ディレクターを使用した Service Telemetry Framework 用の 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。
- RHOSP のインストールと設定 STF を複数のクラウドで計画するには、「複数のクラウドの設定」を参照してください。
- RHOSP のオーバークラウド導入の一環として、お客様の環境で追加の機能を設定する必要がある場合があります。
    - データコレクターサービスを無効にするには、次を参照してください。「Service Telemetry Framework で使用される Red Hat OpenStack Platform サービスの無効化」

4.1. ディレクターを使用した SERVICE TELEMETRY FRAMEWORK 用の RED HAT OPENSTACK PLATFORM オーバークラウドのデプロイ

ディレクターを使用する Red Hat OpenStack Platform (RHOSP) オーバークラウドのデプロイメントの一環として、データコレクターおよびデータトランスポートを Service Telemetry Framework (STF) に設定する必要があります。

手順

1. AMQ Interconnect のパスワードの取得
2. AMQ Interconnect ルートアドレスの取得
3. STF の基本設定の作成
4. オーバークラウドの STF 接続の設定
5. オーバークラウドのデプロイ
6. クライアント側のインストールの検証

関連情報

- director を使用した OpenStack クラウドのデプロイに関する詳細は、director のインストールと使用方法 を参照してください。
- AMQ Interconnect でデータを収集するには、the amqp1 plug-in を参照してください。

4.1.1. AMQ Interconnect のパスワードの取得

Service Telemetry Framework (STF) 向けに Red Hat OpenStack Platform (RHOSP) オーバークラウドを設定する場合、STF 接続ファイルに AMQ Interconnect のパスワードを指定する必要があります。

ServiceTelemetry 仕様の **transports.qdr.auth** パラメーター値を **none** に設定することで、AMQ Interconnect 接続の Basic 認証を無効にできます。**transports.qdr.auth** パラメーターは 1.5.3 より前の STF バージョンには存在しないため、デフォルトの動作では Basic 認証が無効になります。STF 1.5.3 以降の新規インストールでは、**transports.qdr.auth** のデフォルト値は **basic** ですが、STF 1.5.3 にアップグレードした場合、**transports.qdr.auth** のデフォルト値は **none** になります。

手順

1. STF がホストされている Red Hat OpenShift Container Platform 環境にログインします。
2. **service-telemetry** プロジェクトに変更します。

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

3. AMQ Interconnect のパスワードを取得します。

```
$ oc get secret default-interconnect-users -o json | jq -r .data.guest | base64 -d
```

4.1.2. AMQ Interconnect ルートアドレスの取得

Service Telemetry Framework (STF) 向けに Red Hat OpenStack Platform (RHOSP) 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. STF がホストされている Red Hat OpenShift Container Platform 環境にログインします。
2. **service-telemetry** プロジェクトに変更します。

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

3. AMQ インターコネクトの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"
default-interconnect-5671-service-telemetry.apps.infra.watch
```

4.1.3. STF の基本設定の作成

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. アンダークラウドホストに **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 **enable-stf.yaml** という名前の設定ファイルを作成します。

重要

PipelinePublishers を空のリストに設定すると、Gnocchi や Panko などの RHOSP Telemetry コンポーネントにメトリクスデータは渡されません。追加のパイプラインにデータを送信する必要がある場合、**ExtraConfig** で指定する **30** 秒の Ceilometer ポーリング間隔により、RHOSP テレメトリーコンポーネントが過負荷になる可能性があります。間隔を **300** などの大きな値に増やす必要があります。その場合、STF でのテレメトリー解像度が低くなります。

enable-stf.yaml

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

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

  # enable Ceilometer metrics
  CeilometerQdrPublishMetrics: true

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

  # enable collection of containerized service metrics
  CollectdEnableLibpodstats: true

  # set collectd overrides for higher telemetry resolution and extra plugins
  # to load
  CollectdConnectionType: amqp1
  CollectdAmqpInterval: 30
  CollectdDefaultPollingInterval: 30
  CollectdExtraPlugins:
  - vmem

  # set standard prefixes for where metrics 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
    - memory.usage

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

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

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

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

# set memcached collectd plugin to report its metrics by hostname
# rather than host IP, ensuring metrics in the dashboard remain uniform
collectd::plugin::memcached::instances:
  local:
    host: "%{hiera('fqdn_canonical')}"
    port: 11211
```

4.1.4. 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。メトリクスの収集と STF への保存を有効にし、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ト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。複数のクラウドのデプロイメントの設定については、「複数のクラウドの設定」 を参照してください。

前提条件

- AMQ Interconnect のパスワードを取得する。詳細は、「AMQ Interconnect のパスワードの取得」 を参照してください。
- AMQ Interconnect のルートアドレスを取得する。詳細は、「AMQ Interconnect ルートアドレスの取得」 を参照してください。

手順

1. アンダークラウドホストに **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。このファイルの Sensubility、Ceilometer、および collectd のトピックアドレスを、STF のデフォルトに一致するように設定します。トピックおよびクラウド設定のカスタマイズに関する詳細は、「複数のクラウドの設定」 を参照してください。

stf-connectors.yaml

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

parameter_defaults:
    ExtraConfig:
        qdr::router_id: "%{::hostname}.cloud1"
```

```
MetricsQdrConnectors:
  - host: default-interconnect-5671-service-telemetry.apps.infra.watch
    port: 443
    role: edge
    verifyHostname: false
    sslProfile: sslProfile
    saslUsername: guest@default-interconnect
    saslPassword: pass:<password_from_stf>

