OpenShift Container Platform 4.20

Distributed Tracing

Configuring and using distributed tracing in OpenShift Container Platform

Last Updated: 2026-01-20

OpenShift Container Platform 4.20 Distributed Tracing

Configuring and using distributed tracing in OpenShift Container Platform

Legal Notice

Abstract

Use distributed tracing to store, analyze, and visualize microservices transactions passing through distributed systems in OpenShift Container Platform.

Table of Contents

CHAPTER 1. RELEASE NOTES FOR THE RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM 3.8.1

1.1. ABOUT THIS RELEASE

Distributed Tracing Platform 3.8.1 is provided through the Tempo Operator 0.19.0 and based on the open source Grafana Tempo 2.9.0.

NOTE

Only supported features are documented. Undocumented features are currently unsupported. If you need assistance with a feature, contact Red Hat's support.

1.2. FIXED ISSUES

The Distributed Tracing Platform 3.8.1 patch release includes a fix for the following issue:

CVE-2025-58183

Before this update, malicious tar archives could be used for denial of service by triggering memory overuse by the **tar.Reader** decompressor. With this update, the Go version upgrade to 1.25.3 enforces a maximum limit on the count of sparse region data blocks. As a result, memory usage when processing these archives is bounded and secure.
CVE-2025-58183

The Distributed Tracing Platform 3.8 release fixes the following issues:

Resolved issue with TLS certificates affecting Tempo pods

Before this update, the Tempo pods stopped communicating because internal TLS certificates were renewed. With this update, the Tempo pods automatically restart when certificates are renewed.
TRACING-5622

Tempo query frontend no longer fails to fetch trace JSON

Before this update, clicking on **Trace** in the Jaeger UI and refreshing the page, or accessing **Trace → Trace Timeline → Trace JSON** from the Tempo query frontend, might result in the Tempo query pod failing with an EOF error. With this update, this issue is resolved.
TRACING-5483

NOTE

Some linked Jira tickets are accessible only with Red Hat credentials.

1.3. GETTING SUPPORT

If you experience difficulty with a procedure described in this documentation, or with OpenShift Container Platform in general, visit the Red Hat Customer Portal .

From the Customer Portal, you can:

- Search or browse through the Red Hat Knowledgebase of articles and solutions relating to Red Hat products.
- Submit a support case to Red Hat Support.
- Access other product documentation.

To identify issues with your cluster, you can use Red Hat Lightspeed in OpenShift Cluster Manager. Red Hat Lightspeed provides details about issues and, if available, information on how to solve a problem.

If you have a suggestion for improving this documentation or have found an error, submit a Jira issue for the most relevant documentation component. Please provide specific details, such as the section name and OpenShift Container Platform version.

WARNING

The deprecated Red Hat OpenShift Distributed Tracing Platform (Jaeger) 3.5 was the last release of the Red Hat OpenShift Distributed Tracing Platform (Jaeger) that Red Hat supported.

All support and maintenance for the deprecated Red Hat OpenShift Distributed Tracing Platform (Jaeger) 3.5 ended on November 3, 2025.

If you still use Red Hat OpenShift Distributed Tracing Platform (Jaeger), you must migrate to Red Hat build of OpenTelemetry Operator and Tempo Operator for distributed tracing collection and storage. For more information, see "Migrating" in the Red Hat build of OpenTelemetry documentation, "Installing" in the Red Hat build of OpenTelemetry documentation, and "Installing" in the Red Hat OpenShift Distributed Tracing Platform documentation.

For more information, see the Red Hat Knowledgebase solution Jaeger Deprecation and Removal in OpenShift.

CHAPTER 2. ABOUT THE DISTRIBUTED TRACING PLATFORM

2.1. KEY CONCEPTS IN DISTRIBUTED TRACING

Every time a user takes an action in an application, a request is executed by the architecture that may require dozens of different services to participate to produce a response. Red Hat OpenShift Distributed Tracing Platform lets you perform distributed tracing, which records the path of a request through various microservices that make up an application.

Distributed tracing is a technique that is used to tie the information about different units of work together — usually executed in different processes or hosts — to understand a whole chain of events in a distributed transaction. Developers can visualize call flows in large microservice architectures with distributed tracing. It is valuable for understanding serialization, parallelism, and sources of latency.

Red Hat OpenShift Distributed Tracing Platform records the execution of individual requests across the whole stack of microservices, and presents them as traces. A *trace* is a data/execution path through the system. An end-to-end trace consists of one or more spans.

A *span* represents a logical unit of work in Red Hat OpenShift Distributed Tracing Platform that has an operation name, the start time of the operation, and the duration, as well as potentially tags and logs. Spans may be nested and ordered to model causal relationships.

As a service owner, you can use distributed tracing to instrument your services to gather insights into your service architecture. You can use Red Hat OpenShift Distributed Tracing Platform for monitoring, network profiling, and troubleshooting the interaction between components in modern, cloud-native, microservices-based applications.

