OpenShift Container Platform 4.17

分散トレーシング

OpenShift Container Platform での分散トレーシングの設定と使用

Last Updated: 2025-12-06

OpenShift Container Platform 4.17 分散トレーシング

OpenShift Container Platform での分散トレーシングの設定と使用

Legal Notice

Abstract

分散トレーシングを使用して、OpenShift Container Platform の分散システムを通過するマイクロサービストランザクションを保存、分析、視覚化します。

Table of Contents

第1章 RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM 3.8 のリリースノート

1.1. このリリースについて

分散 Tracing Platform 3.8 は、Tempo Operator 0.19.0 を通じて提供され、オープンソース Grafana Tempo 2.9.0 に基づいています。

注記

サポートされている機能のみが文書化されています。文書化されていない機能は現在サポートされていません。機能に関してサポートが必要な場合は、Red Hat のサポートにお問い合わせください。

1.2. 修正された問題

Tempo Pod に影響する TLS 証明書に関する問題を解決しました。

この更新の前は、内部 TLS 証明書が更新されたため、Tempo Pod は通信を停止していました。今回の更新により、証明書が更新されると Tempo Pod が自動的に再起動するようになりました。TRACING-5622

tempo クエリーフロントエンドがトレース JSON のフェッチに失敗しなくなりました。

今回の更新以前は、Jaeger UI で **Trace** をクリックし、ページを更新するか、Tempo クエリー frontend から **Trace → Trace Timeline → Trace JSON** にアクセスすると、Tempo クエリー Pod が EOF エラーを出して失敗する場合があります。今回の更新で、この問題は解決されました。TRACING-5483

注記

リンクされた Jira チケットの一部は、Red Hat の認証情報でのみアクセスできます。

1.3. サポートの利用

このドキュメントで説明されている手順、または OpenShift Container Platform 全般で問題が発生した場合は、Red Hat カスタマーポータル にアクセスしてください。

カスタマーポータルでは、次のことができます。

- Red Hat 製品に関するアーティクルおよびソリューションを対象とした Red Hat ナレッジベースの検索またはブラウズ。
- Red Hat サポートに対するサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

クラスターの問題を特定するには、OpenShift Cluster Manager で Insights を使用できます。Insights により、問題の詳細と、利用可能な場合は問題の解決方法に関する情報が提供されます。

このドキュメントを改善するための提案がある場合、またはエラーを見つけた場合は、最も関連性の高いドキュメントコンポーネントについて Jira 課題 を送信してください。セクション名や OpenShift Container Platform バージョンなどの具体的な情報を提供してください。

警告

非推奨になった Red Hat OpenShift Distributed Tracing Platform (Jaeger) 3.5 は、Red Hat がサポートする Red Hat OpenShift Distributed Tracing Platform (Jaeger) の最後のリリースでした。

非推奨になった Red Hat OpenShift Distributed Tracing Platform (Jaeger) 3.5 のサポートおよびメンテナンスはすべて、2025 年 11 月 3 日に終了します。

Red Hat OpenShift Distributed Tracing Platform (Jaeger)を依然として使用する場合は、分散トレースコレクションおよびストレージ用に Red Hat build of OpenTelemetry Operator および Tempo Operator に移行する必要があります。詳細は、Red Hat build of OpenTelemetry ドキュメントのMigrating"、Red Hat build of OpenTelemetry ドキュメントの Installing、および Red Hat OpenShift Distributed Tracing Platform ドキュメントの Installing を参照してください。

詳細は、Red Hat ナレッジベースソリューション Jaeger Deprecation and Removal in OpenShift を参照してください。

第2章 DISTRIBUTED TRACING PLATFORM について

2.1. 分散トレーシングの主要概念

ユーザーがアプリケーションでアクションを実行するたびに、応答を生成するために多数の異なるサービスに参加を要求する可能性のあるアーキテクチャーによって要求が実行されます。Red Hat OpenShift Distributed Tracing Platform を使用すると、分散トレーシングを実行し、アプリケーションを構成するさまざまなマイクロサービスによる要求のパスを記録できます。

分散トレーシング は、さまざまな作業単位 (通常は別々のプロセスまたはホストで実行されるもの) に関する情報を結び付けて、分散トランザクション内の一連のイベント全体を把握するために使用される手法です。分散トレーシングを使用すると、開発者は大規模なマイクロサービスアーキテクチャー内の呼び出しフローを可視化できます。これは、シリアル化、並行処理、およびレイテンシーのソースに関する理解にも役立ちます。

Red Hat OpenShift Distributed Tracing Platform は、マイクロサービスのスタック全体における個々の要求の実行を記録し、トレースとして表示します。**トレース** とは、システムにおけるデータ/実行パスです。エンドツーエンドのトレースは、1 つ以上のスパンで構成されます。

スパン は、Red Hat OpenShift Distributed Tracing Platform における論理的な作業単位を表します。これには、操作名、操作の開始時刻、期間、および場合によってはタグとログが含まれます。スパンは因果関係をモデル化するためにネスト化され、順序付けられます。

サービス所有者は、分散トレーシングを使用してサービスを計装し、サービスアーキテクチャーに関する分析情報を収集できます。Red Hat OpenShift Distributed Tracing Platform を使用すると、最新のクラウドネイティブのマイクロサービスベースのアプリケーションにおけるコンポーネント間の相互作用の監視、ネットワークプロファイリング、トラブルシューティングを行うことができます。

Distributed Tracing Platform を使用すると、次の機能を実行できます。

- 分散トランザクションの監視
- パフォーマンスとレイテンシーの最適化
- 根本原因分析の実行

Distributed Tracing Platform を OpenShift Container Platform の他の関連コンポーネントと組み合わせることができます。

- TempoStack インスタンスにトレースを転送するための Red Hat build of OpenTelemetry
- Cluster Observability Operator (COO) の分散トレーシング UI プラグイン

関連情報

- Red Hat build of OpenTelemetry
- 分散トレーシング UI プラグイン

2.2. RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM の機能

Red Hat OpenShift Distributed Tracing Platform は、次の機能を提供します。

- Kiali との統合 - 適切に設定すると、Kiali コンソールから Distributed Tracing Platform データを表示できます。

- 高いスケーラビリティー - Distributed Tracing Platform のバックエンドは、単一障害点がなく、ビジネスニーズに合わせて拡張できるように設計されています。
- 分散コンテキスト伝播 - さまざまなコンポーネントのデータを相互に接続して、完全なエンドツーエンドのトレースを作成できます。
- Zipkin との下位互換性 - Red Hat OpenShift Distributed Tracing Platform には、Zipkin のドロップインリプレースメントとして使用できる API があります。ただし、Red Hat はこのリリースでは Zipkin との互換性をサポートしていません。

2.3. RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM のアーキテクチャー

Red Hat OpenShift Distributed Tracing Platform は、トレースデータを収集、保存、表示するために連携して動作する複数のコンポーネントで構成されています。

- **Red Hat OpenShift Distributed Tracing Platform** このコンポーネントは、オープンソースの Grafana Tempo プロジェクト に基づいています。
  - **Gateway**: ゲートウェイは、認証、認可、およびディストリビューターまたはクエリーフロントエンドサービスへのリクエストの転送を処理します。
  - **Distributor**: ディストリビューターは、Jaeger、OpenTelemetry、Zipkin などの複数の形式のスパンを受け入れます。**traceID** をハッシュ化し、分散コンシステントハッシュリングを使用して、スパンを Ingester にルーティングします。
  - **Ingester**: Ingester はトレースをブロックにバッチ化し、ブルームフィルターとインデックスを作成してすべてバックエンドにフラッシュします。
  - **Query Frontend** - Query Frontend は、受信クエリーの検索スペースを分割し、クエリーをクエリー実行者に送信します。Query Frontend のデプロイメントでは、Tempo Query サイドカーを介して Jaeger UI が公開されます。
  - **Querier**: Querier は、Ingester またはバックエンドストレージで要求されたトレース ID を検索します。パラメーターに応じて、Ingester にクエリーを実行し、バックエンドから Bloom インデックスを取得して、オブジェクトストレージ内のブロックを検索できます。
  - **Compactor**: Compactor は、ブロックをバックエンドストレージとの間でストリーミングして、ブロックの総数を減らします。
- **Red Hat build of OpenTelemetry**- このコンポーネントは、オープンソースの OpenTelemetry プロジェクト に基づいています。
  - **OpenTelemetry Collector**: OpenTelemetry Collector は、テレメトリーデータを受信、処理、エクスポートするためのベンダーに依存しない方法です。OpenTelemetry Collector は、Jaeger や Prometheus などのオープンソースの可観測性データ形式をサポートし、1 つ以上のオープンソースまたは商用バックエンドに送信します。Collector は、インストルメンテーションライブラリーがテレメトリーデータをエクスポートするデフォルトの場所です。

第3章 DISTRIBUTED TRACING PLATFORM のインストール

Distributed Tracing Platform をインストールするには、次の手順を実行します。

1. Tempo Operator をインストールします。
2. サポートされているオブジェクトストアを設定し、オブジェクトストアの認証情報のシークレットを作成します。
3. 権限とテナントを設定します。
4. ユースケースに応じて次のデプロイメントを選択してインストールします。
   - マイクロサービスモードの **TempoStack** インスタンス
   - モノリシックモードの **TempoMonolithic** インスタンス

3.1. TEMPO OPERATOR のインストール

Tempo Operator は、Web コンソールまたはコマンドラインを使用してインストールできます。

3.1.1. Web コンソールを使用した Tempo Operator のインストール

Tempo Operator は、OpenShift Container Platform Web コンソールからインストールできます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者として、OpenShift Container Platform Web コンソールにログインしている。
- Red Hat OpenShift Dedicated の場合、**dedicated-admin** ロールを持つアカウントを使用してログインしている。
- サポートされているプロバイダーによる必要なオブジェクトストレージ Red Hat OpenShift Data Foundation、MinIO、Amazon S3、Azure Blob Storage、Google Cloud Storage の設定が完了している。詳細は、「オブジェクトストレージのセットアップ」を参照してください。

警告

オブジェクトストレージは必須ですが、Distributed Tracing Platform には含まれていません。Distributed Tracing Platform をインストールする前に、サポートされているプロバイダーによるオブジェクトストレージを選択して設定する必要があります。

手順

1. Web コンソールで、**Tempo Operator** を検索します。

ヒント

OpenShift Container Platform 4.19 以前では、**Operators → OperatorHub** に移動します。

OpenShift Container Platform 4.20 以降では、**エコシステム → ソフトウェアカタログ** に移動します。

2. **Red Hat が提供** する **Tempo Operator** を選択します。

重要

次の選択は、この Operator のデフォルトのプリセットです。

- **Update channel → stable**
- **Installation mode → All namespaces on the cluster**
- **Installed Namespace → openshift-tempo-operator**
- **Update approval → Automatic**

3. **Enable Operator recommended cluster monitoring on this Namespace**チェックボックスを選択します。
4. **Install → Install → View Operator** を選択します。

検証

- インストール済み Operator ページの **Details** タブの **ClusterServiceVersion details** で、インストールの **Status** が **Succeeded** であることを確認します。

3.1.2. CLI を使用した Tempo Operator のインストール

Tempo Operator はコマンドラインからインストールできます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者によるアクティブな OpenShift CLI (**oc**) セッション。

ヒント

  - OpenShift CLI (**oc**) のバージョンが最新であり、OpenShift Container Platform バージョンと一致していることを確認してください。
  - **oc login** を実行します。

```
$ oc login --username=<your_username>
```

- サポートされているプロバイダーによる必要なオブジェクトストレージ Red Hat OpenShift Data Foundation、MinIO、Amazon S3、Azure Blob Storage、Google Cloud Storage の設定が完了している。詳細は、「オブジェクトストレージのセットアップ」を参照してください。

警告

オブジェクトストレージは必須ですが、Distributed Tracing Platform には含まれていません。Distributed Tracing Platform をインストールする前に、サポートされているプロバイダーによるオブジェクトストレージを選択して設定する必要があります。