MetricsQdrSSLProfiles:
  - name: sslProfile

CeilometerQdrMetricsConfig:
  driver: amqp
  topic: cloud1-metering

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

CollectdSensubilityResultsChannel: sensubility/cloud1-telemetry
```

- **qdr::router_id** 設定は、ホストの完全修飾ドメイン名 (FQDN) を使用するデフォルト値をオーバーライドします。場合によっては、FQDN を使用することでルーター ID の長さが 61 文字を超え、QDR 接続が失敗することがあります。FQDN 値がそれより短いデプロイメントの場合、これは必要ありません。
- 複数のクラウドデプロイメント用に **collectd-write-qdr.yaml** 環境ファイルを含めないため、**resource_registry** 設定は collectd サービスを直接ロードします。
- **MetricsQdrConnectors** の **host** サブパラメーターを、「AMQ Interconnect ルートアドレスの取得」 で取得した値に置き換えます。
- **MetricsQdrConnectors** の **saslPassword** サブパラメーターの **<password_from_stf>** 部分を、「AMQ Interconnect のパスワードの取得」 で取得した値に置き換えます。
- **CeilometerQdrMetricsConfig.topic** の **topic** 値を設定して、Ceilometer メトリクスのトピックを定義します。値は、**cloud1-metering** などのクラウドの一意のトピック識別子です。
- **CollectdAmqpInstances** サブパラメーターを設定して、collectd メトリクスのトピックを定義します。セクション名は、**cloud1-telemetry** などのクラウドの一意のトピック識別子です。
- **CollectdSensubilityResultsChannel** を設定して、collectd-sensubility イベントのトピックを定義します。値は、**sensubility/cloud1-telemetry** などのクラウドの一意のトピック識別子です。

注記

collectd および Ceilometer のトピックを定義すると、指定した値は、Smart Gateway クライアントがメッセージをリッスンするために使用する完全なトピックに置き換えられます。

Ceilometer トピック値はトピックアドレス **anycast/ceilometer/<TOPIC>.sample** に置き換えられ、collectd トピック値はトピックアドレス **collectd/<TOPIC>** に置き換えられます。sensubility の値は完全なトピックパスであり、トピック値からトピックアドレスへの転置はありません。

完全なトピックアドレスを参照する **ServiceTelemetry** オブジェクトのクラウド設定の例については、「clouds パラメーター」 を参照してください。

4.1.5. 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. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

3. データコレクションと AMQ Interconnect 環境ファイルを他の環境ファイルとともにスタックに追加し、オーバークラウドをデプロイします。

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

- Ceilometer Telemetry が STF に送信されるようにするには、**ceilometer-write-qdr.yaml** ファイルを含めます。
- **qdr-edge-only.yaml** ファイルを含めて、メッセージバスが有効になり、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します。
- デフォルトが正しく設定されていることを確認するには、**enable-stf.yaml** 環境ファイルを組み込みます。
- STF への接続を定義するための **stf-connectors.yaml** 環境ファイルを含めます。

4.1.6. 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 (RHOSP) 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

一部のテレメトリーデータは、RHOSP にアクティブなワークロードがある場合にのみ利用可能です。

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** とコレクションエージェントコンテナーがノード上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 collectd
ceilometer_agent_notification ceilometer_agent_central
running
running
running
running
```

注記

Compute ノードで次のコマンドを使用します。

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

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    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
```

接続は 4 つあります。

- STF へのアウトバウンド接続
- ceilometer からのインバウンド接続
- collectd 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、リンクをリスト表示して 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.lSY6Mcicol4J2Kp  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     0
```

6. RHOSP ノードから STF へのアドレスをリスト表示するには、Red Hat OpenShift Container Platform に接続して AMQ Interconnect Pod 名を取得し、接続をリスト表示します。利用可能な AMQ Interconnect Pod をリスト表示します。

```
$ 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、既知の接続をリスト表示します。この例では、RHOSP ノードから接続 **id** 22、23、および 24 の 3 つの **edge** 接続があります。

```
$ oc exec -it deploy/default-interconnect -- 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 deploy/default-interconnect -- 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. SERVICE TELEMETRY FRAMEWORK で使用される RED HAT OPENSTACK PLATFORM サービスの無効化

Red Hat OpenStack Platform (RHOSP) をデプロイして Service Telemetry Framework (STF) に接続するときに使用されるサービスを無効にします。サービスの無効化の一部として、ログまたは生成された設定ファイルが削除されることはありません。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

3. **disable-stf.yaml** 環境ファイルを作成します。

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

4. RHOSP director デプロイメントから以下のファイルを削除します。
   - **ceilometer-write-qdr.yaml**
   - **qdr-edge-only.yaml**
   - **enable-stf.yaml**
   - **stf-connectors.yaml**
5. RHOSP オーバークラウドを更新します。環境ファイルのリストの早い段階で **disable-stf.yaml** ファイルを使用していることを確認してください。リストの早い段階で **disable-stf.yaml** を 追加することにより、他の環境ファイルは、サービスを無効にする設定をオーバーライドできます。

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

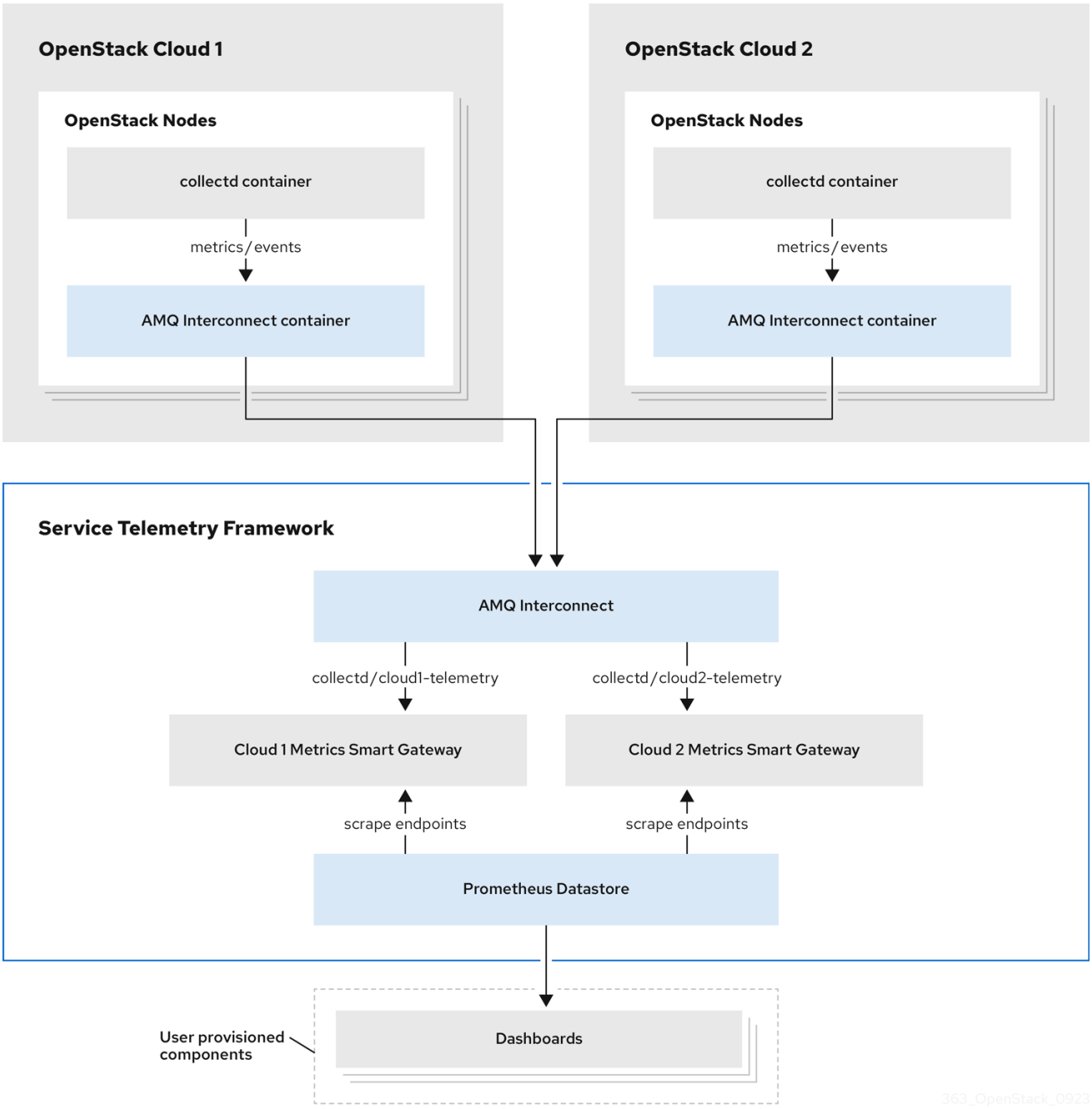
4.3. 複数のクラウドの設定

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 (RHOSP) 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、クラウドはすべて独自で一意の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。