With Distributed Tracing Platform, you can perform the following functions:

- Monitor distributed transactions
- Optimize performance and latency
- Perform root cause analysis

You can combine Distributed Tracing Platform with other relevant components of the OpenShift Container Platform:

- Red Hat build of OpenTelemetry for forwarding traces to a TempoStack instance
- Distributed tracing UI plugin of the Cluster Observability Operator (COO)

Additional resources

- Red Hat build of OpenTelemetry
- Distributed tracing UI plugin

2.2. RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM FEATURES

Red Hat OpenShift Distributed Tracing Platform provides the following capabilities:

- Integration with Kiali – When properly configured, you can view Distributed Tracing Platform data from the Kiali console.

- High scalability – The Distributed Tracing Platform back end is designed to have no single points of failure and to scale with the business needs.
- Distributed Context Propagation – Enables you to connect data from different components together to create a complete end-to-end trace.
- Backwards compatibility with Zipkin – Red Hat OpenShift Distributed Tracing Platform has APIs that enable it to be used as a drop-in replacement for Zipkin, but Red Hat is not supporting Zipkin compatibility in this release.

2.3. RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM ARCHITECTURE

Red Hat OpenShift Distributed Tracing Platform is made up of several components that work together to collect, store, and display tracing data.

- **Red Hat OpenShift Distributed Tracing Platform** - This component is based on the open source Grafana Tempo project.
  - **Gateway** – The Gateway handles authentication, authorization, and forwarding of requests to the Distributor or Query front-end service.
  - **Distributor** – The Distributor accepts spans in multiple formats, including Jaeger, OpenTelemetry, and Zipkin. It routes spans to Ingesters by hashing the **traceID** and using a distributed consistent hash ring.
  - **Ingester** – The Ingester batches a trace into blocks, creates bloom filters and indexes, and then flushes it all to the back end.
  - **Query Frontend** – The Query Frontend shards the search space for an incoming query and sends the query to the Queriers. The Query Frontend deployment exposes the Jaeger UI through the Tempo Query sidecar.
  - **Querier** - The Querier is responsible for finding the requested trace ID in either the Ingesters or the back-end storage. Depending on parameters, it can query the Ingesters and pull Bloom indexes from the back end to search blocks in object storage.
  - **Compactor** – The Compactor streams blocks to and from the back-end storage to reduce the total number of blocks.
- **Red Hat build of OpenTelemetry** - This component is based on the open source OpenTelemetry project.
  - **OpenTelemetry Collector** - The OpenTelemetry Collector is a vendor-agnostic way to receive, process, and export telemetry data. The OpenTelemetry Collector supports open-source observability data formats, for example, Jaeger and Prometheus, sending to one or more open-source or commercial back-ends. The Collector is the default location instrumentation libraries export their telemetry data.

CHAPTER 3. INSTALLING THE DISTRIBUTED TRACING PLATFORM

Installing the Distributed Tracing Platform involves the following steps:

1. Installing the Tempo Operator.
2. Setting up a supported object store and creating a secret for the object store credentials.
3. Configuring the permissions and tenants.
4. Depending on your use case, installing your choice of deployment:
   - Microservices-mode **TempoStack** instance
   - Monolithic-mode **TempoMonolithic** instance

3.1. INSTALLING THE TEMPO OPERATOR

You can install the Tempo Operator by using the web console or the command line.

3.1.1. Installing the Tempo Operator by using the web console

You can install the Tempo Operator from the OpenShift Container Platform web console.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a cluster administrator with the **cluster-admin** role.
- For Red Hat OpenShift Dedicated, you must be logged in using an account with the **dedicated-admin** role.
- You have completed setting up the required object storage by a supported provider: Red Hat OpenShift Data Foundation, MinIO, Amazon S3, Azure Blob Storage, Google Cloud Storage. For more information, see "Object storage setup".

WARNING

Object storage is required and not included with the Distributed Tracing Platform. You must choose and set up object storage by a supported provider before installing the Distributed Tracing Platform.

Procedure

1. In the web console, search for **Tempo Operator**.

TIP

In OpenShift Container Platform 4.19 or earlier, go to **Operators → OperatorHub**.

In OpenShift Container Platform 4.20 or later, go to **Ecosystem → Software Catalog**.

2. Select the **Tempo Operator** that is **provided by Red Hat**

IMPORTANT

The following selections are the default presets for this Operator:

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

3. Select the **Enable Operator recommended cluster monitoring on this Namespace**checkbox.
4. Select **Install → Install → View Operator**.

Verification

- In the **Details** tab of the page of the installed Operator, under **ClusterServiceVersion details**, verify that the installation **Status** is **Succeeded**.

3.1.2. Installing the Tempo Operator by using the CLI

You can install the Tempo Operator from the command line.

Prerequisites

- An active OpenShift CLI (**oc**) session by a cluster administrator with the **cluster-admin** role.

  TIP

  - Ensure that your OpenShift CLI (**oc**) version is up to date and matches your OpenShift Container Platform version.
  - Run **oc login**:

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

- You have completed setting up the required object storage by a supported provider: Red Hat OpenShift Data Foundation, MinIO, Amazon S3, Azure Blob Storage, Google Cloud Storage. For more information, see "Object storage setup".