手順

1. 以下のコマンドを実行して、Tempo Operator のプロジェクトを作成します。

```
$ oc apply -f - << EOF
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  labels:
    kubernetes.io/metadata.name: openshift-tempo-operator
    openshift.io/cluster-monitoring: "true"
  name: openshift-tempo-operator
EOF
```

2. 以下のコマンドを実行して、Operator グループを作成します。

```
$ oc apply -f - << EOF
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-tempo-operator
  namespace: openshift-tempo-operator
spec:
  upgradeStrategy: Default
EOF
```

3. 以下のコマンドを実行して、サブスクリプションを作成します。

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

検証

- 次のコマンドを実行して、Operator のステータスを確認します。

```
$ oc get csv -n openshift-tempo-operator
```

3.2. オブジェクトストレージのセットアップ

サポートされているオブジェクトストレージを設定する際に、次の設定パラメーターを使用できます。

重要

オブジェクトストレージを使用するには、**TempoStack** または **TempoMonolithic** インスタンスをデプロイする前に、サポートされているオブジェクトストアを設定し、オブジェクトストアの認証情報のシークレットを作成する必要があります。

表3.1 必須のシークレットパラメーター

ストレージプロバイダー
Secret パラメーター
Red Hat OpenShift Data Foundation
name: tempostack-dev-odf # example **bucket: <bucket_name> # requires an ObjectBucketClaim** **endpoint: https://s3.openshift-storage.svc** **access_key_id: <data_foundation_access_key_id>** **access_key_secret: <data_foundation_access_key_secret>**
MinIO
MinIO Operator を参照してください。 **name: tempostack-dev-minio # example** **bucket: <minio_bucket_name> # MinIO documentation** **endpoint: <minio_bucket_endpoint>** **access_key_id: <minio_access_key_id>** **access_key_secret: <minio_access_key_secret>**
Amazon S3

ストレージプロバイダー
name: tempostack-dev-s3 # example **bucket: <s3_bucket_name> # Amazon S3 documentation** **endpoint: <s3_bucket_endpoint>** **access_key_id: <s3_access_key_id>** **access_key_secret: <s3_access_key_secret>**
Security Token Service (STS) を使用する Amazon S3
name: tempostack-dev-s3 # example **bucket: <s3_bucket_name> # Amazon S3 documentation** **region: <s3_region>** **role_arn: <s3_role_arn>**
Microsoft Azure Blob Storage
name: tempostack-dev-azure # example **container: <azure_blob_storage_container_name> # Microsoft Azure documentation** **account_name: <azure_blob_storage_account_name>** **account_key: <azure_blob_storage_account_key>**
Google Cloud Storage on Google Cloud
name: tempostack-dev-gcs # example **bucketname: <google_cloud_storage_bucket_name> # requires a bucket created in a Google Cloud project** **key.json: <path/to/key.json> # requires a service account in the bucket's GCP project for GCP authentication**

3.2.1. Security Token Service を使用する Amazon S3 ストレージの設定

Security Token Service (STS) と AWS Command Line Interface (AWS CLI) を使用して、Amazon S3 ストレージをセットアップできます。必要に応じて、Cloud Credential Operator (CCO) を使用することもできます。

重要

Amazon S3 ストレージおよび STS を使用した Distributed Tracing Platform の使用は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

前提条件

- AWS CLI の最新バージョンがインストールされている。
- CCO を使用する場合は、クラスターに CCO がインストールおよび設定されている。

手順

1. AWS S3 バケットを作成します。
2. 次のステップで作成する AWS Identity and Access Management (AWS IAM) ロールと、**TempoStack** または **TempoMonolithic** インスタンスのいずれかのサービスアカウントとの間に信頼関係を設定するために、AWS IAM ポリシー用に次の **trust.json** ファイルを作成します。

   trust.json

```
{
   "Version": "2012-10-17",
   "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<aws_account_id>:oidc-provider/<oidc_provider>" (1)
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "<oidc_provider>:sub": [
            "system:serviceaccount:<openshift_project_for_tempo>:tempo-<tempo_custom_resource_name>" (2)
            "system:serviceaccount:<openshift_project_for_tempo>:tempo-<tempo_custom_resource_name>-query-frontend"
          ]
        }
      }
    }
   ]
}
```

(1) OpenShift Container Platform で設定した OpenID Connect (OIDC) プロバイダー。

2 **TempoStack** または **TempoMonolithic** インスタンスのいずれかを作成する namespace。**<tempo_custom_resource_name>** は、**TempoStack** または

ヒント

次のコマンドを実行して、OIDC プロバイダーの値を取得することもできます。

```
$ oc get authentication cluster -o json | jq -r '.spec.serviceAccountIssuer' | sed
's~http[s]*://~~g'
```

3. 作成した **trust.json** ポリシーファイルをアタッチして AWS IAM ロールを作成します。これを行うには、次のコマンドを実行します。

```
$ aws iam create-role \
    --role-name "tempo-s3-access" \
    --assume-role-policy-document "file:///tmp/trust.json" \
    --query Role.Arn \
    --output text
```

4. 作成した AWS IAM ロールに AWS IAM ポリシーをアタッチします。これを行うには、次のコマンドを実行します。

```
$ aws iam attach-role-policy \
    --role-name "tempo-s3-access" \
    --policy-arn "arn:aws:iam::aws:policy/AmazonS3FullAccess"
```

5. CCO を使用していない場合は、このステップをスキップしてください。CCO を使用している場合は、Tempo Operator のクラウドプロバイダー環境を設定します。これを行うには、次のコマンドを実行します。

```
$ oc patch subscription <tempo_operator_sub> \ 1
      -n <tempo_operator_namespace> \ 2
      --type='merge' -p '{"spec": {"config": {"env": [{"name": "ROLEARN", "value": "'"
<role_arn>"'"}]}}}' 3
```

1 Tempo Operator サブスクリプションの名前。

2 Tempo Operator の namespace。

3 AWS STS では、Tempo Operator サブスクリプションに **ROLEARN** 環境変数を追加する必要があります。**<role_arn>** 値として、ステップ 3 で作成した AWS IAM ロールの Amazon Resource Name (ARN) を追加します。

6. OpenShift Container Platform で、次のように、キーを使用してオブジェクトストレージシークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name>
stringData:
```

```
  bucket: <s3_bucket_name>
  region: <s3_region>
  role_arn: <s3_role_arn>
type: Opaque
```

7. オブジェクトストレージシークレットが作成されたら、Distributed Tracing Platform インスタンスの関連するカスタムリソースを次のように更新します。

TempoStack カスタムリソースの例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  storage:
    secret: 1
      name: <secret_name>
      type: s3
      credentialMode: token-cco 2
# ...
```

1 前のステップで作成したシークレット。

2 CCO を使用していない場合は、この行を省略します。CCO を使用している場合は、**token-cco** 値とともにこのパラメーターを追加します。

TempoMonolithic カスタムリソースの例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  storage:
    traces:
      backend: s3
      s3:
        secret: <secret_name> 1
        credentialMode: token-cco 2
# ...
```

1 前のステップで作成したシークレット。

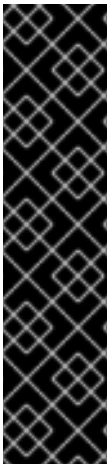
2 CCO を使用していない場合は、この行を省略します。CCO を使用している場合は、**token-cco** 値とともにこのパラメーターを追加します。

関連情報

- AWS Identity and Access Management Documentation (AWS ドキュメント)
- AWS Command Line Interface Documentation (AWS ドキュメント)
- OpenID Connect アイデンティティープロバイダーの設定
- Identify AWS resources with Amazon Resource Names (ARNs) (AWS ドキュメント)

3.2.2. Security Token Service を使用した Azure ストレージの設定

Azure Command Line Interface (Azure CLI) を使用して、Security Token Service (STS) を備えた Azure ストレージをセットアップできます。



重要

Azure ストレージおよび STS での Distributed Tracing Platform の使用は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

前提条件

- Azure CLI の最新バージョンがインストールされている。
- Azure ストレージアカウントを作成している。
- Azure Blob ストレージコンテナーを作成している。

手順

1. 次のコマンドを実行して、Azure マネージドアイデンティティーを作成します。

```
$ az identity create \
  --name <identity_name> \ 1
  --resource-group <resource_group> \ 2
  --location <region> \ 3
  --subscription <subscription_id> 4
```

1. マネージドアイデンティティーに選択した名前。
2. アイデンティティーを作成する Azure リソースグループ。
3. Azure リージョン。リソースグループと同じリージョンである必要があります。
4. Azure サブスクリプション ID。

2. Query Frontend を除く Distributed Tracing Platform のすべてのコンポーネントで使用するために、OpenShift Container Platform サービスアカウントのフェデレーションアイデンティティー認証情報を作成します。これを行うには、次のコマンドを実行します。

```
$ az identity federated-credential create \ (1)
  --name <credential_name> \ (2)
  --identity-name <identity_name> \
  --resource-group <resource_group> \
  --issuer <oidc_provider> \ (3)
  --subject <tempo_service_account_subject> \ (4)
  --audiences <audience> (5)
```

(1) フェデレーションアイデンティティー認証情報を使用すると、OpenShift Container Platform サービスアカウントは、シークレットを保存したり、Azure サービスプリンシパルアイデンティティーを使用したりすることなく、Azure マネージドアイデンティティーとして認証できます。

(2) フェデレーション認証情報に選択した名前。

(3) クラスターの OpenID Connect (OIDC) プロバイダーの URL。

(4) **system:serviceaccount:<namespace>:tempo-<tempostack_instance_name>** 形式のクラスターのサービスアカウントサブジェクト。

(5) フェデレーションアイデンティティー認証情報に対して発行されたトークンを検証するために使用される、想定されるオーディエンス。これは通常、**api://AzureADTokenExchange** に設定されます。

ヒント

次のコマンドを実行すると、クラスターの OpenID Connect (OIDC) 発行者の URL を取得できます。

```
$ oc get authentication cluster -o json | jq -r .spec.serviceAccountIssuer
```

3. Distributed Tracing Platform の Query Frontend コンポーネントで使用するために、OpenShift Container Platform サービスアカウントのフェデレーションアイデンティティー認証情報を作成します。これを行うには、次のコマンドを実行します。

```
$ az identity federated-credential create \ (1)
  --name <credential_name>-frontend \ (2)
  --identity-name <identity_name> \
  --resource-group <resource_group> \
  --issuer <cluster_issuer> \
  --subject <tempo_service_account_query_frontend_subject> \ (3)
  --audiences <audience> | jq
```

(1) フェデレーションアイデンティティー認証情報を使用すると、OpenShift Container Platform サービスアカウントは、シークレットを保存したり、Azure サービスプリンシパルアイデンティティーを使用したりすることなく、Azure マネージドアイデンティティーとして認証できます。

2 フロントエンドフェデレーションアイデンティティー認証情報に選択した名前。

3 **system:serviceaccount:<namespace>:tempo-<tempostack_instance_name>** 形式のクラスターのサービスアカウントサブジェクト。

4. 作成された Azure マネージドアイデンティティーの Azure サービスプリンシパルアイデンティティーに、Storage Blob Data Contributor ロールを割り当てます。これを行うには、次のコマンドを実行します。

```
$ az role assignment create \
  --assignee <assignee_name> \ 1
  --role "Storage Blob Data Contributor" \
  --scope "/subscriptions/<subscription_id>
```

1 ステップ 1 で作成した Azure マネージドアイデンティティーの Azure サービスプリンシパルアイデンティティー。

ヒント

次のコマンドを実行すると、**<assignee_name>** の値を取得できます。

```
$ az ad sp list --all --filter "servicePrincipalType eq 'ManagedIdentity'" | jq -r --arg idName
<identity_name> '.[] | select(.displayName == $idName) | .appId'`
```

5. ステップ 1 で作成した Azure マネージドアイデンティティーのクライアント ID を取得します。