警告

各クラウドは、必ず固有のクラウドドメイン設定でデプロイしてください。クラウドデプロイメント用のドメイン設定について、詳しくは 「独自のクラウドドメインの設定」 を参照してください。

図4.1 RHOSP の 2 つのクラウドが STF に接続



複数のクラウドのシナリオで RHOSP オーバークラウドを設定するには、次のタスクを実行します。

1. 各クラウドで使用する AMQP アドレスの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. 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、対応するアドレス接頭辞をリッスンします。詳細は、「Smart Gateway の導入」 を参照してください。
3. 各クラウドに固有のドメイン名を設定します。詳細は、「独自のクラウドドメインの設定」 を参照してください。
4. STF の基本設定を作成します。詳細は、「STF の基本設定の作成」 を参照してください。

5. 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、「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。

4.3.1. AMQP アドレス接頭辞の計画

デフォルトでは、Red Hat OpenStack Platform (RHOSP) ノードは、collectd と Ceilometer という 2 つのデータコレクターを通じてデータを取得します。collectd-sensubility プラグインでは、固有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、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。アドレスの 2 番目の部分に、クラウド識別子の接頭辞を追加します。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**
- **sensubility/cloud1-telemetry**
- **anycast/ceilometer/cloud1-metering.sample**
- **anycast/ceilometer/cloud1-event.sample**
- **collectd/cloud2-telemetry**
- **collectd/cloud2-notify**
- **sensubility/cloud2-telemetry**
- **anycast/ceilometer/cloud2-metering.sample**
- **anycast/ceilometer/cloud2-event.sample**
- **collectd/us-east-1-telemetry**
- **collectd/us-west-3-telemetry**

4.3.2. Smart Gateway の導入

各クラウドのデータ収集タイプごとに、collectd メトリクス用、collectd イベント用、Ceilometer メトリクス用、Ceilometer イベント用、collectd-sensubility メトリクス用の Smart Gateway を導入する必要があります。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。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 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。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

前提条件

- クラウドの命名方法を決めています。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。
- clouds オブジェクトのリストを作成しました。**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
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
```

4.3.3. デフォルトの Smart Gateway を削除

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** リスト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リストを定義します。

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リストで **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. 修正内容を保存します。
4. オペレーターが Smart Gateway を削除したことを確認します。Operator が変更を調整する間、数分かかることがあります。

```
$ oc get smartgateways
```

4.3.4. 独自のクラウドドメインの設定

さまざまな Red Hat OpenStack Platform (RHOSP) クラウドから Service Telemetry Framework (STF) への Telemetry が一意に識別され、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。

警告

既存のデプロイメントではホスト名またはドメイン名を変更しないようにしてください。ホストとドメイン名の設定は、新しいクラウドデプロイメントでのみサポートされます。

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。

   hostnames.yaml

   ```
   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。

関連情報

- 「複数クラウドの Red Hat OpenStack Platform 環境ファイルの作成」
- **オーバークラウドパラメーター** ガイドの コアオーバークラウドパラメーター

4.3.5. 複数クラウドの Red Hat OpenStack Platform 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、**CeilometerQdrMetricsConfig** と **CollectdAmqpInstances** の値を AMQP アドレス接頭辞スキームと一致するように調整します。

注記

コンテナーのヘルスおよび 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。詳細は、「Red Hat OpenStack Platform API のステータスおよびコンテナー化されたサービスの健全性」 を参照してください。

前提条件

- clouds オブジェクトのリストを作成しました。clouds パラメーターのコンテンツを作成する方法については、clouds 設定パラメーター を参照してください。
- AMQ Interconnect のルートアドレスを取得しました。詳細は、「AMQ Interconnect ルートアドレスの取得」 を参照してください。
- これで STF の基本設定ができました。詳細は、「STF の基本設定の作成」 を参照してください。
- 固有のドメイン名環境ファイルを作成しました。詳細は、「独自のクラウドドメインの設定」を参照してください。

手順

1. アンダークラウドホストに **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。**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 アドレスと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

parameter_defaults:
    ExtraConfig:
      qdr::router_id: %{::hostname}.cloud1

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

    MetricsQdrSSLProfiles:
      - name: sslProfile

    CeilometerQdrMetricsConfig:
      driver: amqp
      topic: cloud1-metering

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

    CollectdSensubilityResultsChannel: sensubility/cloud1-telemetry
```

- **qdr::router_id** 設定は、ホストの完全修飾ドメイン名 (FQDN) を使用するデフォルト値をオーバーライドします。場合によっては、FQDN を使用することでルーター ID の長さが 61 文字を超え、QDR 接続が失敗することがあります。FQDN 値がそれより短いデプロイメントの場合、これは必要ありません。

- 複数のクラウドデプロイメント用に **collectd-write-qdr.yaml** 環境ファイルを含めないため、**resource_registry** 設定は collectd サービスを直接ロードします。

- **host** パラメーターを、「AMQ Interconnect ルートアドレスの取得」 で取得した値に置き換えます。

- **MetricsQdrConnectors** の **host** サブパラメーターを、「AMQ Interconnect ルートアドレスの取得」 で取得した値に置き換えます。

- **CeilometerQdrMetricsConfig.topic** の **topic** 値を設定して、Ceilometer メトリクスのトピックを定義します。値は、**cloud1-metering** などのクラウドの一意のトピック識別子です。

- **CollectdAmqpInstances** サブパラメーターを設定して、collectd メトリクスのトピックを

- **CollectdAmqpInstances** サブパラメーターを設定して、collectd メトリクスのトピックを定義します。セクション名は、**cloud1-telemetry** などのクラウドの一意のトピック識別子です。

- **CollectdSensubilityResultsChannel** を設定して、collectd-sensubility イベントのトピックを定義します。値は、**sensubility/cloud1-telemetry** などのクラウドの一意のトピック識別子です。

注記

collectd および Ceilometer のトピックを定義すると、指定した値は、Smart Gateway クライアントがメッセージをリッスンするために使用する完全なトピックに置き換えられます。

Ceilometer トピック値はトピックアドレス **anycast/ceilometer/<TOPIC>.sample** に置き換えられ、collectd トピック値はトピックアドレス **collectd/<TOPIC>** に置き換えられます。sensubility の値は完全なトピックパスであり、トピック値からトピックアドレスへの転置はありません。

完全なトピックアドレスを参照する **ServiceTelemetry** オブジェクトのクラウド設定の例については、「clouds パラメーター」 を参照してください。

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、**CeilometerQdrMetricsConfig.topic** フィールドの値を **cloud1-metering** に設定します。

5. アンダークラウドホストに **stack** ユーザーとしてログインします。

6. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source stackrc
```