WARNING

Object storage is required and not included with the Distributed Tracing Platform. You must choose and set up object storage by a supported provider before installing the Distributed Tracing Platform.

Procedure

1. Create a project for the Tempo Operator by running the following command:

```
$ 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. Create an Operator group by running the following command:

```
$ 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. Create a subscription by running the following command:

```
$ 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
```

Verification

- Check the Operator status by running the following command:

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

3.2. OBJECT STORAGE SETUP

You can use the following configuration parameters when setting up a supported object storage.

IMPORTANT

Using object storage requires setting up a supported object store and creating a secret for the object store credentials before deploying a **TempoStack** or **TempoMonolithic** instance.

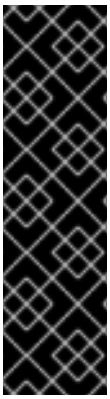
Table 3.1. Required secret parameters

Storage provider	Secret parameters
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	See MinIO Operator. **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>**
IBM Cloud Object Storage (COS)	**bucket: <ibm_bucket_name>** **endpoint: <ibm_bucket_endpoint>** **access_key_id: <ibm_bucket_access_key>** **access_key_secret: <ibm_bucket_secret_key>**

Storage provider	Secret parameters
Amazon S3	**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>**
Amazon S3 with Security Token Service (STS)	**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. Setting up the Amazon S3 storage with the Security Token Service

You can set up the Amazon S3 storage with the Security Token Service (STS) and AWS Command Line Interface (AWS CLI). Optionally, you can also use the Cloud Credential Operator (CCO).



IMPORTANT

Using the Distributed Tracing Platform with the Amazon S3 storage and STS is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Prerequisites

- You have installed the latest version of the AWS CLI.
- If you intend to use the CCO, you have installed and configured the CCO in your cluster.

Procedure

1. Create an AWS S3 bucket.
2. Create the following **trust.json** file for the AWS Identity and Access Management (AWS IAM) policy for the purpose of setting up a trust relationship between the AWS IAM role, which you will create in the next step, and the service account of either the **TempoStack** or **TempoMonolithic** instance:

   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) The OpenID Connect (OIDC) provider that you have configured on the OpenShift Container Platform.

(2) The namespace in which you intend to create either a **TempoStack** or **TempoMonolithic** instance. Replace **<tempo_custom_resource_name>** with the **metadata** name that you define in your **TempoStack** or **TempoMonolithic** custom resource.

TIP

You can also get the value for the OIDC provider by running the following command:

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

3. Create an AWS IAM role by attaching the created **trust.json** policy file. You can do this by running the following command:

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

4. Attach an AWS IAM policy to the created AWS IAM role. You can do this by running the following command:

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

5. If you are not using the CCO, skip this step. If you are using the CCO, configure the cloud provider environment for the Tempo Operator. You can do this by running the following command:

```
$ 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. The name of the Tempo Operator subscription.
2. The namespace of the Tempo Operator.
3. The AWS STS requires adding the **ROLEARN** environment variable to the Tempo Operator subcription. As the **<role_arn>** value, add the Amazon Resource Name (ARN) of the AWS IAM role that you created in step 3.

6. In the OpenShift Container Platform, create an object storage secret with keys as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name>
stringData:
  bucket: <s3_bucket_name>
  region: <s3_region>
  role_arn: <s3_role_arn>
type: Opaque
```

7. When the object storage secret is created, update the relevant custom resource of the Distributed Tracing Platform instance as follows:

Example TempoStack custom resource

```
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 The secret that you created in the previous step.

2 If you are not using the CCO, omit this line. If you are using the CCO, add this parameter with the **token-cco** value.

Example TempoMonolithic custom resource

```
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 The secret that you created in the previous step.

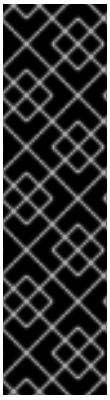
2 If you are not using the CCO, omit this line. If you are using the CCO, add this parameter with the **token-cco** value.

Additional resources

- AWS Identity and Access Management Documentation (AWS documentation)
- AWS Command Line Interface Documentation (AWS documentation)
- Configuring an OpenID Connect identity provider
- Identify AWS resources with Amazon Resource Names (ARNs) (AWS documentation)

3.2.2. Setting up the Azure storage with the Security Token Service

You can set up the Azure storage with the Security Token Service (STS) by using the Azure Command Line Interface (Azure CLI).



IMPORTANT

Using the Distributed Tracing Platform with the Azure storage and STS is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Prerequisites

- You have installed the latest version of the Azure CLI.
- You have created an Azure storage account.
- You have created an Azure blob storage container.

Procedure

1. Create an Azure managed identity by running the following command:

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

1. The name you have chosen for the managed identity.
2. The Azure resource group where you want the identity to be created.
3. The Azure region, which must be the same region as for the resource group.
4. The Azure subscription ID.