```
CLIENT_ID=$(az identity show \
  --name <identity_name> \ 1
  --resource-group <resource_group> \ 2
  --query clientId \
  -o tsv)
```

1 ステップ 1 の **<identity_name>** 値をコピーして貼り付けます。

2 ステップ 1 の **<resource_group>** 値をコピーして貼り付けます。

6. Azure Workload Identity Federation (WIF) 用の OpenShift Container Platform シークレットを作成します。これを行うには、次のコマンドを実行します。

```
$ oc create -n <tempo_namespace> secret generic azure-secret \
  --from-literal=container=<azure_storage_azure_container> \ 1
  --from-literal=account_name=<azure_storage_azure_accountname> \ 2
  --from-literal=client_id=<client_id> \ 3
  --from-literal=audience=<audience> \ 4
  --from-literal=tenant_id=<tenant_id> 5
```

1 Azure Blob Storage コンテナーの名前。

2 Azure Storage アカウントの名前。

3 前のステップで取得したマネージドアイデンティティーのクライアント ID。

4 オプション: デフォルトは **api://AzureADTokenExchange** です。

5 Azure テナント ID。

7. オブジェクトストレージシークレットが作成されたら、Distributed Tracing Platform インスタンスの関連するカスタムリソースを次のように更新します。

TempoStack カスタムリソースの例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  storage:
    secret: 1
      name: <secret_name>
      type: azure
# ...
```

1 前のステップで作成したシークレット。

TempoMonolithic カスタムリソースの例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  storage:
    traces:
      backend: azure
      azure:
        secret: <secret_name> 1
# ...
```

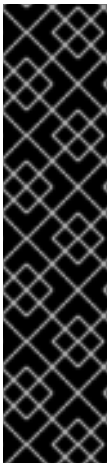
1 前のステップで作成したシークレット。

関連情報

- Install the Azure CLI on Linux (Azure ドキュメント)

3.2.3. Security Token Service を使用した Google Cloud ストレージのセットアップ

Google Cloud CLI を使用して、Security Token Service (STS) を利用する Google Cloud Storage (GCS) をセットアップできます。



重要

GCS および STS を使用した Distributed Tracing Platform の使用は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

前提条件

- Google Cloud CLI の最新バージョンがインストールされている。

手順

1. Google Cloud に GCS バケットを作成します。
2. Google の Identity and Access Management (IAM) を使用して、サービスアカウントを作成または再利用します。

```
SERVICE_ACCOUNT_EMAIL=$(gcloud iam service-accounts create
<iam_service_account_name> \ 1
   --display-name="Tempo Account" \
   --project <project_id>  \ 2
   --format='value(email)' \
   --quiet)
```

1. Google Cloud 上のサービスアカウントの名前。
2. Google Cloud 上のサービスアカウントのプロジェクト ID。

3. 必要な Google Cloud ロールを、プロジェクトレベルで作成されたサービスアカウントにバインドします。これを行うには、次のコマンドを実行します。

```
$ gcloud projects add-iam-policy-binding <project_id> \
   --member "serviceAccount:$SERVICE_ACCOUNT_EMAIL" \
   --role "roles/storage.objectAdmin"
```

4. クラスターに関連付けられている Google Cloud Workload Identity Pool の **POOL_ID** 値を取得します。この値を取得する方法は環境によって異なるため、次のコマンドは単なる例です。

```
$ OIDC_ISSUER=$(oc get authentication.config cluster -o
jsonpath='{.spec.serviceAccountIssuer}') \
&&
  POOL_ID=$(echo "$OIDC_ISSUER" | awk -F'/' '{print $NF}' | sed 's/-oidc$//')
```

5. IAM ポリシーバインディングを追加します。これを行うには、次のコマンドを実行します。

```
$ gcloud iam service-accounts add-iam-policy-binding "$SERVICE_ACCOUNT_EMAIL" \ 1
  --role="roles/iam.workloadIdentityUser" \
  --
member="principal://iam.googleapis.com/projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/subject/system:serviceaccount:<tempo_namespace>:tempo-<tempo_name>" \
  --project=<project_id> \
  --quiet \
&&
  gcloud iam service-accounts add-iam-policy-binding "$SERVICE_ACCOUNT_EMAIL" \
  --role="roles/iam.workloadIdentityUser" \
  --
member="principal://iam.googleapis.com/projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/subject/system:serviceaccount:<tempo_namespace>:tempo-<tempo_name>-query-frontend" \
  --project=<project_id> \
  --quiet
&&
  gcloud storage buckets add-iam-policy-binding "gs://$BUCKET_NAME" \
  --role="roles/storage.admin" \
  --member="serviceAccount:$SERVICE_ACCOUNT_EMAIL" \
  --condition=None
```

1 **$SERVICE_ACCOUNT_EMAIL** は、ステップ 2 のコマンドの出力です。

6. **TempoStack** カスタムリソースで使用するストレージシークレットの **key.json** キーの認証情報ファイルを作成します。これを行うには、次のコマンドを実行します。

```
$ gcloud iam workload-identity-pools create-cred-config \

"projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/providers/<provider_id>" \
    --service-account="$SERVICE_ACCOUNT_EMAIL" \
    --credential-source-file=/var/run/secrets/storage/serviceaccount/token \ 1
    --credential-source-type=text \
    --output-file=<output_file_path> 2
```

1 Operator はこのパスからトークンをマウントするため、**credential-source-file** パラメーターは常に **/var/run/secrets/storage/serviceaccount/token** パスを指している必要があります。

2 出力ファイルを保存するためのパス。

7. 次のコマンドを実行して、正しいオーディエンスを取得します。

```
$ gcloud iam workload-identity-pools providers describe "$PROVIDER_NAME" --format='value(oidc.allowedAudiences[0])'
```

8. 次のコマンドを実行して、Distributed Tracing Platform のストレージシークレットを作成します。

```
$ oc -n <tempo_namespace> create secret generic gcs-secret \
  --from-literal=bucketname="<bucket_name>" \ 1
  --from-literal=audience="<audience>" \ 2
  --from-file=key.json=<output_file_path> 3
```

1. Google Cloud Storage のバケット名。
2. 前のステップで取得したオーディエンス。
3. ステップ 6 で作成した認証情報ファイル。

9. オブジェクトストレージシークレットが作成されたら、Distributed Tracing Platform インスタンスの関連するカスタムリソースを次のように更新します。

TempoStack カスタムリソースの例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  storage:
    secret: 1
      name: <secret_name>
      type: gcs
# ...
```

1. 前のステップで作成したシークレット。

TempoMonolithic カスタムリソースの例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  storage:
    traces:
      backend: gcs
      gcs:
        secret: <secret_name> 1
# ...
```

1. 前のステップで作成したシークレット。

関連情報

- Install the gcloud CLI (Google Cloud ドキュメント)
- Service accounts overview (Google Cloud ドキュメント)

3.2.4. IBM Cloud Object Storage の設定

OpenShift CLI (**oc**) を使用して IBM Cloud Object Storage をセットアップできます。

前提条件

- OpenShift CLI (**oc**) の最新バージョンをインストールした。詳細は、**設定: CLI ツール** の「OpenShift CLI の使用を開始する」を参照してください。
- IBM Cloud Command Line Interface (**ibmcloud**) の最新バージョンをインストールした。詳細は、**IBM Cloud Docs** の「Getting started with the IBM Cloud CLI」を参照してください。
- IBM Cloud Object Storage を設定した。詳細は、**IBM Cloud Docs** の「Choosing a plan and creating an instance」を参照してください。
  - IBM Cloud Platform アカウントを持っている。
  - IBM Cloud Object Storage のプランを発注した。
  - IBM Cloud Object Storage のインスタンスを作成した。

手順

1. IBM Cloud でオブジェクトストアバケットを作成します。
2. IBM Cloud で、次のコマンドを実行して、オブジェクトストアバケットに接続するためのサービスキーを作成します。

```
$ ibmcloud resource service-key-create <tempo_bucket> Writer \
  --instance-name <tempo_bucket> --parameters '{"HMAC":true}'
```

3. IBM Cloud で、次のコマンドを実行して、バケット認証情報を含むシークレットを作成します。

```
$ oc -n <namespace> create secret generic <ibm_cos_secret> \
  --from-literal=bucket="<tempo_bucket>" \
  --from-literal=endpoint="<ibm_bucket_endpoint>" \
  --from-literal=access_key_id="<ibm_bucket_access_key>" \
  --from-literal=access_key_secret="<ibm_bucket_secret_key>"
```

4. OpenShift Container Platform で、次のように、キーを使用してオブジェクトストレージシークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: <ibm_cos_secret>
stringData:
  bucket: <tempo_bucket>
  endpoint: <ibm_bucket_endpoint>
```

```
  access_key_id: <ibm_bucket_access_key>
  access_key_secret: <ibm_bucket_secret_key>
type: Opaque
```

5. OpenShift Container Platform で、**TempoStack** カスタムリソースのストレージセクションを次のように設定します。

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
# ...
spec:
# ...
  storage:
    secret:
      name: <ibm_cos_secret> (1)
      type: s3
# ...
```

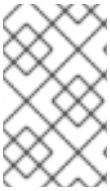
(1) IBM Cloud Storage のアクセスキーとシークレットキーが含まれるシークレットの名前。

関連情報

- OpenShift CLI の使用を開始する
- Getting started with the IBM Cloud CLI (IBM Cloud Docs)
- Choosing a plan and creating an instance (IBM Cloud Docs)
- Getting started with IBM Cloud Object Storage: Before you begin (IBM Cloud Docs)

3.3. 権限とテナントの設定

TempoStack または **TempoMonolithic** インスタンスをインストールする前に、1 つ以上のテナントを定義し、テナントの読み取りおよび書き込みアクセス権を設定する必要があります。このような認可設定は、Kubernetes のロールベースアクセス制御 (RBAC) のクラスターロールとクラスターロールバインディングを使用して設定できます。デフォルトでは、どのユーザーにも読み取り権限または書き込み権限は付与されません。詳細は、「テナントの読み取り権限の設定」および「テナントの書き込み権限の設定」を参照してください。



注記

Red Hat build of OpenTelemetry の OpenTelemetry Collector は、データの書き込み用のサービスアカウントと RBAC を使用して、トレースデータを **TempoStack** または **TempoMonolithic** インスタンスに送信できます。

表3.2 認証および認可

Component	Tempo Gateway サービス	OpenShift OAuth	**TokenReview** API	**SubjectAccess Review** API
Authentication	X	X	X	

Component	Tempo Gateway サービス	OpenShift OAuth	TokenReview API	SubjectAccess Review API
認可	X			X

3.3.1. テナントの読み取り権限の設定

テナントの読み取り権限は、Web コンソールの **Administrator** ビューまたはコマンドラインから設定できます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者として、OpenShift Container Platform Web コンソールにログインしている。
- Red Hat OpenShift Dedicated の場合、**dedicated-admin** ロールを持つアカウントを使用してログインしている。

手順

1. 任意の値を指定した **tenantName** および **tenantId** パラメーターを **TempoStack** カスタムリソース (CR) に追加して、テナントを定義します。

   TempoStack CR のテナントの例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: redmetrics
spec:
# ...
  tenants:
    mode: openshift
    authentication:
      - tenantName: dev (1)
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" (2)
# ...
```

   (1) ユーザーが選択した **tenantName** 値。

   (2) ユーザーが選択した **tenantId** 値。

2. トレースを読み取るための読み取り (**get**) 権限を持つクラスターロールにテナントを追加します。

   ClusterRole リソースの RBAC 設定の例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
  name: tempostack-traces-reader
rules:
  - apiGroups:
      - 'tempo.grafana.com'
    resources: 1
      - dev
      - prod
    resourceNames:
      - traces
    verbs:
      - 'get' 2
```

1 この例では、前のステップで **tenantName** パラメーターを使用して定義したテナント (**dev** および **prod**) をリストします。

2 リストしたテナントの読み取り操作を有効にします。

3. 上記ステップのクラスターロールのクラスターロールバインディングを定義して、認証されたユーザーにトレースデータの読み取り権限を付与します。

ClusterRoleBinding リソースの RBAC 設定の例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tempostack-traces-reader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tempostack-traces-reader
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated 1
```