7. 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 deployment** 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)$ openstack overcloud deploy --templates \
-e [your environment files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/metrics/ceilometer-write-qdr.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/metrics/qdr-edge-only.yaml \
```

```
-e /home/stack/hostnames.yaml \
-e /home/stack/enable-stf.yaml \
-e /home/stack/stf-connectors.yaml
```

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

関連情報

- 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。

4.3.6. 複数のクラウドからメトリクスデータを照会

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 用の RED HAT OPENSTACK PLATFORM DIRECTOR を設定する

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 (RHOSP) 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。RHOSP と STF のデフォルトの設定は、単一のクラウドのインストールのために設定されています。

- デフォルト設定で director Operator を使用したシングル RHOSP オーバークラウドデプロイメントについて、詳しくは 「director Operator を使用して Service Telemetry Framework 用の Red Hat OpenStack Platform オーバークラウドをデプロイする」 を参照してください。

5.1. DIRECTOR OPERATOR を使用して SERVICE TELEMETRY FRAMEWORK 用の RED HAT OPENSTACK PLATFORM オーバークラウドをデプロイする

director Operator を使用して Red Hat OpenStack Platform (RHOSP) オーバークラウドデプロイメントをデプロイする場合は、Service Telemetry Framework (STF) のデータコレクターとデータトランスポートを設定する必要があります。

前提条件

- RHOSP director Operator を使用した RHOSP のデプロイと管理に精通している。

手順

1. AMQ Interconnect ルートアドレスの取得
2. STF 用の director Operator の基本設定を作成する
3. オーバークラウドの STF 接続の設定
4. director Operator 用のオーバークラウドをデプロイする

関連情報

- director Operator を使用した OpenStack クラウドのデプロイに関する詳細は、OpenShift Container Platform 向けの RHOSP director Operator を参照してください。
- AMQ Interconnect でデータを収集するには、the amqp1 plug-in を参照してください。

5.1.1. AMQ Interconnect ルートアドレスの取得

Service Telemetry Framework (STF) 向けに Red Hat OpenStack Platform (RHOSP) 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. STF がホストされている Red Hat OpenShift Container Platform 環境にログインします。

2. **service-telemetry** プロジェクトに変更します。

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

3. AMQ インターコネクトの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"
default-interconnect-5671-service-telemetry.apps.infra.watch
```

5.1.2. STF 用の director Operator の基本設定を作成する

heat-env-config-deploy ConfigMap を編集して、ベースとなる Service Telemetry Framework (STF) 設定をオーバークラウドノードに追加します。

手順

1. RHOSP director Operator がデプロイされている Red Hat OpenShift Container Platform 環境にログインし、RHOSP デプロイメントをホストするプロジェクトに変更します。

```
$ oc project openstack
```

2. **heat-env-config-deploy ConfigMap** CR を編集するために開きます。

```
$ oc edit heat-env-config-deploy
```

3. **enable-stf.yaml** 設定を **heat-env-config-deploy** ConfigMap に追加し、編集内容を保存してファイルを閉じます。

enable-stf.yaml

```
apiVersion: v1
data:
  [...]
  enable-stf.yaml: |
    parameter_defaults:
      # only send to STF, not other publishers
      PipelinePublishers: []

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

      # enable Ceilometer metrics and events
      CeilometerQdrPublishMetrics: true

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

      # enable collection of containerized service metrics
      CollectdEnableLibpodstats: true
```

```
  # set collectd overrides for higher telemetry resolution and extra plugins
  # to load
  CollectdConnectionType: amqp1
  CollectdAmqpInterval: 30
  CollectdDefaultPollingInterval: 30
  CollectdExtraPlugins:
  - vmem

  # set standard prefixes for where metrics 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
    - memory.usage

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

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

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

    # provide the human-friendly name of the virtual instance
    collectd::plugin:ConfigMap :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
```

関連情報

- **enable-stf.yaml** 環境ファイルの設定について、詳しくは 「STF の基本設定の作成」 を参照してください。
- Red Hat OpenStack Platform director Operator デプロイメントへの heat テンプレートの追加について、詳しくは director Operator を使用して heat テンプレートと環境ファイルを追加する を参照してください。

5.1.3. オーバークラウドの director Operator の STF 接続を設定する

heat-env-config-deploy ConfigMap を編集して、Red Hat OpenStack Platform (RHOSP) から Service Telemetry Framework への接続を作成します。

手順

1. RHOSP director Operator がデプロイされている Red Hat OpenShift Container Platform 環境にログインし、RHOSP デプロイメントをホストするプロジェクトに変更します。

```
$ oc project openstack
```

2. **heat-env-config-deploy** ConfigMap を開いて編集します。

```
$ oc edit configmap heat-env-config-deploy
```

3. 環境に適切な **stf-connectors.yaml** 設定を **heat-env-config-deploy** ConfigMap に追加し、編集内容を保存してファイルを閉じます。

stf-connectors.yaml

```
apiVersion: v1
data:
  [...]
  stf-connectors.yaml: |
    resource_registry:
      OS::TripleO::Services::Collectd: /usr/share/openstack-tripleo-heat-
templates/deployment/metrics/collectd-container-puppet.yaml

    parameter_defaults:
      MetricsQdrConnectors:
        - host: default-interconnect-5671-service-telemetry.apps.ostest.test.metalkube.org
          port: 443
          role: edge
          verifyHostname: false
          sslProfile: sslProfile
          saslUsername: guest@default-interconnect
          saslPassword: pass:<password_from_stf>

      MetricsQdrSSLProfiles:
        - name: sslProfile

      CeilometerQdrMetricsConfig:
        driver: amqp
        topic: cloud1-metering

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

      CollectdSensubilityResultsChannel: sensubility/cloud1-telemetry
```

関連情報

- **stf-connectors.yaml** 環境ファイルの詳細は、「オーバークラウドの STF 接続の設定」 を参照してください。
- RHOSP director Operator デプロイメントへの heat テンプレートの追加について、詳しくは director Operator を使用して heat テンプレートと環境ファイルを追加する を参照してください。

5.1.4. director Operator のオーバークラウドをデプロイする

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. RHOSP director Operator がデプロイされている Red Hat OpenShift Container Platform 環境にログインし、RHOSP デプロイメントをホストするプロジェクトに変更します。

```
$ oc project openstack
```

2. **OpenStackConfigGenerator** カスタムリソースを、編集するために開きます。

```
$ oc edit OpenStackConfigGenerator
```