2. Create a federated identity credential for the OpenShift Container Platform service account for use by all components of the Distributed Tracing Platform except the Query Frontend. You can do this by running the following command:

```
$ 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. Federated identity credentials allow OpenShift Container Platform service accounts to authenticate as an Azure managed identity without storing secrets or using an Azure service principal identity.

2 The name you have chosen for the federated credential.

3 The URL of the OpenID Connect (OIDC) provider for your cluster.

4 The service account subject for your cluster in the following format: **system:serviceaccount:<namespace>:tempo-<tempostack_instance_name>**.

5 The expected audience, which is to be used for validating the issued tokens for the federated identity credential. This is commonly set to **api://AzureADTokenExchange**.

TIP

You can get the URL of the OpenID Connect (OIDC) issuer for your cluster by running the following command:

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

3. Create a federated identity credential for the OpenShift Container Platform service account for use by the Query Frontend component of the Distributed Tracing Platform. You can do this by running the following command:

```
$ 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 Federated identity credentials allow OpenShift Container Platform service accounts to authenticate as an Azure managed identity without storing secrets or using an Azure service principal identity.

2 The name you have chosen for the frontend federated identity credential.

3 The service account subject for your cluster in the following format: **system:serviceaccount:<namespace>:tempo-<tempostack_instance_name>**.

4. Assign the Storage Blob Data Contributor role to the Azure service principal identity of the created Azure managed identity. You can do this by running the following command:

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

1 The Azure service principal identity of the Azure managed identity that you created in step 1.

TIP

You can get the **<assignee_name>** value by running the following command:

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

5. Fetch the client ID of the Azure managed identity that you created in step 1:

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

1. Copy and paste the **<identity_name>** value from step 1.
2. Copy and paste the **<resource_group>** value from step 1.

6. Create an OpenShift Container Platform secret for the Azure workload identity federation (WIF). You can do this by running the following command:

```
$ 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. The name of the Azure Blob Storage container.
2. The name of the Azure Storage account.
3. The client ID of the managed identity that you fetched in the previous step.
4. Optional: Defaults to **api://AzureADTokenExchange**.
5. Azure Tenant ID.

7. When the object storage secret is created, update the relevant custom resource of the Distributed Tracing Platform instance as follows:

Example TempoStack custom resource

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  storage:
```

```
    secret: 1
      name: <secret_name>
      type: azure
# ...
```

1 The secret that you created in the previous step.

Example TempoMonolithic custom resource

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

1 The secret that you created in the previous step.

Additional resources

- Install the Azure CLI on Linux (Azure documentation)

3.2.3. Setting up the Google Cloud storage with the Security Token Service

You can set up the Google Cloud Storage (GCS) with the Security Token Service (STS) by using the Google Cloud CLI.

IMPORTANT

Using the Distributed Tracing Platform with the GCS and STS is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Prerequisites

- You have installed the latest version of the Google Cloud CLI.

Procedure

1. Create a GCS bucket on the Google Cloud.

2. Create or reuse a service account with Google's Identity and Access Management (IAM):

```
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. The name of the service account on the Google Cloud.
2. The project ID of the service account on the Google Cloud.

3. Bind the required Google Cloud roles to the created service account at the project level. You can do this by running the following command:

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

4. Retrieve the **POOL_ID** value of the Google Cloud Workload Identity Pool that is associated with the cluster. How you can retrieve this value depends on your environment, so the following command is only an example:

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

5. Add the IAM policy bindings. You can do this by running the following commands:

```
$ gcloud iam service-accounts add-iam-policy-binding "$SERVICE_ACCOUNT_EMAIL" \ 1
  --role="roles/iam.workloadIdentityUser" \
  --
member="principal://iam.googleapis.com/projects/<project_number>/locations/global/workloadIc
entityPools/<pool_id>/subject/system:serviceaccount:<tempo_namespace>:tempo-
<tempo_name>" \
  --project=<project_id> \
  --quiet \
&&
  gcloud iam service-accounts add-iam-policy-binding "$SERVICE_ACCOUNT_EMAIL" \
  --role="roles/iam.workloadIdentityUser" \
  --
member="principal://iam.googleapis.com/projects/<project_number>/locations/global/workloadIc
entityPools/<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. The **$SERVICE_ACCOUNT_EMAIL** is the output of the command in step 2.

6. Create a credential file for the **key.json** key of the storage secret for use by the **TempoStack** custom resource. You can do this by running the following command:

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

"projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/providers/<provide
r_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. The **credential-source-file** parameter must always point to the **/var/run/secrets/storage/serviceaccount/token** path because the Operator mounts the token from this path.
2. The path for saving the output file.

7. Get the correct audience by running the following command:

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

8. Create a storage secret for the Distributed Tracing Platform by running the following command.

```
$ 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. The bucket name of the Google Cloud Storage.
2. The audience that you got in the previous step.
3. The credential file that you created in step 6.

9. When the object storage secret is created, update the relevant custom resource of the Distributed Tracing Platform instance as follows:

Example TempoStack custom resource

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
```

```
  namespace: <namespace>
spec:
# ...
  storage:
    secret: 1
      name: <secret_name>
      type: gcs
# ...
```

1 The secret that you created in the previous step.

Example TempoMonolithic custom resource

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

1 The secret that you created in the previous step.

Additional resources

- Install the gcloud CLI (Google Cloud Documentation)
- Service accounts overview (Google Cloud Documentation)

3.2.4. Setting up IBM Cloud Object Storage

You can set up IBM Cloud Object Storage by using the OpenShift CLI (**oc**).