1 認証されたユーザー全員に、トレースデータの読み取り権限を付与します。

3.3.2. テナントの書き込み権限の設定

テナントの書き込み権限は、Web コンソールの **Administrator** ビューまたはコマンドラインから設定できます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者として、OpenShift Container Platform Web コンソールにログインしている。
- Red Hat OpenShift Dedicated の場合、**dedicated-admin** ロールを持つアカウントを使用してログインしている。

- OpenTelemetry Collector をインストールし、権限を持つ許可済みのサービスアカウントを使用するように Collector を設定した。詳細は、Red Hat build of OpenTelemetry ドキュメントの「必要な RBAC リソースの自動作成」を参照してください。

手順

1. OpenTelemetry Collector で使用するためのサービスアカウントを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: otel-collector
  namespace: <project_of_opentelemetry_collector_instance>
```

2. トレースを書き込むための書き込み (**create**) 権限を持つクラスターロールにテナントを追加します。

ClusterRole リソースの RBAC 設定の例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tempostack-traces-write
rules:
  - apiGroups:
      - 'tempo.grafana.com'
    resources: 1
      - dev
    resourceNames:
      - traces
    verbs:
      - 'create' 2
```

1. テナントをリスト表示します。
2. 書き込み操作を有効にします。

3. OpenTelemetry Collector のサービスアカウントを割り当てるためのクラスターロールバインディングを定義して、OpenTelemetry Collector に書き込み権限を付与します。

ClusterRoleBinding リソースの RBAC 設定の例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tempostack-traces
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tempostack-traces-write
subjects:
```

```
- kind: ServiceAccount
  name: otel-collector 1
  namespace: otel
```

1 前のステップで作成したサービスアカウント。これは、クライアントがトレースデータをエクスポートするときに使用されます。

4. **OpenTelemetryCollector** カスタムリソースを次のように設定します。
   - トレーシングパイプラインサービスに、**bearertokenauth** エクステンションと有効なトークンを追加します。
   - **otlp/otlphttp** エクスポーターにテナント名を **X-Scope-OrgID** ヘッダーとして追加します。
   - 有効な認証局ファイルを使用して TLS を有効にします。

   OpenTelemetry CR 設定のサンプル

```
apiVersion: opentelemetry.io/v1beta1
kind: OpenTelemetryCollector
metadata:
  name: cluster-collector
  namespace: <project_of_tempostack_instance>
spec:
  mode: deployment
  serviceAccount: otel-collector 1
  config: |
    extensions:
      bearertokenauth: 2
        filename: "/var/run/secrets/kubernetes.io/serviceaccount/token" 3
    exporters:
      otlp/dev: 4
        endpoint: sample-gateway.tempo.svc.cluster.local:8090
        tls:
          insecure: false
          ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt" 5
        auth:
          authenticator: bearertokenauth
        headers:
          X-Scope-OrgID: "dev" 6
      otlphttp/dev: 7
        endpoint: https://sample-gateway.
<project_of_tempostack_instance>.svc.cluster.local:8080/api/traces/v1/dev
        tls:
          insecure: false
          ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
        auth:
          authenticator: bearertokenauth
        headers:
          X-Scope-OrgID: "dev"
    service:
      extensions: [bearertokenauth]
      pipelines:
        traces:
```

```
        exporters: [otlp/dev] 8
# ...
```

1. 書き込み権限が設定されたサービスアカウント。
2. サービスアカウントトークンを使用するためのベアラートークンエクステンション。
3. サービスアカウントトークン。このトークンは、ベアラートークンヘッダーとして、クライアントによりトレーシングパイプラインサービスに送信されます。
4. OTLP gRPC Exporter (**otlp/dev**) または OTLP HTTP Exporter (**otlphttp/dev**) のいずれかを指定します。
5. 有効なサービス CA ファイルを使用して TLS を有効にします。
6. テナント名を含むヘッダー。
7. OTLP gRPC Exporter (**otlp/dev**) または OTLP HTTP Exporter (**otlphttp/dev**) のいずれかを指定します。
8. CR の **exporters** セクションで指定したエクスポーター。

関連情報

- 必要な RBAC リソースの自動作成

3.4. TEMPOSTACK インスタンスのインストール

TempoStack インスタンスは、Web コンソールまたはコマンドラインを使用してインストールできます。

3.4.1. Web コンソールを使用した TempoStack インスタンスのインストール

Web コンソールの **Administrator** ビューから **TempoStack** インスタンスをインストールできます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者として、OpenShift Container Platform Web コンソールにログインしている。
- Red Hat OpenShift Dedicated の場合、**dedicated-admin** ロールを持つアカウントを使用してログインしている。
- サポートされているプロバイダーによる必要なオブジェクトストレージ Red Hat OpenShift Data Foundation、MinIO、Amazon S3、Azure Blob Storage、Google Cloud Storage の設定が完了している。詳細は、「オブジェクトストレージのセットアップ」を参照してください。

警告

オブジェクトストレージは必須ですが、Distributed Tracing Platform には含まれていません。Distributed Tracing Platform をインストールする前に、サポートされているプロバイダーによるオブジェクトストレージを選択して設定する必要があります。

- 1 つ以上のテナントを定義し、読み取りおよび書き込み権限を設定した。詳細は、「テナントの読み取り権限の設定」および「テナントの書き込み権限の設定」を参照してください。

手順

1. **Home → Projects → Create Project** に移動して、後続のステップで作成する **TempoStack** インスタンス用に、許可される任意のプロジェクトを作成します。**openshift-** 接頭辞で始まるプロジェクト名は許可されません。

2. **Workloads → Secrets → Create → From YAML** に移動して、**TempoStack** インスタンス用に作成したプロジェクトに、オブジェクトストレージバケットのシークレットを作成します。詳細は、「オブジェクトストレージのセットアップ」を参照してください。

   Amazon S3 および MinIO ストレージのシークレット例

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: minio-test
   stringData:
     endpoint: http://minio.minio.svc:9000
     bucket: tempo
     access_key_id: tempo
     access_key_secret: <secret>
   type: Opaque
   ```

3. **TempoStack** インスタンスを作成します。

   注記

   同じクラスター上の別々のプロジェクトに、複数の **TempoStack** インスタンスを作成できます。

   a. **Operators → Installed Operators** に移動します。

   b. **TempoStack → Create TempoStack → YAML view** の順に選択します。

   c. **YAML view** で、**TempoStack** カスタムリソース (CR) をカスタマイズします。

      AWS S3 および MinIO ストレージと 2 つのテナント用の TempoStack CR の例

      ```
      apiVersion: tempo.grafana.com/v1alpha1
      ```

```
kind: TempoStack (1)
metadata:
  name: simplest
  namespace: <permitted_project_of_tempostack_instance> (2)
spec: (3)
  storage: (4)
    secret: (5)
      name: <secret_name> (6)
      type: <secret_provider> (7)
  storageSize: <value>Gi (8)
  resources: (9)
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  tenants:
    mode: openshift (10)
    authentication: (11)
      - tenantName: dev (12)
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" (13)
      - tenantName: prod
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
  template:
    gateway:
      enabled: true (14)
    queryFrontend:
      jaegerQuery:
        enabled: true (15)
```

(1) この CR は、HTTP および OpenTelemetry Protocol (OTLP) 経由で Jaeger Thrift を受信するように設定された **TempoStack** デプロイメントを作成します。

(2) **TempoStack** デプロイメント用に選択したプロジェクト。**openshift-** 接頭辞で始まるプロジェクト名は許可されません。

(3) Red Hat は、Red Hat OpenShift Distributed Tracing Platform ドキュメントに記載されているカスタムリソースオプションのみをサポートしています。

(4) トレースを保存するためのストレージを指定します。

(5) 前提条件の 1 つとして設定したオブジェクトストレージ用に、ステップ 2 で作成したシークレット。

(6) シークレットの **metadata** セクションにある **name** フィールドの値。たとえば、**minio** です。

(7) この値には、Azure Blob Storage の場合は **azure**、Google Cloud Storage の場合は **gcs**、Amazon S3、MinIO、または Red Hat OpenShift Data Foundation の場合は **s3** を使用できます。たとえば、**s3** です。

(8) Tempo Write-Ahead Logging (WAL) の永続ボリューム要求のサイズ。デフォルトは **10Gi** です。たとえば、**1Gi** のように指定します。

(9) 任意。

10 値は **openshift** である必要があります。

11 テナントのリスト。

12 テナント名。**X-Scope-OrgId** HTTP ヘッダーの値として使用されます。

13 テナントの一意の識別子。**TempoStack** デプロイメントのライフサイクル全体を通じて一意である必要があります。Distributed Tracing Platform は、この ID を使用して、オブジェクトストレージ内のオブジェクトに接頭辞を付けます。UUID または **tempoName** フィールドの値を再利用できます。

14 認証と認可を実行するゲートウェイを有効にします。

15 **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search** のルート経由で、データを視覚化する Jaeger UI を公開します。

d. **Create** を選択します。

検証

1. **Project:** ドロップダウンリストを使用して、**TempoStack** インスタンスのプロジェクトを選択します。
2. **Operators → Installed Operators** に移動して、**TempoStack** インスタンスの **Status** が **Condition: Ready** であることを確認します。
3. **Workloads → Pods** に移動して、**TempoStack** インスタンスのすべてのコンポーネント Pod が稼働していることを確認します。
4. Tempo コンソールにアクセスします。
   a. **Networking → Routes** に移動し、**Ctrl+F** で **tempo** を検索します。
   b. **Location** 列で URL を開き、Tempo コンソールにアクセスします。

注記

Tempo コンソールをインストールした直後は、Tempo コンソールにトレースデータは表示されません。

3.4.2. CLI を使用した TempoStack インスタンスのインストール

コマンドラインから **TempoStack** インスタンスをインストールできます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者によるアクティブな OpenShift CLI (**oc**) セッション。

ヒント

- OpenShift CLI (**oc**) のバージョンが最新であり、OpenShift Container Platform バージョンと一致していることを確認してください。
- **oc login** コマンドを実行します。

```
$ oc login --username=<your_username>
```

- サポートされているプロバイダーによる必要なオブジェクトストレージ Red Hat OpenShift Data Foundation、MinIO、Amazon S3、Azure Blob Storage、Google Cloud Storage の設定が完了している。詳細は、「オブジェクトストレージのセットアップ」を参照してください。

警告

オブジェクトストレージは必須ですが、Distributed Tracing Platform には含まれていません。Distributed Tracing Platform をインストールする前に、サポートされているプロバイダーによるオブジェクトストレージを選択して設定する必要があります。

- 1 つ以上のテナントを定義し、読み取りおよび書き込み権限を設定した。詳細は、「テナントの読み取り権限の設定」および「テナントの書き込み権限の設定」を参照してください。

手順

1. 次のコマンドを実行して、後続のステップで作成する **TempoStack** インスタンス用に、許可される任意のプロジェクトを作成します。

```
$ oc apply -f - << EOF
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  name: <permitted_project_of_tempostack_instance> 1
EOF
```

1 **openshift-** 接頭辞で始まるプロジェクト名は許可されません。

2. **TempoStack** インスタンス用に作成したプロジェクトで、次のコマンドを実行して、オブジェクトストレージバケットのシークレットを作成します。

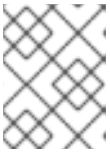
```
$ oc apply -f - << EOF
<object_storage_secret>
EOF
```

詳細は、「オブジェクトストレージのセットアップ」を参照してください。

Amazon S3 および MinIO ストレージのシークレット例

```
apiVersion: v1
kind: Secret
metadata:
  name: minio-test
stringData:
  endpoint: http://minio.minio.svc:9000
  bucket: tempo
  access_key_id: tempo
  access_key_secret: <secret>
type: Opaque
```