3. **heatEnvs** パラメーターの値として、**metrics/ceilometer-write-qdr.yaml** および **metrics/qdr-edge-only.yaml** 環境ファイルを追加します。編集内容を保存し、**OpenStackConfigGenerator** カスタムリソースを閉じます。

注記

すでに director Operator を使用して Red Hat OpenStack Platform 環境をデプロイしている場合は、**OpenStackConfigVersion** を再生成するために、既存の **OpenStackConfigGenerator** を削除し、完全な設定で新しいオブジェクトを作成する必要があります。

OpenStackConfigGenerator

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
metadata:
  name: default
  namespace: openstack
spec:
  heatEnvConfigMap: heat-env-config-deploy
  heatEnvs:
  - <existing_environment_file_references>
  - metrics/ceilometer-write-qdr.yaml
  - metrics/qdr-edge-only.yaml
```

4. すでに director Operator を使用して Red Hat OpenStack Platform 環境をデプロイし、新しい **OpenStackConfigVersion** を生成している場合は、デプロイメントの **OpenStackDeploy** オブジェクトを編集し、**spec.configVersion** の値を新しい **OpenStackConfigVersion** に設定してオーバークラウドデプロイメントを更新します。

関連情報

- 最新の **OpenStackConfigVersion** の取得について、詳しくは 最新の OpenStackConfigVersion を取得する を参照してください。
- director Operator を使用したオーバークラウド設定の適用について、詳しくは director Operator を使用してオーバークラウド設定を適用する を参照してください。

第6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
- 代替可観測性ストラテジーの設定
- OpenStack サービスのリソース使用状況の監視
- コンテナーの健全性と API の状態を監視

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

サードパーティーアプリケーション Grafana を使用して、データコレクター collectd および Ceilometer が個々のホストノードごとに収集するシステムレベルのメトリクスを視覚化します。

データコレクターの設定の詳細は、次を参照してください。「ディレクターを使用した Service Telemetry Framework 用の Red Hat OpenStack Platform オーバークラウドのデプロイ」

ダッシュボードを使用してクラウドをモニターできます。

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、1 つのノードのメトリクスを一度に表示します。ダッシュボードの左上からノードを選択します。

クラウドビューダッシュボード

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のデータを提供するには、API ヘルスモニタリングとサービスモニタリングを有効にする必要があります。API ヘルスモニタリングは、STF ベース設定でデフォルトで有効にされます。詳細は、「STF の基本設定の作成」 を参照してください。

- API ヘルス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ー化されたサービスの健全性」 を参照してください。
- RHOSP サービス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。

仮想マシンビューダッシュボード

仮想マシンビューダッシュボードを使用してパネルを表示し、仮想マシンインフラストラクチャーの使用状況をモニターします。ダッシュボードの左上隅からクラウドとプロジェクトを選択します。このダッシュボードでイベントのアノテーションを有効にする場合は、イベントストレージを有効にする必要があります。詳細は、「Red Hat OpenShift Container Platform での ServiceTelemetry オブジェクトの作成」 を参照してください。

Memcached ビューのダッシュボード

memcached ビューダッシュボードを使用してパネルを表示し、接続、可用性、システムメトリクス、およびキャッシュパフォーマンスをモニターします。ダッシュボードの左上隅からクラウドを選択します。

6.1.1. 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) デプロイメントには含まれていないため、コミュニティーオペレーターの CatalogSource 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. STF がホストされている Red Hat OpenShift Container Platform 環境にログインします。
2. Community-operators CatalogSource を使用して、Grafana Operator を購読します。

警告

コミュニティーオペレーターは、Red Hat による精査または検証を受けていないオペレーターです。コミュニティーオペレーターは安定性が不明であるため、注意して使用する必要があります。Red Hat は、Community Operators のサポートを提供しません。

Red Hat のサードパーティーソフトウェアサポートポリシーの詳細については、こちらをご覧ください。

```
$ oc apply -f - <<EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  labels:
    operators.coreos.com/grafana-operator.openshift-operators: ""
  name: grafana-operator
  namespace: openshift-operators
spec:
  channel: v5
  installPlanApproval: Automatic
  name: grafana-operator
  source: community-operators
  sourceNamespace: openshift-marketplace
EOF
```

3. 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 wait --for jsonpath="{.status.phase}"=Succeeded csv --namespace openshift-operators -l operators.coreos.com/grafana-operator.openshift-operators

clusterserviceversion.operators.coreos.com/grafana-operator.v5.6.0 condition met
```

4. 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。オプションで、**graphing.grafana.baseImage** の値を、デプロイされる Grafana ワークロードコンテナーイメージに設定します。

```
$ oc edit stf default

apiVersion: infra.watch/v1beta1
kind: ServiceTelemetry
...
spec:
  ...
  graphing:
    enabled: true
    grafana:
      ingressEnabled: true
      baseImage: 'registry.redhat.io/rhel8/grafana:9'
```

5. Grafana インスタンスがデプロイされたことを確認します。

```
$ oc wait --for jsonpath="{.status.phase}"=Running pod -l app=default-grafana --
timeout=600s

pod/default-grafana-deployment-669968df64-wz5s2 condition met
```

6. Grafana のデータ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 grafanadatasources.grafana.integreatly.org

NAME                      NO MATCHING INSTANCES   LAST RESYNC   AGE
default-ds-stf-prometheus                         2m35s         2m56s
```

7. Grafana のルートが存在することを確認します。

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

NAME                    HOST/PORT                                                PATH   SERVICES
PORT   TERMINATION   WILDCARD
default-grafana-route   default-grafana-route-service-telemetry.apps.infra.watch
default-grafana-service   web     reencrypt     None
```

6.1.2. ダッシュボードの有効化

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。Service Telemetry Operator は、Grafana インスタンスにダッシュボードをロードする **GrafanaDashboard** オブジェクトを作成する一連のデフォルトダッシュボードを有効にすることができます。

次のダッシュボードを Grafana にロードするには、**graphing.grafana.dashboards.enabled** の値を **true** に設定します。

- インフラストラクチャーダッシュボード
- クラウドビューダッシュボード

- 仮想マシンビューダッシュボード
- Memcached ビューのダッシュボード

GrafanaDashboard オブジェクトを使用して、追加のダッシュボードを作成し、Grafana にロードできます。Grafana Operator を使用したダッシュボードの管理に関する詳細は、**Grafana Operator プロジェクトドキュメント** の Dashboards を参照してください。

前提条件

- **ServiceTelemetry** オブジェクトでグラフ作成を有効にしました。グラフ作成の詳細は、「ダッシュボードをホストするための Grafana の設定」 を参照してください。

手順

1. 管理対象のダッシュボードを有効にするには、**ServiceTelemetry** オブジェクトを作成または変更します。**graphing.grafana.dashboards.enabled** を **true** に設定します。

```
$ oc edit stf default

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

2. Grafana ダッシュボードが作成されたことを確認します。Service Telemetry Operator がダッシュボードを作成するプロセスには、時間がかかる場合があります。