Prerequisites

- You have installed the latest version of OpenShift CLI (**oc**). For more information, see "Getting started with the OpenShift CLI" in *Configure: CLI tools*.
- You have installed the latest version of IBM Cloud Command Line Interface (**ibmcloud**). For more information, see "Getting started with the IBM Cloud CLI" in *IBM Cloud Docs*.
- You have configured IBM Cloud Object Storage. For more information, see "Choosing a plan and creating an instance" in *IBM Cloud Docs*.
  - You have an IBM Cloud Platform account.
  - You have ordered an IBM Cloud Object Storage plan.

- You have created an instance of IBM Cloud Object Storage.

Procedure

1. On IBM Cloud, create an object store bucket.
2. On IBM Cloud, create a service key for connecting to the object store bucket by running the following command:

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

3. On IBM Cloud, create a secret with the bucket credentials by running the following command:

```
$ oc -n <namespace> create secret generic <ibm_cos_secret> \
  --from-literal=bucket="<ibm_bucket_name>" \
  --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. On OpenShift Container Platform, create an object storage secret with keys as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: <ibm_cos_secret>
stringData:
  bucket: <ibm_bucket_name>
  endpoint: <ibm_bucket_endpoint>
  access_key_id: <ibm_bucket_access_key>
  access_key_secret: <ibm_bucket_secret_key>
type: Opaque
```

5. On OpenShift Container Platform, set the storage section in the **TempoStack** custom resource as follows:

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

1 Name of the secret that contains the IBM Cloud Storage access and secret keys.

Additional resources

- Getting started with the OpenShift CLI

- 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. CONFIGURING THE PERMISSIONS AND TENANTS

Before installing a **TempoStack** or **TempoMonolithic** instance, you must define one or more tenants and configure their read and write access. You can configure such an authorization setup by using a cluster role and cluster role binding for the Kubernetes Role-Based Access Control (RBAC). By default, no users are granted read or write permissions. For more information, see "Configuring the read permissions for tenants" and "Configuring the write permissions for tenants".

NOTE

The OpenTelemetry Collector of the Red Hat build of OpenTelemetry can send trace data to a **TempoStack** or **TempoMonolithic** instance by using the service account with RBAC for writing the data.

Table 3.2. Authentication and authorization

Component	Tempo Gateway service	OpenShift OAuth	**TokenReview** API	**SubjectAccess Review** API
Authentication	X	X	X	
Authorization	X			X

3.3.1. Configuring the read permissions for tenants

You can configure the read permissions for tenants from the **Administrator** view of the web console or from the command line.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a cluster administrator with the **cluster-admin** role.
- For Red Hat OpenShift Dedicated, you must be logged in using an account with the **dedicated-admin** role.

Procedure

1. Define the tenants by adding the **tenantName** and **tenantId** parameters with your values of choice to the **TempoStack** custom resource (CR):

Tenant example in a TempoStack CR

```
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) A **tenantName** value of the user's choice.

(2) A **tenantId** value of the user's choice.

2. Add the tenants to a cluster role with the read (**get**) permissions to read traces.

Example RBAC configuration in a ClusterRole resource

```
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) Lists the tenants, **dev** and **prod** in this example, which are defined by using the **tenantName** parameter in the previous step.

(2) Enables the read operation for the listed tenants.

3. Grant authenticated users the read permissions for trace data by defining a cluster role binding for the cluster role from the previous step.

Example RBAC configuration in a ClusterRoleBinding resource

```
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) Grants all authenticated users the read permissions for trace data.

3.3.2. Configuring the write permissions for tenants

You can configure the write permissions for tenants from the **Administrator** view of the web console or from the command line.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a cluster administrator with the **cluster-admin** role.
- For Red Hat OpenShift Dedicated, you must be logged in using an account with the **dedicated-admin** role.
- You have installed the OpenTelemetry Collector and configured it to use an authorized service account with permissions. For more information, see "Creating the required RBAC resources automatically" in the Red Hat build of OpenTelemetry documentation.

Procedure

1. Create a service account for use with OpenTelemetry Collector.

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

2. Add the tenants to a cluster role with the write (**create**) permissions to write traces.

Example RBAC configuration in a ClusterRole resource

```
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) Lists the tenants.

2 Enables the write operation.

3. Grant the OpenTelemetry Collector the write permissions by defining a cluster role binding to attach the OpenTelemetry Collector service account.

Example RBAC configuration in a ClusterRoleBinding resource

```
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 The service account that you created in a previous step. The client uses it when exporting trace data.

4. Configure the **OpenTelemetryCollector** custom resource as follows:

- Add the **bearertokenauth** extension and a valid token to the tracing pipeline service.
- Add the tenant name in the **otlp/otlphttp** exporters as the **X-Scope-OrgID** headers.
- Enable TLS with a valid certificate authority file.

Sample OpenTelemetry CR configuration

```
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. Service account configured with write permissions.
2. Bearer Token extension to use service account token.
3. The service account token. The client sends the token to the tracing pipeline service as the bearer token header.
4. Specify either the OTLP gRPC Exporter (**otlp/dev**) or the OTLP HTTP Exporter (**otlphttp/dev**).
5. Enabled TLS with a valid service CA file.
6. Header with tenant name.
7. Specify either the OTLP gRPC Exporter (**otlp/dev**) or the OTLP HTTP Exporter (**otlphttp/dev**).
8. The exporter you specified in **exporters** section of the CR.