3. **TempoStack** インスタンス用に作成したプロジェクトに TempoStack インスタンスを作成します。



注記

同じクラスター上の別々のプロジェクトに、複数の **TempoStack** インスタンスを作成できます。

a. **TempoStack** カスタムリソース (CR) をカスタマイズします。

AWS S3 および MinIO ストレージと 2 つのテナント用の TempoStack CR の例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack (1)
metadata:
  name: simplest
  namespace: <permitted_project_of_tempostack_instance> (2)
spec: (3)
  storage: (4)
    secret: (5)
      name: <secret_name> (6)
      type: <secret_provider> (7)
  storageSize: <value>Gi (8)
  resources: (9)
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  tenants:
    mode: openshift (10)
    authentication: (11)
      - tenantName: dev (12)
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" (13)
      - tenantName: prod
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
  template:
    gateway:
      enabled: true (14)
    queryFrontend:
      jaegerQuery:
        enabled: true (15)
```

1. この CR は、HTTP および OpenTelemetry Protocol (OTLP) 経由で Jaeger Thrift を受信するように設定された **TempoStack** デプロイメントを作成します。
2. **TempoStack** デプロイメント用に選択したプロジェクト。**openshift-** 接頭辞で始まるプロジェクト名は許可されません。
3. Red Hat は、Red Hat OpenShift Distributed Tracing Platform ドキュメントに記載されているカスタムリソースオプションのみをサポートしています。
4. トレースを保存するためのストレージを指定します。
5. 前提条件の 1 つとして設定したオブジェクトストレージ用に、ステップ 2 で作成したシークレット。
6. シークレットの **metadata** セクションにある **name** フィールドの値。たとえば、**minio** です。
7. この値には、Azure Blob Storage の場合は **azure**、Google Cloud Storage の場合は **gcs**、Amazon S3、MinIO、または Red Hat OpenShift Data Foundation の場合は **s3** を使用できます。たとえば、**s3** です。
8. Tempo Write-Ahead Logging (WAL) の永続ボリューム要求のサイズ。デフォルトは **10Gi** です。たとえば、**1Gi** のように指定します。
9. 任意。
10. 値は **openshift** である必要があります。
11. テナントのリスト。
12. テナント名。**X-Scope-OrgId** HTTP ヘッダーの値として使用されます。
13. テナントの一意の識別子。**TempoStack** デプロイメントのライフサイクル全体を通じて一意である必要があります。Distributed Tracing Platform は、この ID を使用して、オブジェクトストレージ内のオブジェクトに接頭辞を付けます。UUID または **tempoName** フィールドの値を再利用できます。
14. 認証と認可を実行するゲートウェイを有効にします。
15. **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search** のルート経由で、データを視覚化する Jaeger UI を公開します。

b. 次のコマンドを実行して、カスタマイズされた CR を適用します。

```
$ oc apply -f - << EOF
<tempostack_cr>
EOF
```

検証

1. 次のコマンドを実行して、すべての **TempoStack components** の **status** が **Running**、**conditions** が **type: Ready** になっていることを確認します。

```
$ oc get tempostacks.tempo.grafana.com simplest -o yaml
```

2. 次のコマンドを実行して、すべての **TempoStack** コンポーネント Pod が稼働していることを確認します。

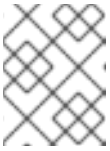
```
$ oc get pods
```

3. Tempo コンソールにアクセスします。

   a. 以下のコマンドを実行してルートの詳細をクエリーします。

```
$ oc get route
```

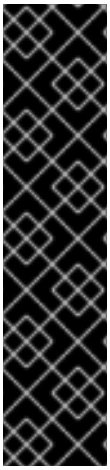
   b. Web ブラウザーで **https://<route_from_previous_step>** を開きます。



注記

Tempo コンソールをインストールした直後は、Tempo コンソールにトレースデータは表示されません。

3.5. TEMPOMONOLITHIC インスタンスのインストール



重要

TempoMonolithic インスタンスは、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

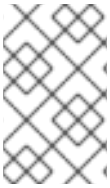
- テクノロジープレビュー機能のサポート範囲

TempoMonolithic インスタンスは、Web コンソールまたはコマンドラインを使用してインストールできます。

TempoMonolithic カスタムリソース (CR) は、モノリシックモードで Tempo デプロイメントを作成します。コンパクター、ディストリビューター、インジェスター、クエリアー、クエリーフロントエンドなど、Tempo デプロイメントのすべてのコンポーネントが、単一のコンテナーに含まれます。

TempoMonolithic インスタンスは、インメモリーストレージ、永続ボリューム、またはオブジェクトストレージへのトレースの保存をサポートしています。

小規模なデプロイメント、デモンストレーション、テストには、モノリシックモードでの Tempo デプロイメントが適しています。



注記

Tempo のモノリシックデプロイメントは水平方向にスケーリングできません。水平スケーリングが必要な場合は、マイクロサービスモードでの Tempo デプロイメント用の **TempoStack** CR を使用してください。

3.5.1. Web コンソールを使用した TempoMonolithic インスタンスのインストール

重要

TempoMonolithic インスタンスは、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

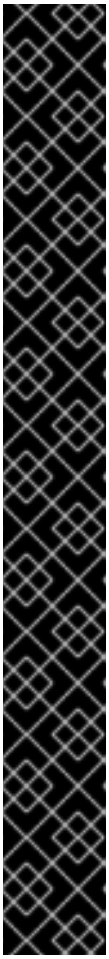
Web コンソールの **Administrator** ビューから **TempoMonolithic** インスタンスをインストールできます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者として、OpenShift Container Platform Web コンソールにログインしている。
- Red Hat OpenShift Dedicated の場合、**dedicated-admin** ロールを持つアカウントを使用してログインしている。
- 1 つ以上のテナントを定義し、読み取りおよび書き込み権限を設定した。詳細は、「テナントの読み取り権限の設定」および「テナントの書き込み権限の設定」を参照してください。

手順

1. **Home → Projects → Create Project** に移動して、後続のステップで作成する **TempoMonolithic** インスタンス用に、許可される任意のプロジェクトを作成します。**openshift-** 接頭辞で始まるプロジェクト名は許可されません。
2. トレースの保存に使用するサポート対象のストレージのタイプ (インメモリーストレージ、永続ボリューム、オブジェクトストレージ) を決定します。



重要

オブジェクトストレージは、Distributed Tracing Platform には含まれていません。そのため、サポートされているプロバイダー (Red Hat OpenShift Data Foundation、MinIO、Amazon S3、Azure Blob Storage、または Google Cloud Storage) によるオブジェクトストアを設定する必要があります。

また、オブジェクトストレージを選択するには、**TempoMonolithic** インスタンス用に作成したプロジェクトにオブジェクトストレージバケットのシークレットを作成する必要があります。これは、**Workloads → Secrets → Create → From YAML** で実行できます。

詳細は、「オブジェクトストレージのセットアップ」を参照してください。

Amazon S3 および MinIO ストレージのシークレット例

```
apiVersion: v1
kind: Secret
metadata:
  name: minio-test
stringData:
  endpoint: http://minio.minio.svc:9000
  bucket: tempo
  access_key_id: tempo
  access_key_secret: <secret>
type: Opaque
```

3. **TempoMonolithic** インスタンスを作成します。

注記

同じクラスター上の別々のプロジェクトに複数の **TempoMonolithic** インスタンスを作成できます。

a. **Operators → Installed Operators** に移動します。

b. **TempoMonolithic → Create TempoMonolithic → YAML view** を選択します。

c. **YAML view** で、**TempoMonolithic** カスタムリソース (CR) をカスタマイズします。

TempoMonolithic CR の例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic (1)
metadata:
  name: <metadata_name>
  namespace: <permitted_project_of_tempomonolithic_instance> (2)
spec: (3)
  storage: (4)
    traces:
      backend: <supported_storage_type> (5)
      size: <value>Gi (6)
      s3: (7)
```

```
      secret: <secret_name> 8
    tls: 9
      enabled: true
      caName: <ca_certificate_configmap_name> 10
  jaegerui:
    enabled: true 11
    route:
      enabled: true 12
  resources: 13
    total:
      limits:
        memory: <value>Gi
        cpu: <value>m
  multitenancy:
    enabled: true
    mode: openshift
    authentication: 14
      - tenantName: dev 15
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" 16
      - tenantName: prod
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
```

1. この CR は、OTLP プロトコルでトレースの取り込みを行う **TempoMonolithic** デプロイメントを作成します。
2. **TempoMonolithic** デプロイメント用に選択したプロジェクト。**openshift-** 接頭辞で始まるプロジェクト名は許可されません。
3. Red Hat は、Red Hat OpenShift Distributed Tracing Platform ドキュメントに記載されているカスタムリソースオプションのみをサポートしています。
4. トレースを保存するためのストレージを指定します。
5. トレースを保存するストレージのタイプ (インメモリーストレージ、永続ボリューム、またはオブジェクトストレージ)。永続ボリュームの値は **pv** です。オブジェクトストレージの値は、使用するオブジェクトストアのタイプに応じて、**s3**、**gcs**、または **azure** が受け入れられます。デフォルト値は、**tmpfs** インメモリーストレージの **memory** です。これは、Pod がシャットダウンするとデータが保持されないため、開発、テスト、デモ、および概念検証用の環境にのみ適しています。
6. メモリーサイズ: インメモリーストレージの場合、これは **tmpfs** ボリュームのサイズを意味します。デフォルトは **2Gi** です。永続ボリュームの場合、これは永続ボリューム要求のサイズを意味します。デフォルトは **10Gi** です。オブジェクトストレージの場合、これは Tempo Write-Ahead Logging (WAL) の永続ボリューム要求のサイズを意味し、デフォルトは **10Gi** です。
7. オプション: オブジェクトストレージの場合、オブジェクトストレージのタイプ。使用するオブジェクトストアのタイプに応じて、**s3**、**gcs**、および **azure** が値として受け入れられます。
8. オプション: オブジェクトストレージの場合、ストレージシークレットの **metadata** 内の **name** の値。ストレージシークレットは、**TempoMonolithic** インスタンスと同じ namespace にあり、「表 1.必要なシークレットパラメーター」(「オブジェクトストレージのセットアップ」セクションを参照) で指定えているフィールドを含んでいる

必要があります。

9 オプション:

10 オプション: CA 証明書を含む **ConfigMap** オブジェクトの名前。

11 **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search** のルート経由で、データを視覚化する Jaeger UI を公開します。

12 Jaeger UI のルートの作成を有効にします。

13 任意。

14 テナントをリスト表示します。

15 テナント名。**X-Scope-OrgId** HTTP ヘッダーの値として使用されます。

16 テナントの一意の識別子。**TempoMonolithic** デプロイメントのライフサイクル全体を通じて一意である必要があります。この ID は、オブジェクトストレージ内のオブジェクトの接頭辞として追加されます。UUID または **tempoName** フィールドの値を再利用できます。

d. **Create** を選択します。

検証

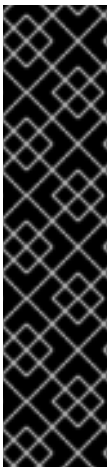
1. **Project:** ドロップダウンリストを使用して、**TempoMonolithic** インスタンスのプロジェクトを選択します。
2. **Operator → Installed Operator** に移動して、**TempoMonolithic** インスタンスの **Status** が **Condition: Ready** であることを確認します。
3. **Workloads → Pod** に移動して、**TempoMonolithic** インスタンスの Pod が実行中であることを確認します。
4. Jaeger UI にアクセスします。
   a. **Networking → Routes** に移動し、**Ctrl+F** を押して **jaegerui** を検索します。

注記

Jaeger UI は、**tempo-<metadata_name_of_TempoMonolithic_CR>-jaegerui** ルートを使用します。

   b. **Location** 列で URL を開き、Jaeger UI にアクセスします。
5. **TempoMonolithic** インスタンスの Pod の準備ができたら、クラスター内の **tempo-<metadata_name_of_TempoMonolithic_CR>:4317** (OTLP/gRPC) および **tempo-<metadata_name_of_TempoMonolithic_CR>:4318** (OTLP/HTTP) エンドポイントにトレースを送信できます。
   Tempo API は、クラスター内の **tempo-<metadata_name_of_TempoMonolithic_CR>:3200** エンドポイントで利用できます。