```
$ oc get grafanadashboards.grafana.integreatly.org

NAME                          NO MATCHING INSTANCES   LAST RESYNC   AGE
memcached-dashboard-1                                 38s           38s
rhos-cloud-dashboard-1                                39s           39s
rhos-dashboard-1                                      39s           39s
virtual-machine-dashboard-1                           37s           37s
```

3. Grafana のルートアドレスを取得します。

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

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

4. Web ブラウザーで、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。
5. OpenShift 認証情報を使用してログインします。ログインの詳細は、「STF コンポーネントのユーザーインターフェイスへのアクセス」 を参照してください。

6. ダッシュボードを表示するには、**Dashboards** および **Browse** をクリックします。管理対象のダッシュボードは、**service-telemetry** フォルダーで利用できます。

6.1.3. 外部ダッシュボードシステムの接続

サードパーティーの視覚化ツールを設定して、メトリクスを取得するために STF Prometheus に接続することができます。アクセスは OAuth トークンによって制御され、必要な権限 (のみ) を持つ ServiceAccount がすでに作成されています。外部システムが使用するように、このアカウントに対して新しい OAuth トークンを生成できます。

認証トークンを使用するには、RFC6750 で説明されているように、HTTP Bearer Token Authorization ヘッダーを提供するようにサードパーティーツールを設定する必要があります。このヘッダーの設定方法については、サードパーティーツールのドキュメントを参照してください。例: **Grafana ドキュメント** の Configure Prometheus - Custom HTTP Headers

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-prometheus-reader サービスアカウントの新しいトークンシークレットを作成します。

```
$ oc create -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: my-prometheus-reader-token
  namespace: service-telemetry
  annotations:
    kubernetes.io/service-account.name: stf-prometheus-reader
type: kubernetes.io/service-account-token
EOF
```

4. シークレットからトークンを取得します。

```
$ TOKEN=$(oc get secret my-prometheus-reader-token -o template='{{.data.token}}' |
base64 -d)
```

5. Prometheus のホスト名を取得します。

```
$ PROM_HOST=$(oc get route default-prometheus-proxy -ogo-template='{{ .spec.host }}')
```

6. アクセストークンをテストします。

```
$ curl -k -H "Authorization: Bearer ${TOKEN}" https://${PROM_HOST}/api/v1/query?
query=up

{"status":"success",[...]
```

7. 上記の PROM_HOST および TOKEN 値を使用して、サードパーティーツールを設定します。

```
$ echo $PROM_HOST
$ echo $TOKEN
```

8. トークンは、シークレットが存在する限り有効です。シークレットを削除すると、トークンを取り消すことができます。

```
$ oc delete secret my-prometheus-reader-token
secret "my-prometheus-reader-token" deleted
```

関連情報

サービスアカウントトークンシークレットの詳細は、**OpenShift Container Platform ドキュメント** の サービスアカウントトークンシークレットの作成 を参照してください。

6.2. 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。

関連情報

- 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 のデータ保存に関する推奨事項や、ストレージ容量の見積もりについては、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。

6.2.1. 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. Service Telemetry 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: default
  namespace: service-telemetry
spec:
  ...
  backends:
    metrics:
      prometheus:
        enabled: true
        storage:
          strategy: persistent
          retention: 7d
    ...
```

5. 変更内容を保存し、オブジェクトを閉じます。
6. 新しい設定で prometheus が再起動されるまで待ちます。

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

7. Pod で使用されているコマンドライン引数をチェックして、新しい保持設定を確認します。

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

関連情報

- 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。

6.3. SERVICE TELEMETRY FRAMEWORK でのアラート

Prometheus ではアラートルールを作成し、Alertmanager では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、以下のタスクを行います。

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。アラートルートを作成するには、2 つの方法があります。
   - Alertmanager での標準的なアラートルートの作成。

- Alertmanager のテンプレート化によるアラートルートの作成。

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。

6.3.1. 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.rhobs/v1
kind: PrometheusRule
metadata:
  creationTimestamp: null
  labels:
    prometheus: default
    role: alert-rules
  name: prometheus-alarm-rules
  namespace: service-telemetry
spec:
  groups:
    - name: ./openstack.rules
      rules:
        - alert: Collectd metrics receive rate is zero
          expr: rate(sg_total_collectd_msg_received_count[1m]) == 0
EOF
```

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。

4. Operator がルールを Prometheus に読み込んだことを確認するには、Basic 認証で default-prometheus-proxy ルートに対して **curl** コマンドを実行します。

```
$ curl -k -H "Authorization: Bearer $(oc create token stf-prometheus-reader)" https://$(oc get route default-prometheus-proxy -ogo-template='{{ .spec.host }}')/api/v1/rules

{"status":"success","data":{"groups":
[{"name":"./openstack.rules","file":"/etc/prometheus/rules/prometheus-default-rulefiles-
```

```
0/service-telemetry-prometheus-alarm-rules.yaml","rules":
[{"state":"inactive","name":"Collectd metrics receive count is
zero","query":"rate(sg_total_collectd_msg_received_count[1m]) == 0","duration":0,"labels":
{},"annotations":{},"alerts":
[],"health":"ok","evaluationTime":0.00034627,"lastEvaluation":"2021-12-
07T17:23:22.160448028Z","type":"alerting"}],"interval":30,"evaluationTime":0.000353787,"last
Evaluation":"2021-12-07T17:23:22.160444017Z"}]}}
```


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。

6.3.2. 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

1. **oc edit** コマンドを使用します。

```
$ oc edit prometheusrules.monitoring.rhobs prometheus-alarm-rules
```

2. **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。

関連情報

- アラートルールの設定方法は、https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/ を参照してください。
- Prometheus Rules オブジェク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。

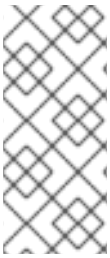
6.3.3. Alertmanager での標準的なアラートルートの作成

Alertmanager を使用して、電子メール、IRC、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配信します。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 に追加する必要があります。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。



注記

alertmanagerConfigManifest に、送信されるアラートのタイトルとテキストを設定するカスタムテンプレートが含まれている場合は、Base64 エンコードされた設定を使用して、**alertmanagerConfigManifest** のコンテンツをデプロイする必要があります。詳細は、「Alertmanager のテンプレート化によるアラートルートの作成」 を参照してください。

手順

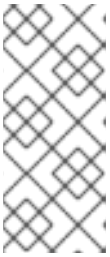
1. Red Hat OpenShift Container Platform にログインします。
2. **service-telemetry** namespace に切り替えます。

```
$ 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 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たことを確認します。

```
$ 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'
```

6. Prometheus Pod から **alertmanager-proxy** サービスに対して **wget** コマンドを実行して、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。