Additional resources

- Creating the required RBAC resources automatically

3.4. INSTALLING A TEMPOSTACK INSTANCE

You can install a **TempoStack** instance by using the web console or command line.

3.4.1. Installing a TempoStack instance by using the web console

You can install a **TempoStack** instance from the **Administrator** view of the web console.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a cluster administrator with the **cluster-admin** role.
- For Red Hat OpenShift Dedicated, you must be logged in using an account with the **dedicated-admin** role.
- You have completed setting up the required object storage by a supported provider: Red Hat OpenShift Data Foundation, MinIO, Amazon S3, Azure Blob Storage, Google Cloud Storage. For more information, see "Object storage setup".

WARNING

Object storage is required and not included with the Distributed Tracing Platform. You must choose and set up object storage by a supported provider before installing the Distributed Tracing Platform.

- You have defined one or more tenants and configured the read and write permissions. For more information, see "Configuring the read permissions for tenants" and "Configuring the write permissions for tenants".

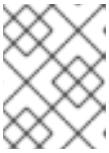
Procedure

1. Go to **Home → Projects → Create Project** to create a permitted project of your choice for the **TempoStack** instance that you will create in a subsequent step. Project names beginning with the **openshift-** prefix are not permitted.
2. Go to **Workloads → Secrets → Create → From YAML** to create a secret for your object storage bucket in the project that you created for the **TempoStack** instance. For more information, see "Object storage setup".

   Example secret for Amazon S3 and MinIO storage

   ```
   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. Create a **TempoStack** instance.



NOTE

You can create multiple **TempoStack** instances in separate projects on the same cluster.

a. Go to **Ecosystem → Installed Operators**.

b. Select **TempoStack → Create TempoStack → YAML view**.

c. In the **YAML view**, customize the **TempoStack** custom resource (CR):

Example TempoStack CR for AWS S3 and MinIO storage and two tenants

```
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) This CR creates a **TempoStack** deployment, which is configured to receive Jaeger Thrift over the HTTP and OpenTelemetry Protocol (OTLP).

(2) The project that you have chosen for the **TempoStack** deployment. Project names beginning with the **openshift-** prefix are not permitted.

(3) Red Hat supports only the custom resource options that are available in the Red Hat OpenShift Distributed Tracing Platform documentation.

(4) Specifies the storage for storing traces.

(5) The secret you created in step 2 for the object storage that had been set up as one of the prerequisites.

(6) The value of the **name** field in the **metadata** section of the secret. For example: **minio**.

7. The accepted values are **azure** for Azure Blob Storage; **gcs** for Google Cloud Storage; and **s3** for Amazon S3, MinIO, or Red Hat OpenShift Data Foundation. For example:

8. The size of the persistent volume claim for the Tempo Write-Ahead Logging (WAL). The default is **10Gi**. For example: **1Gi**.

9. Optional.

10. The value must be **openshift**.

11. The list of tenants.

12. The tenant name, which is used as the value for the **X-Scope-OrgId** HTTP header.

13. The unique identifier of the tenant. Must be unique throughout the lifecycle of the **TempoStack** deployment. The Distributed Tracing Platform uses this ID to prefix objects in the object storage. You can reuse the value of the UUID or **tempoName** field.

14. Enables a gateway that performs authentication and authorization.

15. Exposes the Jaeger UI, which visualizes the data, via a route at **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search**.

d. Select **Create**.

Verification

1. Use the **Project:** dropdown list to select the project of the **TempoStack** instance.
2. Go to **Ecosystem → Installed Operators** to verify that the **Status** of the **TempoStack** instance is **Condition: Ready**.
3. Go to **Workloads → Pods** to verify that all the component pods of the **TempoStack** instance are running.
4. Access the Tempo console:
   a. Go to **Networking → Routes** and **Ctrl+F** to search for **tempo**.
   b. In the **Location** column, open the URL to access the Tempo console.

NOTE

The Tempo console initially shows no trace data following the Tempo console installation.

3.4.2. Installing a TempoStack instance by using the CLI

You can install a **TempoStack** instance from the command line.

Prerequisites

- An active OpenShift CLI (**oc**) session by a cluster administrator with the **cluster-admin** role.

TIP

- Ensure that your OpenShift CLI (**oc**) version is up to date and matches your OpenShift Container Platform version.
- Run the **oc login** command:

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

- You have completed setting up the required object storage by a supported provider: Red Hat OpenShift Data Foundation, MinIO, Amazon S3, Azure Blob Storage, Google Cloud Storage. For more information, see "Object storage setup".

WARNING

Object storage is required and not included with the Distributed Tracing Platform. You must choose and set up object storage by a supported provider before installing the Distributed Tracing Platform.