3.5.2. CLI を使用した TempoMonolithic インスタンスのインストール



重要

TempoMonolithic インスタンスは、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

コマンドラインから **TempoMonolithic** インスタンスをインストールできます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者によるアクティブな OpenShift CLI (**oc**) セッション。

  ヒント

  - OpenShift CLI (**oc**) のバージョンが最新であり、OpenShift Container Platform バージョンと一致していることを確認してください。
  - **oc login** コマンドを実行します。

    ```
    $ oc login --username=<your_username>
    ```

- 1 つ以上のテナントを定義し、読み取りおよび書き込み権限を設定した。詳細は、「テナントの読み取り権限の設定」および「テナントの書き込み権限の設定」を参照してください。

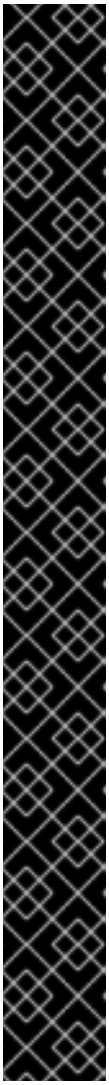
手順

1. 次のコマンドを実行して、後続のステップで作成する **TempoMonolithic** インスタンス用に、許可される任意のプロジェクトを作成します。

   ```
   $ oc apply -f - << EOF
   apiVersion: project.openshift.io/v1
   kind: Project
   metadata:
     name: <permitted_project_of_tempomonolithic_instance> 1
   EOF
   ```

   1 **openshift-** 接頭辞で始まるプロジェクト名は許可されません。

2. トレースの保存に使用するサポート対象のストレージのタイプ (インメモリーストレージ、永続ボリューム、オブジェクトストレージ) を決定します。



重要

オブジェクトストレージは、Distributed Tracing Platform には含まれていません。そのため、サポートされているプロバイダー (Red Hat OpenShift Data Foundation、MinIO、Amazon S3、Azure Blob Storage、または Google Cloud Storage) によるオブジェクトストアを設定する必要があります。

また、オブジェクトストレージを選択するには、**TempoMonolithic** インスタンス用に作成したプロジェクトにオブジェクトストレージバケットのシークレットを作成する必要があります。これを行うには、次のコマンドを実行します。

```
$ oc apply -f - << EOF
<object_storage_secret>
EOF
```

詳細は、「オブジェクトストレージのセットアップ」を参照してください。

Amazon S3 および MinIO ストレージのシークレット例

```
apiVersion: v1
kind: Secret
metadata:
  name: minio-test
stringData:
  endpoint: http://minio.minio.svc:9000
  bucket: tempo
  access_key_id: tempo
  access_key_secret: <secret>
type: Opaque
```

3. **TempoMonolithic** インスタンス用に作成したプロジェクト内に TempoMonolithic インスタンスを作成します。

ヒント

同じクラスター上の別々のプロジェクトに複数の **TempoMonolithic** インスタンスを作成できます。

a. **TempoMonolithic** カスタムリソース (CR) をカスタマイズします。

TempoMonolithic CR の例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic (1)
metadata:
  name: <metadata_name>
  namespace: <permitted_project_of_tempomonolithic_instance> (2)
spec: (3)
  storage: (4)
    traces:
      backend: <supported_storage_type> (5)
      size: <value>Gi (6)
```

```
    s3: 7
      secret: <secret_name> 8
  tls: 9
    enabled: true
    caName: <ca_certificate_configmap_name> 10
jaegerui:
  enabled: true 11
  route:
    enabled: true 12
resources: 13
  total:
    limits:
      memory: <value>Gi
      cpu: <value>m
multitenancy:
  enabled: true
  mode: openshift
  authentication: 14
    - tenantName: dev 15
      tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" 16
    - tenantName: prod
      tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
```

1. この CR は、OTLP プロトコルでトレースの取り込みを行う **TempoMonolithic** デプロイメントを作成します。

2. **TempoMonolithic** デプロイメント用に選択したプロジェクト。**openshift-** 接頭辞で始まるプロジェクト名は許可されません。

3. Red Hat は、Red Hat OpenShift Distributed Tracing Platform ドキュメントに記載されているカスタムリソースオプションのみをサポートしています。

4. トレースを保存するためのストレージを指定します。

5. トレースを保存するストレージのタイプ (インメモリーストレージ、永続ボリューム、またはオブジェクトストレージ)。永続ボリュームの値は **pv** です。オブジェクトストレージの値は、使用するオブジェクトストアのタイプに応じて、**s3**、**gcs**、または **azure** が受け入れられます。デフォルト値は、**tmpfs** インメモリーストレージの **memory** です。これは、Pod がシャットダウンするとデータが保持されないため、開発、テスト、デモ、および概念検証用の環境にのみ適しています。

6. メモリーサイズ: インメモリーストレージの場合、これは **tmpfs** ボリュームのサイズを意味します。デフォルトは **2Gi** です。永続ボリュームの場合、これは永続ボリューム要求のサイズを意味します。デフォルトは **10Gi** です。オブジェクトストレージの場合、これは Tempo Write-Ahead Logging (WAL) の永続ボリューム要求のサイズを意味し、デフォルトは **10Gi** です。

7. オプション: オブジェクトストレージの場合、オブジェクトストレージのタイプ。使用するオブジェクトストアのタイプに応じて、**s3**、**gcs**、および **azure** が値として受け入れられます。

8. オプション: オブジェクトストレージの場合、ストレージシークレットの **metadata** 内の **name** の値。ストレージシークレットは、**TempoMonolithic** インスタンスと同じ namespace にあり、「表 1.必要なシークレットパラメーター」(「オブジェクトスト

レージのセットアップ」セクションを参照) で指定えているフィールドを含んでいる必要があります。

9 オプション:

10 オプション: CA 証明書を含む **ConfigMap** オブジェクトの名前。

11 **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search** のルート経由で、データを視覚化する Jaeger UI を公開します。

12 Jaeger UI のルートの作成を有効にします。

13 任意。

14 テナントをリスト表示します。

15 テナント名。**X-Scope-OrgId** HTTP ヘッダーの値として使用されます。

16 テナントの一意の識別子。**TempoMonolithic** デプロイメントのライフサイクル全体を通じて一意である必要があります。この ID は、オブジェクトストレージ内のオブジェクトの接頭辞として追加されます。UUID または **tempoName** フィールドの値を再利用できます。

b. 次のコマンドを実行して、カスタマイズされた CR を適用します。

```
$ oc apply -f - << EOF
<tempomonolithic_cr>
EOF
```

検証

1. 次のコマンドを実行して、すべての **TempoMonolithic components** の **status** が **Running** であり、**conditions** が **type: Ready** であることを確認します。

```
$ oc get tempomonolithic.tempo.grafana.com <metadata_name_of_tempomonolithic_cr> -o yaml
```

2. 次のコマンドを実行して、**TempoMonolithic** インスタンスの Pod が実行中であることを確認します。

```
$ oc get pods
```

3. Jaeger UI にアクセスします。

a. 次のコマンドを実行して、**tempo-<metadata_name_of_tempomonolithic_cr>-jaegerui** ルートのルート詳細をクエリーします。

```
$ oc get route
```

b. Web ブラウザーで **https://<route_from_previous_step>** を開きます。

4. **TempoMonolithic** インスタンスの Pod の準備ができたら、クラスター内の **tempo-<metadata_name_of_tempomonolithic_cr>:4317** (OTLP/gRPC) および **tempo-**

<metadata_name_of_tempomonolithic_cr>:4318 (OTLP/HTTP) エンドポイントにトレースを送信できます。
Tempo API は、クラスター内の **tempo-<metadata_name_of_tempomonolithic_cr>:3200** エンドポイントで利用できます。

3.6. 関連情報

- クラスター管理者の作成
- OperatorHub.io
- Web コンソールへのアクセス
- Web コンソールを使用した OperatorHub からのインストール
- インストールされた Operator からのアプリケーションの作成
- OpenShift CLI の使用を開始する

第4章 DISTRIBUTED TRACING PLATFORM の設定

Tempo Operator は、Distributed Tracing Platform のリソースを作成およびデプロイするためのアーキテクチャーと設定を定義したカスタムリソース定義 (CRD) ファイルを使用します。デフォルト設定をインストールすることも、ファイルを変更することもできます。

4.1. バックエンドストレージの設定

バックエンドストレージの設定は、永続ストレージについて および選択したストレージオプションに関連する設定セクションを参照してください。

4.2. TEMPOSTACK 設定パラメーターの概要

TempoStack カスタムリソース (CR) は、Distributed Tracing Platform のリソースを作成するためのアーキテクチャーと設定を定義したものです。これらのパラメーターを変更して、実装をビジネスニーズに合わせてカスタマイズできます。

TempoStack CR の例