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

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

7. **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。

関連情報

- Red Hat Open Shift Container Platform のシークレットと Prometheus オペレーターの詳細については、Prometheus user guide on alerting を参照してください。

6.3.4. Alertmanager のテンプレート化によるアラートルートの作成

Alertmanager を使用して、電子メール、IRC、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配信します。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'
```

alertmanagerConfigManifest パラメーターに、送信されたアラートのタイトルとテキストを設定するためのカスタムテンプレートなどが含まれている場合、Base64 エンコードされた設定を使用して、**alertmanagerConfigManifest** のコンテンツ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. 次の例のように、alertmanager.yaml というファイルに必要な alertmanager 設定を作成します。

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

4. 設定マニフェストを生成して、STF デプロイメントの **ServiceTelemetry** オブジェクトに追加します。

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

5. 設定がシークレットに適用されたことを確認します。

注記

operator が各オブジェクトを更新するため、少し時間がかかります。

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

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

6. Prometheus Pod から **alertmanager-proxy** サービスに対して **wget** コマンドを実行して、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。

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

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

7. **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。

関連情報

- Red Hat Open Shift Container Platform のシークレットと Prometheus オペレーターの詳細については、Prometheus user guide on alerting を参照してください。

6.4. アラートを SNMP トラップとして送信する

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。SNMP トラップはバージョン 2c を使用して送信されます。

6.4.1. snmpTraps の設定パラメーター

snmpTraps パラメーターには、アラート受信者を設定するための次のサブパラメーターが含まれています。

enabled

SNMP トラップアラートレシーバーを有効にするには、このサブパラメーターの値を true に設定します。デフォルト値は false です。

target

SNMP トラップを送信するターゲットアドレス。値は文字列です。デフォルトは **192.168.24.254** です。

port

SNMP トラップを送信するターゲットポート。値は整数です。デフォルトは **162** です。

community

SNMP トラップの送信先のターゲットコミュニティー。値は文字列です。デフォルトは **public** です。

retries

SNMP トラップの再試行配信制限。値は整数です。デフォルトは **5** です。

timeout

秒単位で定義されていSNMP トラップ配信タイムアウト。値は整数です。デフォルトは **1** です。

alertOidLabel

SNMP トラップの送信に使用する OID 値を定義するアラート内のラベル名。値は文字列です。デフォルトは **oid** です。

trapOidPrefix

変数バインディングの SNMP トラップ OID 接頭辞。値は文字列です。デフォルトは **1.3.6.1.4.1.50495.15** です。

trapDefaultOid

アラートにアラート OID ラベルが指定されていない場合の SNMP トラップ OID。値は文字列です。デフォルトは **1.3.6.1.4.1.50495.15.1.2.1** です。

trapDefaultSeverity

アラート重大度が設定されていない場合の SNMP トラップ重大度。値は文字列です。デフォルトは空の文字列です。

ServiceTelemetry オブジェクトの **alerting.alertmanager.receivers** 定義の一部として **snmpTraps** パラメーターを設定します。

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

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

6.4.2. MIB 定義の概要

SNMP トラップの配信では、デフォルトでオブジェクト識別子 (OID) 値 **1.3.6.1.4.1.50495.15.1.2.1** が使用されます。管理情報ベース (MIB) スキーマは、https://github.com/infrawatch/prometheus-webhook-snmp/blob/master/PROMETHEUS-ALERT-CEPH-MIB.txt で入手できます。

OID 番号は、次のコンポーネント値で設定されます。* 値 **1.3.6.1.4.1** は、民間企業向けに定義されたグローバル OID です。* 次の識別子 **50495** は IANA によって Ceph 組織に割り当てられた民間企業番号です。※その他の値は親の子 OID です。

15
: prometheus オブジェクト

15.1
: prometheus アラート

15.1.2
: prometheus アラートトラップ

15.1.2.1
: prometheus アラートトラップのデフォルト

prometheus アラートトラップのデフォルト
は、**alerting.alertmanager.receivers.snmpTraps.trapOidPrefix** パラメーターによって定義される OID **1.3.6.1.4.1.50495.15** に対する他のいくつかのサブオブジェクトで設定されるオブジェクトです。

<trapOidPrefix>.1.1.1
: アラート名

<trapOidPrefix>.1.1.2
: status

<trapOidPrefix>.1.1.3
: severity

<trapOidPrefix>.1.1.4
: インスタンス

<trapOidPrefix>.1.1.5
: job

<trapOidPrefix>.1.1.6
: description

<trapOidPrefix>.1.1.7
: labels

<trapOidPrefix>.1.1.8
: timestamp

<trapOidPrefix>.1.1.9
: rawdata

以下は、受信したトラップをコンソールに出力する単純な SNMP トラップ受信者からの出力例です。

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

6.4.3. SNMP トラップの設定

前提条件

- アラートの送信先となる SNMP トラップ受信者の IP アドレスまたはホスト名を知っていることを確認してください。

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. 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
```