- You have defined one or more tenants and configured the read and write permissions. For more information, see "Configuring the read permissions for tenants" and "Configuring the write permissions for tenants".

Procedure

1. Run the following command to create a permitted project of your choice for the **TempoStack** instance that you will create in a subsequent step:

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

1 Project names beginning with the **openshift-** prefix are not permitted.

2. In the project that you created for the **TempoStack** instance, create a secret for your object storage bucket by running the following command:

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

For more information, see "Object storage setup".

Example secret for Amazon S3 and MinIO storage

```
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. Create a **TempoStack** instance in the project that you created for it:

NOTE

You can create multiple **TempoStack** instances in separate projects on the same cluster.

a. Customize the **TempoStack** custom resource (CR):

Example TempoStack CR for AWS S3 and MinIO storage and two tenants

```
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. This CR creates a **TempoStack** deployment, which is configured to receive Jaeger Thrift over the HTTP and OpenTelemetry Protocol (OTLP).
2. The project that you have chosen for the **TempoStack** deployment. Project names beginning with the **openshift-** prefix are not permitted.
3. Red Hat supports only the custom resource options that are available in the Red Hat OpenShift Distributed Tracing Platform documentation.
4. Specifies the storage for storing traces.
5. The secret you created in step 2 for the object storage that had been set up as one of the prerequisites.
6. The value of the **name** field in the **metadata** section of the secret. For example: **minio**.
7. The accepted values are **azure** for Azure Blob Storage; **gcs** for Google Cloud Storage; and **s3** for Amazon S3, MinIO, or Red Hat OpenShift Data Foundation. For example: **s3**.
8. The size of the persistent volume claim for the Tempo Write-Ahead Logging (WAL). The default is **10Gi**. For example: **1Gi**.
9. Optional.
10. The value must be **openshift**.
11. The list of tenants.
12. The tenant name, which is used as the value for the **X-Scope-OrgId** HTTP header.
13. The unique identifier of the tenant. Must be unique throughout the lifecycle of the **TempoStack** deployment. The Distributed Tracing Platform uses this ID to prefix objects in the object storage. You can reuse the value of the UUID or **tempoName** field.
14. Enables a gateway that performs authentication and authorization.
15. Exposes the Jaeger UI, which visualizes the data, via a route at **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search**.

b. Apply the customized CR by running the following command:

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

Verification

1. Verify that the **status** of all **TempoStack components** is **Running** and the **conditions** are **type: Ready** by running the following command:

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

2. Verify that all the **TempoStack** component pods are running by running the following command:

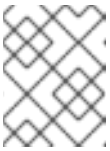
```
$ oc get pods
```

3. Access the Tempo console:

   a. Query the route details by running the following command:

```
$ oc get route
```

   b. Open **https://<route_from_previous_step>** in a web browser.



NOTE

The Tempo console initially shows no trace data following the Tempo console installation.

3.5. INSTALLING A TEMPOMONOLITHIC INSTANCE

IMPORTANT

The **TempoMonolithic** instance is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

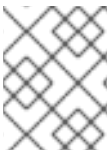
For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

You can install a **TempoMonolithic** instance by using the web console or command line.

The **TempoMonolithic** custom resource (CR) creates a Tempo deployment in monolithic mode. All components of the Tempo deployment, such as the compactor, distributor, ingester, querier, and query frontend, are contained in a single container.

A **TempoMonolithic** instance supports storing traces in in-memory storage, a persistent volume, or object storage.

Tempo deployment in monolithic mode is preferred for a small deployment, demonstration, and testing.



NOTE

The monolithic deployment of Tempo does not scale horizontally. If you require horizontal scaling, use the **TempoStack** CR for a Tempo deployment in microservices mode.

3.5.1. Installing a TempoMonolithic instance by using the web console

IMPORTANT

The **TempoMonolithic** instance is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

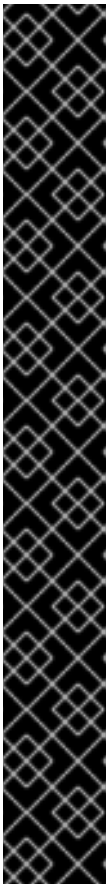
You can install a **TempoMonolithic** instance from the **Administrator** view of the web console.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a cluster administrator with the **cluster-admin** role.
- For Red Hat OpenShift Dedicated, you must be logged in using an account with the **dedicated-admin** role.
- You have defined one or more tenants and configured the read and write permissions. For more information, see "Configuring the read permissions for tenants" and "Configuring the write permissions for tenants".

Procedure

1. Go to **Home → Projects → Create Project** to create a permitted project of your choice for the **TempoMonolithic** instance that you will create in a subsequent step. Project names beginning with the **openshift-** prefix are not permitted.
2. Decide which type of supported storage to use for storing traces: in-memory storage, a persistent volume, or object storage.



IMPORTANT

Object storage is not included with the Distributed Tracing Platform and requires setting up an object store by a supported provider: Red Hat OpenShift Data Foundation, MinIO, Amazon S3, Azure Blob Storage, or Google Cloud Storage.