```
apiVersion: tempo.grafana.com/v1alpha1 1
kind: TempoStack 2
metadata: 3
  name: <name> 4
spec: 5
  storage: {} 6
  resources: {} 7
  replicationFactor: 1 8
  retention: 9
    global:
      traces: 48h
    perTenant: {}
  template:
      distributor: {} 10
      ingester: {} 11
      compactor: {} 12
      querier: {} 13
      queryFrontend: {} 14
      gateway: {} 15
  limits: 16
    global:
      ingestion: {} 17
      query: {} 18
  observability: 19
    grafana: {}
    metrics: {}
    tracing: {}
  search: {} 20
  managementState: managed 21
```

1 オブジェクトの作成時に使用する API バージョン。

2 作成する Kubernetes オブジェクトの種類を定義します。

3 **name** の文字列、**UID**、オプションの **namespace** などのオブジェクトを一意に識別するデータ。OpenShift Container Platform は **UID** を自動的に生成し、オブジェクトが作成されるプロジェクトの名前で **namespace** を完了します。

4 TempoStack インスタンスの名前。

5 TempoStack インスタンスのすべての設定パラメーターが含まれます。すべての Tempo コンポーネントに共通の定義が必要な場合は、**spec** セクションで定義します。定義が個々のコンポーネントに関連している場合は、**spec.template.<component>** セクションに配置します。

6 ストレージはインスタンスのデプロイメント時に指定されます。インスタンスのストレージオプションの詳細は、インストールページを参照してください。

7 Tempo コンテナーのコンピュートリソースを定義します。

8 スパンを受け入れる前にディストリビューターからのデータを確認する必要があるインジェスターの数を表す整数値。

9 トレースの保持に関する設定オプション。デフォルト値は **48h** です。

10 Tempo **distributor** コンポーネントの設定オプション。

11 Tempo **ingester** コンポーネントの設定オプション。

12 Tempo **compactor** コンポーネントの設定オプション。

13 Tempo **querier** コンポーネントの設定オプション。

14 Tempo **query-frontend** コンポーネントの設定オプション。

15 Tempo **gateway** コンポーネントの設定オプション。

16 取り込みとクエリーのレートを制限します。

17 取り込みの流量制御を定義します。

18 クエリーの流量制御を定義します。

19 テレメトリーデータを処理するためのオペランドを設定します。

20 検索機能を設定します。

21 この CR が Operator によって管理されるかどうかを定義します。デフォルト値は **managed** です。

表4.1 TempoStack CR のパラメーター

パラメーター	説明	値	デフォルト値
apiVersion:	オブジェクトの作成時に使用する API バージョン。	**tempo.grafana.com/v1alpha1**	**tempo.grafana.com/v1alpha1**

パラメーター	説明	値	デフォルト値
kind:	作成する Kubernetes オブジェクトの種類を定義します。	**tempo**	
metadata:	**name** の文字列、**UID**、オプションの **namespace** などのオブジェクトを一意に識別するデータ。		OpenShift Container Platform は **UID** を自動的に生成し、オブジェクトが作成されるプロジェクトの名前で **namespace** を完了します。
name:	オブジェクトの名前。	TempoStack インスタンスの名前。	**tempo-all-in-one-inmemory**
spec:	作成するオブジェクトの仕様。	TempoStack インスタンスのすべての設定パラメーターが含まれています。すべての Tempo コンポーネントの共通定義が必要な場合、**spec** ノードで定義されます。個々のコンポーネントに関連する定義は、**spec.template.<component>** ノードに置かれます。	該当なし
resources:	TempoStack インスタンスに割り当てられたリソース。		
storageSize:	Ingester PVC のストレージサイズ。		
replicationFactor:	レプリケーション係数の設定。		
retention:	トレースの保持に関する設定オプション。		
storage:	ストレージを定義する設定オプション。		
template.distributor:	Tempo ディストリビューターの設定オプション。		

パラメーター	説明	値	デフォルト値
template.ingester:	Tempo インジェスターの設定オプション。		
template.compactor:	Tempo コンパクターの設定オプション。		
template.querier:	Tempo クエリアーの設定オプション。		
template.queryFrontend:	Tempo クエリーフロントエンドの設定オプション。		
template.gateway:	Tempo ゲートウェイの設定オプション。		

関連情報

- TempoStack インスタンスのインストール
- TempoMonolithic インスタンスのインストール

4.3. クエリー設定オプション

Distributed Tracing Platform の 2 つのコンポーネントであるクエリアーとクエリーフロントエンドがクエリーを管理します。これらのコンポーネントは両方とも設定できます。

クエリアーコンポーネントは、インジェスターまたはバックエンドストレージで要求されたトレース ID を検索します。設定されたパラメーターに応じて、クエリアーコンポーネントはインジェスターの両方にクエリーを実行し、bloom またはインデックスをバックエンドからプルして、オブジェクトストレージ内のブロックを検索できます。クエリアーコンポーネントは **GET /querier/api/traces/<trace_id>** で HTTP エンドポイントを公開します。ただし、このエンドポイントを直接使用することは想定されていません。クエリーはクエリーフロントエンドに送信する必要があります。

表4.2 クエリアーコンポーネントの設定パラメーター

パラメーター	説明	値
nodeSelector	ノード選択制約の単純な形式。	type: object
replicas	コンポーネントに対して作成されるレプリカの数。	type: integer; format: int32
toleration	コンポーネント固有の Pod 容認。	type: array

クエリーフロントエンドコンポーネントは、受信クエリーの検索スペースをシャーディングする役割を

持ちます。クエリーフロントエンドは、単純な HTTP エンドポイント (**GET /api/traces/<trace_id>**) を介してトレースを公開します。内部的には、クエリーフロントエンドコンポーネントは **blockID** スペースを設定可能な数のシャードに分割し、これらのリクエストをキューに登録します。クエリアーコンポーネントは、ストリーミング gRPC 接続を介してクエリーフロントエンドコンポーネントに接続し、これらのシャードクエリーを処理します。

表4.3 クエリーフロントエンドコンポーネントの設定パラメーター

パラメーター	説明	値
component	クエリーフロントエンドコンポーネントの設定。	type: object
component.nodeSelector	ノード選択制約の単純な形式。	type: object
component.replicas	クエリーフロントエンドコンポーネントに対して作成されるレプリカの数。	type: integer; format: int32
component.tolerations	クエリーフロントエンドコンポーネントに固有の Pod 容認。	type: array
jaegerQuery	Jaeger Query コンポーネントに固有のオプション。	type: object
jaegerQuery.enabled	**enabled** にすると、Jaeger Query コンポーネント **jaegerQuery** が作成されます。	type: boolean
jaegerQuery.ingress	Jaeger Query Ingress のオプション。	type: object
jaegerQuery.ingress.annotations	Ingress オブジェクトのアノテーション。	type: object
jaegerQuery.ingress.host	Ingress オブジェクトのホスト名。	type: string
jaegerQuery.ingress.ingressClassName	IngressClass クラスターリソースの名前。この Ingress リソースを提供する Ingress コントローラーを定義します。	type: string
jaegerQuery.ingress.route	OpenShift ルートのオプション。	type: object
jaegerQuery.ingress.route.termination	終端タイプ。デフォルトは **edge** です。	type: string (enum: insecure, edge, passthrough, reencrypt)

パラメーター	説明	値
jaegerQuery.ingress.type	Jaeger Query UI の Ingress のタイプ。サポートされているタイプは、**ingress**、**route**、および **none** です。	type: string (enum: ingress, route)
jaegerQuery.monitorTab	Monitor タブの設定。	type: object
jaegerQuery.monitorTab.enabled	Jaeger コンソールの Monitor タブを有効にします。**PrometheusEndpoint** を設定する必要があります。	type: boolean
jaegerQuery.monitorTab.prometheusEndpoint	スパンのレート、エラー、および期間 (RED) メトリクスを含む Prometheus インスタンスへのエンドポイント。たとえば、**https://thanos-querier.openshift-monitoring.svc.cluster.local:9092** です。	type: string

TempoStack CR のクエリーフロントエンドコンポーネントの設定例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
spec:
  storage:
    secret:
      name: minio
      type: s3
  storageSize: 200M
  resources:
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  template:
    queryFrontend:
      jaegerQuery:
        enabled: true
        ingress:
          route:
            termination: edge
          type: route
```

関連情報

関連情報

- taint および toleration について

4.4. UI の設定

Cluster Observability Operator (COO) の分散トレーシング UI プラグインを、Red Hat OpenShift Distributed Tracing Platform のユーザーインターフェイス (UI) として使用できます。分散トレーシング UI プラグインのインストールと使用の詳細は、**Cluster Observability Operator** の「分散トレーシング UI プラグイン」を参照してください。

関連情報

- 分散トレーシング UI プラグイン

4.5. JAEGER UI の MONITOR タブの設定

リクエストのレート、エラー、および期間 (RED) メトリクスをトレースから抽出して、OpenShift Container Platform Web コンソールの **Monitor** タブの Jaeger コンソールで視覚化できます。メトリクスは、Prometheus によってコレクターからスクレイピングされた OpenTelemetry コレクター内のスパンから導出されます。Prometheus は、ユーザーワークロードモニタリングスタックにデプロイできます。Jaeger UI は、Prometheus エンドポイントからこれらのメトリクスをクエリーし、可視化します。

前提条件

- Distributed Tracing Platform の権限とテナントを設定した。詳細は、「権限とテナントの設定」を参照してください。

手順

1. OpenTelemetry Collector の **OpenTelemetryCollector** カスタムリソースで、Spanmetrics コネクター (**spanmetrics**) を有効にします。このコネクターは、トレースからメトリクスを導出し、そのメトリクスを Prometheus 形式でエクスポートします。

   スパン RED 用の OpenTelemetryCollector カスタムリソースの例

```
apiVersion: opentelemetry.io/v1beta1
kind: OpenTelemetryCollector
metadata:
  name: otel
spec:
  mode: deployment
  observability:
    metrics:
      enableMetrics: true (1)
  config: |
    connectors:
      spanmetrics: (2)
        metrics_flush_interval: 15s

    receivers:
      otlp: (3)
        protocols:
```

```
        grpc:
        http:

    exporters:
      prometheus: 4
        endpoint: 0.0.0.0:8889
        add_metric_suffixes: false
        resource_to_telemetry_conversion:
          enabled: true 5

      otlp:
        auth:
          authenticator: bearertokenauth
        endpoint: tempo-redmetrics-gateway.mynamespace.svc.cluster.local:8090
        headers:
          X-Scope-OrgID: dev
        tls:
          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt
          insecure: false

    extensions:
      bearertokenauth:
        filename: /var/run/secrets/kubernetes.io/serviceaccount/token

    service:
      extensions:
      - bearertokenauth
      pipelines:
        traces:
          receivers: [otlp]
          exporters: [otlp, spanmetrics] 6
        metrics:
          receivers: [spanmetrics] 7
          exporters: [prometheus]

# ...
```

1. **ServiceMonitor** カスタムリソースを作成して、Prometheus エクスポーターの収集を有効にします。
2. Spanmetrics コネクターはトレースを受信し、メトリクスをエクスポートします。
3. OpenTelemetry プロトコルのスパンを受信する OTLP レシーバー。
4. Prometheus エクスポーターは、Prometheus 形式でメトリクスをエクスポートするために使用されます。
5. リソース属性はデフォルトでドロップされます。
6. Spanmetrics コネクターは、トレースパイプラインのエクスポーターとして設定されています。
7. Spanmetrics コネクターは、メトリクスパイプラインのレシーバーとして設定されています。

2. **TempoStack** カスタムリソースで、**Monitor** タブを有効にし、ユーザー定義のモニタリングスタックからデータを照会するように、Prometheus エンドポイントを Thanos Querier サービスに設定します。

Monitor タブが有効な TempoStack カスタムリソースの例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: redmetrics
spec:
  storage:
    secret:
      name: minio-test
      type: s3
  storageSize: 1Gi
  tenants:
    mode: openshift
    authentication:
      - tenantName: dev
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa"
  template:
    gateway:
      enabled: true
    queryFrontend:
      jaegerQuery:
        monitorTab:
          enabled: true (1)
          prometheusEndpoint: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
(2)
          redMetricsNamespace: "" (3)

# ...
```

(1) Jaeger コンソールの監視タブを有効にします。

(2) ユーザーワークロードモニタリングからの Thanos Querier のサービス名。

(3) オプション: Jaeger クエリーが Prometheus メトリクスを取得するメトリクス namespace。この行は、0.109.0 より前のバージョンの OpenTelemetry Collector を使用している場合にのみ含めてください。OpenTelemetry Collector バージョン 0.109.0 以降を使用している場合は、この行を省略します。

3. オプション: **spanmetrics** コネクターによって生成されるスパン RED メトリクスを、アラートルールで使用します。たとえば、このコネクターは、サービスの速度低下に関するアラートの場合や、サービスレベル目標 (SLO) を定義する場合のために、**duration_bucket** ヒストグラムと **calls** カウンターメトリクスを作成します。これらのメトリクスには、サービス、API 名、操作タイプ、その他の属性を識別するラベルが付いています。

表4.4 spanmetrics コネクターで作成されるメトリクスのラベル

ラベル	説明	値
service_name	**otel_service_name** 環境変数によって設定されるサービス名。	**frontend**
span_name	操作の名前。	• / • **/customer**
span_kind	サーバー、クライアント、メッセージング、または内部操作を識別します。	• **SPAN_KIND_SERVER** • **SPAN_KIND_CLIENT** • **SPAN_KIND_PRODUCER** • **SPAN_KIND_CONSUMER** • **SPAN_KIND_INTERNAL**

フロントエンドサービスで 2000 ミリ秒以内に 95% の要求が処理されない場合の SLO のアラートルールを定義する PrometheusRule カスタムリソースの例

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: span-red
spec:
  groups:
  - name: server-side-latency
    rules:
    - alert: SpanREDFrontendAPIRequestLatency
      expr: histogram_quantile(0.95, sum(rate(duration_bucket{service_name="frontend",
span_kind="SPAN_KIND_SERVER"}[5m])) by (le, service_name, span_name)) > 2000 (1)
      labels:
        severity: Warning
      annotations:
        summary: "High request latency on {{$labels.service_name}} and
{{$labels.span_name}}"
        description: "{{$labels.instance}} has 95th request latency above 2s (current value:
{{$value}}s)"
```

(1) 95% のフロントエンドサーバーの応答時間値が 2000 ミリ秒未満であるかどうかを確認する式。時間範囲 (**[5m]**) が収集間隔の 4 倍以上で、メトリクスの変化に対応できる十分な長さである必要があります。

関連情報

- 権限とテナントの設定

4.6. レシーバーの TLS の設定

TempoStack または TempoMonolithic インスタンスのカスタムリソースで、ユーザーが指定する証明書または OpenShift のサービス提供証明書を使用して、レシーバーの TLS を設定できます。