4. **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
```

5. **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。

追加情報

snmpTraps で使用可能なパラメーターの詳細については、「snmpTraps の設定パラメーター」 を参照してください。

6.4.4. SNMP トラップのアラートの作成

prometheus-webhook-snmp ミドルウェアによって解析されるラベルを追加して、トラップ情報と配信されるオブジェクト識別子 (OID) を定義することで、SNMP トラップによって配信されるように設定されたアラートを作成できます。**oid** ラベルまたは **severity** ラベルの追加は、特定のアラート定義のデフォルト値を変更する必要がある場合にのみ必要です。

注記

oid ラベルを設定すると、トップレベルの SNMP トラップ OID は変更されますが、サブ OID は、グローバルの **trapOidPrefix** 値と子 OID 値 **.1.1.1** ~ **.1.1.9** によって定義されたままになります。MIB 定義の詳細は、「MIB 定義の概要」 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. アラートルールと SNMP トラップ OID オーバーライド値を含む **oid** ラベルを含む **PrometheusRule** オブジェクトを作成します。

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

関連情報

アラートの設定については、「Service Telemetry Framework でのアラート」を参照してください。

6.5. 高可用性

警告

STF 高可用性 (HA) モードは非推奨であり、実稼働環境ではサポートされていません。Red Hat OpenShift Container Platform は高可用性プラットフォームであるため、HA モードを有効にすると問題が発生し、STF でのデバッグが複雑になる可能性があります。

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 クラスターにデプロイします。

高可用性を有効にすると、以下のような効果があります。

- 以下のコンポーネントは、デフォルトの 1 つの 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においても、Pod の紛失からの復旧時間は約 2 秒に短縮されます。

6.5.1. 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 コマンドで 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. 変更内容を保存し、オブジェクトを閉じます。

6.6. SERVICE TELEMETRY FRAMEWORK の可観測性ストラテジー

Service Telemetry Framework (STF) に、イベントストレージバックエンドやダッシュボードツールは含まれません。STF は、オプションでコミュニティー Operator を使用して Grafana のデータソース設定を作成し、ダッシュボードインターフェイスを提供できます。

Service Telemetry Operator がカスタムリソース要求を作成する代わりに、これらのアプリケーションまたは他の互換性のあるアプリケーションの独自のデプロイメントを使用し、Telemetry ストレージ用の独自の Prometheus 互換システムに配信するためにメトリクス Smart Gateways を収集できます。**observabilityStrategy** を **none** に設定すると、ストレージバックエンドはデプロイされないため、STF は永続ストレージを必要としません。

STF オブジェクトの observabilityStrategy プロパティーを使用して、デプロイされる可観測性コンポーネントのタイプを指定します。

以下の値を使用できます。

値	意味
use_redhat	STF は、Red Hat のサポート対象コンポーネントを要求します。これには、Cluster Observability Operator からの Prometheus と Alertmanager が含まれますが、Elastic Cloud on Kubernetes (ECK) Operator へのリソース要求は含まれません。有効にすると、Grafana Operator (コミュニティーコンポーネント) からもリソースが要求されます。
use_hybrid	Red Hat のサポート対象コンポーネントに加えて、Elasticsearch および Grafana リソースも要求されます (ServiceTelemetry オブジェクトで指定されている場合)。
use_community	Cluster Observability Operator の代わりに、Prometheus Operator のコミュニティーバージョンが使用されます。Elasticsearch および Grafana リソースも要求されます (ServiceTelemetry オブジェクトで指定されている場合)。

値	意味
none	ストレージまたはアラートコンポーネントはデプロイメントされていません。

注記

1.5.3 の時点で新しくデプロイされた STF 環境は、デフォルトで **use_redhat** になります。1.5.3 より前に作成された既存の STF デプロイメントは、デフォルトで **use_community** になります。

既存の STF デプロイメントを **use_redhat** に移行するには、Red Hat ナレッジベース Migrating Service Telemetry Framework to fully supported operators を参照してください。

6.6.1. 代替可観測性ストラテジーの設定

ストレージ、可視化、およびアラートバックエンドのデプロイメントを省略するには、ServiceTelemetry 仕様に **observabilityStrategy: none** を追加します。このモードでは、AMQ Interconnect ルーターと Smart Gateway のみをデプロイし、STF Smart Gateway からメトリクスを収集する外部 Prometheus 互換システムと、転送されたイベントを受信する外部 Elasticsearch を設定する必要があります。

手順

1. **spec** パラメーターに **observabilityStrategy: none** プロパティーを指定して **ServiceTelemetry** オブジェクトを作成します。マニフェストは、すべてのメトリクスコレクタータイプを持つ単一のクラウドから Telemetry を受け取るのに適した STF のデフォルトデプロイメントを示しています。

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

2. コミュニティー Operator が管理する残りのオブジェクトを削除します。

```
$ for o in alertmanagers.monitoring.rhobs/default prometheuses.monitoring.rhobs/default elasticsearch/elasticsearch grafana/default-grafana; do oc delete $o; done
```

3. すべてのワークロードが適切に動作していることを確認するには、Pod および各 Pod のステータスを確認します。

```
$ oc get pods
NAME                                                READY   STATUS    RESTARTS   AGE
default-cloud1-ceil-event-smartgateway-6f8547df6c-p2db5   3/3     Running   0          132m
default-cloud1-ceil-meter-smartgateway-59c845d65b-gzhcs   3/3     Running   0          132m
default-cloud1-coll-event-smartgateway-bf859f8d77-tzb66   3/3     Running   0          132m
```

```
default-cloud1-coll-meter-smartgateway-75bbd948b9-d5phm   3/3     Running   0          132m
default-cloud1-sens-meter-smartgateway-7fdbb57b6d-dh2g9   3/3     Running   0          132m
default-interconnect-668d5bbcd6-57b2l                     1/1     Running   0          132m
interconnect-operator-b8f5bb647-tlp5t                     1/1     Running   0          47h
service-telemetry-operator-566b9dd695-wkvjq               1/1     Running   0          156m
smart-gateway-operator-58d77dcf7-6xsq7                    1/1     Running   0          47h
```

関連情報

- 追加のクラウドの設定、またはサポート対象のコレクターのセットの変更に関する詳細は、「Smart Gateway の導入」 を参照してください。
- 既存の STF デプロイメントを **use_redhat** に移行するには、Red Hat ナレッジベース Migrating Service Telemetry Framework to fully supported operators を参照してください。

6.7. 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 (RHOSP) サービスのリソース使用状況を監視して、Compute 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。

関連情報

- 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。

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

RHOSP コンテナー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。

前提条件

- **stf-connectors.yaml** ファイルを作成している。詳細は、「ディレクターを使用した Service Telemetry Framework 用の 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 Open Stack Platform (RHOSP) 16.2 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- primarys.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導入手順を続行します。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。

6.8. RED HAT OPENSTACK PLATFORM API のステータスおよびコンテナー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実行して、各 Red Hat OpenStack Platform (RHOSP) サービスのコンテナー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 RHOSP サービスは、問題をログに記録し、バイナリーの状態を返すヘルスチェックを実装しています。RHOSP API の場合、ヘルスチェックはルートエンドポイントに問い合わせを行い、応答時間に基づいてヘルスを判断します。

RHOSP コンテナーの健全性と API の状態を監視する機能がデフォルトで有効になっています。

関連情報

- RHOSP コンテナーの健全性と API ステータスの監視を無効にするには、「コンテナーの正常性および 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。

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

RHOSP コンテナー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。

前提条件

- 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、「ディレクターを使用した Service Telemetry Framework 用の 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 Open Stack Platform (RHOSP) 16.2 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- primarys.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導入手順を続行します。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。

関連情報

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

第7章 RED HAT OPENSHIFT 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。

Red Hat OpenShift Container Platform 環境から STF を削除するには、以下のタスクを実行する必要があります。

1. namespace を削除します。
2. cert-manager Operator を削除します。
3. Cluster Observability Operator を削除します。

7.1. 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. 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.
```

7.2. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT の削除

他のアプリケーションに cert-manager Operator for Red Hat OpenShift を使用していない場合は、Subscription、ClusterServiceVersion、および CustomResourceDefinitions を削除します。

cert-manager Operator for Red Hat OpenShift の削除について、詳しくは **OpenShift Container Platform ドキュメント** の cert-manager Operator for Red Hat OpenShift の削除 を参照してください。

関連情報

- クラスターかの Operator を削除する

7.3. CLUSTER OBSERVABILITY OPERATOR の削除

他のアプリケーションに Cluster Observability Operator を使用していない場合は、Subscription、ClusterServiceVersion、および CustomResourceDefinitions を削除します。

Cluster Observability Operator の削除について、詳しくは **OpenShift Container Platform ドキュメント** の Web コンソールを使用して Cluster Observability Operator をアンインストールする を参照してください。

関連情報

- クラスターかの Operator を削除する