4.6.1. TempoStack インスタンス用のレシーバーの TLS 設定

シークレットの TLS 証明書を指定することも、OpenShift Container Platform によって生成されるサービス提供証明書を使用することもできます。

- シークレットの TLS 証明書を指定するには、**TempoStack** カスタムリソースでそれを設定します。

注記

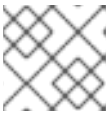
この機能は、有効な Tempo Gateway ではサポートされていません。

レシーバーの TLS と、ユーザーが指定するシークレットの証明書の使用

```
apiVersion: tempo.grafana.com/v1alpha1
kind:  TempoStack
# ...
spec:
# ...
  template:
    distributor:
      tls:
        enabled: true (1)
        certName: <tls_secret> (2)
        caName: <ca_name> (3)
# ...
```

1. Tempo Distributor で TLS が有効になります。
2. 事前に適用する **tls.key** 鍵と **tls.crt** 証明書を含むシークレット。
3. オプション: 相互 TLS 認証 (mTLS) を有効にするための config map 内の CA。

- または、OpenShift Container Platform によって生成されたサービス提供証明書を使用することもできます。



注記

この機能では、相互 TLS 認証 (mTLS) はサポートされていません。

レシーバーの TLS と、OpenShift Container Platform によって生成されるサービス提供証明書の使用

```
apiVersion: tempo.grafana.com/v1alpha1
kind:  TempoStack
# ...
spec:
# ...
  template:
    distributor:
      tls:
        enabled: true (1)
# ...
```

(1) Tempo Distributor の TLS に十分な設定。

関連情報

- サービス提供証明書について
- サービス CA 証明書

4.6.2. TempoMonolithic インスタンス用のレシーバーの TLS 設定

シークレットの TLS 証明書を指定することも、OpenShift Container Platform によって生成されるサービス提供証明書を使用することもできます。

- シークレットの TLS 証明書を指定するには、**TempoMonolithic** カスタムリソースでそれを設定します。

注記

この機能は、有効な Tempo Gateway ではサポートされていません。

レシーバーの TLS と、ユーザーが指定するシークレットの証明書の使用

```
apiVersion: tempo.grafana.com/v1alpha1
kind:  TempoMonolithic
# ...
  spec:
# ...
  ingestion:
    otlp:
      grpc:
        tls:
          enabled: true (1)
          certName: <tls_secret> (2)
          caName: <ca_name> (3)
# ...
```

(1) Tempo Distributor で TLS が有効になります。

(2) 事前に適用する **tls.key** 鍵と **tls.crt** 証明書を含むシークレット。

(3) オプション: 相互 TLS 認証 (mTLS) を有効にするための config map 内の CA。

- または、OpenShift Container Platform によって生成されたサービス提供証明書を使用することもできます。

注記

この機能では、相互 TLS 認証 (mTLS) はサポートされていません。

レシーバーの TLS と、OpenShift Container Platform によって生成されるサービス提供証明書の使用

```
apiVersion: tempo.grafana.com/v1alpha1
kind:  TempoMonolithic
# ...
  spec:
# ...
  ingestion:
    otlp:
      grpc:
        tls:
          enabled: true
      http:
        tls:
          enabled: true (1)
# ...
```

(1) Tempo Distributor の TLS の最小設定。

関連情報

- サービス提供証明書について
- サービス CA 証明書

4.7. クエリー RBAC の設定

管理者は、クエリーのロールベースアクセス制御 (RBAC) を設定して、ユーザーに権限を付与した namespace ごとにユーザーの SPAN 属性をフィルタリングできます。

注記

クエリー RBAC を有効にすると、ユーザーは引き続きすべての namepsace からのトレースにアクセスできるようになり、**service.name** 属性と **k8s.namespace.name** 属性もすべてのユーザーに表示されます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者によるアクティブな OpenShift CLI (**oc**) セッション。

ヒント

- OpenShift CLI (**oc**) のバージョンが最新であり、OpenShift Container Platform バージョンと一致していることを確認してください。
- **oc login** を実行します。

```
$ oc login --username=<your_username>
```

手順

1. **TempoStack** カスタムリソース (CR) でマルチテナントを有効にし、RBAC のクエリーを実行します。以下はその例です。

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
  namespace: chainsaw-multitenancy
spec:
  storage:
    secret:
      name: minio
      type: s3
  storageSize: 1Gi
  resources:
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  tenants:
    mode: openshift
    authentication:
      - tenantName: dev
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
  template:
    gateway:
      enabled: true (1)
      rbac:
        enabled: true (2)
    queryFrontend:
      jaegerQuery:
        enabled: false (3)
```

(1) 常に **true** に設定されます。

(2) 常に **true** に設定されます。

(3) 常に **false** に設定されます。

2. クラスターロールとクラスターロールバインディングを作成して、**TempoStack** CR で指定したテナントにアクセスするための権限をターゲットユーザーに付与します。以下はその例です。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tempo-dev-read
rules:
- apiGroups: [tempo.grafana.com]
  resources: [dev] 1
  resourceNames: [traces]
  verbs: [get]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tempo-dev-read
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tempo-dev-read
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated 2
```

1 **TempoStack** CR 内のテナント名。

2 認証されたすべての OpenShift ユーザーを意味します。

3. ターゲットユーザーにプロジェクトの属性を読み取る権限を付与します。これを行うには、次のコマンドを実行します。

```
$ oc adm policy add-role-to-user view <username> -n <project>
```

4.8. TAINT および TOLERATION の使用

専用ノードで TempoStack Pod をスケジュールするには、OpenShift 4 で nodeSelector と tolerations を使用してインフラノードにさまざまな TempoStack コンポーネントをデプロイする方法 を参照してください。

4.9. 監視とアラートの設定

Tempo Operator は、distributor や ingester などの各 TempoStack コンポーネントのモニタリングとアラートをサポートし、Operator 自体に関するアップグレードおよび運用のメトリクスを公開します。

4.9.1. TempoStack のメトリクスとアラートの設定

TempoStack インスタンスのメトリクスとアラートを有効にできます。

前提条件

- ユーザー定義プロジェクトのモニタリングがクラスターで有効にされている。

手順

1. TempoStack インスタンスのメトリクスを有効にするには、**spec.observability.metrics.createServiceMonitors** フィールドを **true** に設定します。

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
spec:
  observability:
    metrics:
      createServiceMonitors: true
```

2. TempoStack インスタンスのアラートを有効にするには、**spec.observability.metrics.createPrometheusRules** フィールドを **true** に設定します。

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
spec:
  observability:
    metrics:
      createPrometheusRules: true
```

検証

Web コンソールの **Administrator** ビューを使用して、正常に設定されたことを確認できます。

1. **Observe → Targets** に移動して **Source: User** でフィルタリングし、**tempo-<instance_name>-<component>** 形式の **ServiceMonitors** のステータスが **Up** であることを確認します。

2. アラートが正しく設定されていることを確認するには、**Observe → Alerting → Alerting rules** に移動して **Source: User** でフィルタリングし、TempoStack インスタンスコンポーネントの **Alert rules** が利用可能であることを確認します。

関連情報

- ユーザー定義プロジェクトのモニタリングの有効化

4.9.2. Tempo Operator のメトリクスとアラートの設定

Web コンソールから Tempo Operator をインストールする場合は、**Enable Operator recommended cluster monitoring on this Namespace** チェックボックスを選択すると、Tempo Operator のメトリクスおよびアラートを作成できます。

インストール時にチェックボックスを選択しなかった場合も、Tempo Operator をインストールした後にメトリクスとアラートを手動で有効にできます。

手順

- Tempo Operator がインストールされているプロジェクトに **openshift.io/cluster-monitoring: "true"** ラベルを追加します。デフォルトは **openshift-tempo-operator** です。

検証

Web コンソールの **Administrator** ビューを使用して、正常に設定されたことを確認できます。

1. **Observe → Targets** に移動して **Source: Platform** でフィルタリングし、**tempo-operator** を検索します。その場合は、ステータスは **Up** でなければなりません。

2. アラートが正しく設定されていることを確認するには、**Observe → Alerting → Alerting rules** に移動して **Source: Platform** でフィルタリングし、**Tempo Operator** の **Alert rules** を見つけます。

第5章 DISTRIBUTED TRACING PLATFORM のトラブルシューティング

さまざまなトラブルシューティング方法を使用して、TempoStack または TempoMonolithic インスタンスの問題を診断して修正できます。

5.1. コマンドラインからの診断データの収集

サポートケースを送信するときは、クラスターに関する診断情報を Red Hat サポートに含めると役立ちます。**oc adm must-gather** ツールを使用すると、**TempoStack** や **TempoMonolithic** などのさまざまなタイプのリソースや、**Deployment**、**Pod**、**ConfigMap** などの作成されたリソースの診断データを収集できます。**oc adm must-gather** ツールは、このデータを収集する新しい Pod を作成します。

手順

- 収集したデータを保存するディレクトリーから、**oc adm must-gather** コマンドを実行してデータを収集します。

```
$ oc adm must-gather --image=ghcr.io/grafana/tempo-operator/must-gather -- \
/usr/bin/must-gather --operator-namespace <operator_namespace> 1
```

1 Operator がインストールされるデフォルトの namespace は **openshift-tempo-operator** です。

検証

- 新しいディレクトリーが作成され、収集されたデータが含まれていることを確認します。

第6章 アップグレード

バージョンアップグレードの場合、Tempo Operator は Operator Lifecycle Manager (OLM) を使用します。これは、クラスター内の Operator のインストール、アップグレード、ロールベースのアクセス制御 (RBAC) を制御します。

OLM は、デフォルトで OpenShift Container Platform で実行されます。OLM は利用可能な Operator のクエリーやインストールされた Operator のアップグレードを実行します。

Tempo Operator が新しいバージョンにアップグレードされると、その Operator が管理する実行中の TempoStack インスタンスをスキャンし、新しい Operator バージョンに対応するバージョンにアップグレードします。

6.1. 関連情報

- Operator Lifecycle Manager の概念およびリソース
- インストール済み Operators の更新

第7章 DISTRIBUTED TRACING PLATFORM の削除

OpenShift Container Platform クラスターから Red Hat OpenShift Distributed Tracing Platform を削除する手順は次のとおりです。

1. すべての Distributed Tracing Platform Pod をシャットダウンします。
2. TempoStack インスタンスを削除します。
3. Tempo Operator を削除します。

7.1. WEB コンソールを使用して削除する

Web コンソールの **Administrator** ビューで、TempoStack インスタンスを削除できます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者として、OpenShift Container Platform Web コンソールにログインしている。
- Red Hat OpenShift Dedicated の場合、**dedicated-admin** ロールを持つアカウントを使用してログインしている。

手順

1. **Operators → Installed Operators → Tempo Operator → TempoStack** に移動します。
2. TempoStack インスタンスを削除するには、 → **Delete TempoStack → Delete** を選択します。
3. オプション: Tempo Operator を削除します。

7.2. CLI を使用して削除する

コマンドラインで TempoStack インスタンスを削除できます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者によるアクティブな OpenShift CLI (**oc**) セッション。

  ヒント

  - OpenShift CLI (**oc**) のバージョンが最新であり、OpenShift Container Platform バージョンと一致していることを確認してください。
  - **oc login** を実行します。

    ```
    $ oc login --username=<your_username>
    ```

手順

1. 以下のコマンドを実行して、TempoStack インスタンスの名前を取得します。

```
$ oc get deployments -n <project_of_tempostack_instance>
```

2. 以下のコマンドを実行して、TempoStack インスタンスを削除します。

```
$ oc delete tempo <tempostack_instance_name> -n <project_of_tempostack_instance>
```

3. オプション: Tempo Operator を削除します。

検証

1. 以下のコマンドを実行して、出力に TempoStack インスタンスがないことを確認します。ない場合、正常に削除されています。

```
$ oc get deployments -n <project_of_tempostack_instance>
```

7.3. 関連情報

- クラスターからの Operator の削除
- OpenShift CLI の使用を開始