Additionally, opting for object storage requires creating a secret for your object storage bucket in the project that you created for the **TempoMonolithic** instance. You can do this in **Workloads → Secrets → Create → From YAML**.

For more information, see "Object storage setup".

Example secret for Amazon S3 and MinIO storage

```
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. Create a **TempoMonolithic** instance:

NOTE

You can create multiple **TempoMonolithic** instances in separate projects on the same cluster.

a. Go to **Ecosystem → Installed Operators**.

b. Select **TempoMonolithic → Create TempoMonolithic → YAML view**.

c. In the **YAML view**, customize the **TempoMonolithic** custom resource (CR).

Example TempoMonolithic CR

```
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. This CR creates a **TempoMonolithic** deployment with trace ingestion in the OTLP protocol.
2. The project that you have chosen for the **TempoMonolithic** deployment. Project names beginning with the **openshift-** prefix are not permitted.
3. Red Hat supports only the custom resource options that are available in the Red Hat OpenShift Distributed Tracing Platform documentation.
4. Specifies the storage for storing traces.
5. Type of storage for storing traces: in-memory storage, a persistent volume, or object storage. The value for a persistent volume is **pv**. The accepted values for object storage are **s3**, **gcs**, or **azure**, depending on the used object store type. The default value is **memory** for the **tmpfs** in-memory storage, which is only appropriate for development, testing, demonstrations, and proof-of-concept environments because the data does not persist when the pod is shut down.
6. Memory size: For in-memory storage, this means the size of the **tmpfs** volume, where the default is **2Gi**. For a persistent volume, this means the size of the persistent volume claim, where the default is **10Gi**. For object storage, this means the size of the persistent volume claim for the Tempo Write-Ahead Logging (WAL), where the default is **10Gi**.
7. Optional: For object storage, the type of object storage. The accepted values are **s3**, **gcs**, and **azure**, depending on the used object store type.
8. Optional: For object storage, the value of the **name** in the **metadata** of the storage secret. The storage secret must be in the same namespace as the **TempoMonolithic** instance and contain the fields specified in "Table 1. Required secret parameters" in the section "Object storage setup".
9. Optional.

10 Optional: Name of a **ConfigMap** object that contains a CA certificate.

11 Exposes the Jaeger UI, which visualizes the data, via a route at **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search**.

12 Enables creation of the route for the Jaeger UI.

13 Optional.

14 Lists the tenants.

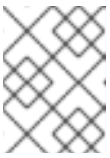
15 The tenant name, which is used as the value for the **X-Scope-OrgId** HTTP header.

16 The unique identifier of the tenant. Must be unique throughout the lifecycle of the **TempoMonolithic** deployment. This ID will be added as a prefix to the objects in the object storage. You can reuse the value of the UUID or **tempoName** field.

d. Select **Create**.

Verification

1. Use the **Project:** dropdown list to select the project of the **TempoMonolithic** instance.
2. Go to **Ecosystem → Installed Operators** to verify that the **Status** of the **TempoMonolithic** instance is **Condition: Ready**.
3. Go to **Workloads → Pods** to verify that the pod of the **TempoMonolithic** instance is running.
4. Access the Jaeger UI:
   a. Go to **Networking → Routes** and **Ctrl+F** to search for **jaegerui**.



NOTE

The Jaeger UI uses the **tempo-<metadata_name_of_TempoMonolithic_CR>-jaegerui** route.

   b. In the **Location** column, open the URL to access the Jaeger UI.
5. When the pod of the **TempoMonolithic** instance is ready, you can send traces to the **tempo-<metadata_name_of_TempoMonolithic_CR>:4317** (OTLP/gRPC) and **tempo-<metadata_name_of_TempoMonolithic_CR>:4318** (OTLP/HTTP) endpoints inside the cluster.
   The Tempo API is available at the **tempo-<metadata_name_of_TempoMonolithic_CR>:3200** endpoint inside the cluster.

3.5.2. Installing a TempoMonolithic instance by using the CLI

IMPORTANT

The **TempoMonolithic** instance is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

You can install a **TempoMonolithic** instance from the command line.

Prerequisites

- An active OpenShift CLI (**oc**) session by a cluster administrator with the **cluster-admin** role.

  TIP

  - Ensure that your OpenShift CLI (**oc**) version is up to date and matches your OpenShift Container Platform version.
  - Run the **oc login** command:

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

- You have defined one or more tenants and configured the read and write permissions. For more information, see "Configuring the read permissions for tenants" and "Configuring the write permissions for tenants".

Procedure

1. Run the following command to create a permitted project of your choice for the **TempoMonolithic** instance that you will create in a subsequent step:

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

   1 Project names beginning with the **openshift-** prefix are not permitted.

2. Decide which type of supported storage to use for storing traces: in-memory storage, a persistent volume, or object storage.

IMPORTANT

Object storage is not included with the Distributed Tracing Platform and requires setting up an object store by a supported provider: Red Hat OpenShift Data Foundation, MinIO, Amazon S3, Azure Blob Storage, or Google Cloud Storage.

Additionally, opting for object storage requires creating a secret for your object storage bucket in the project that you created for the **TempoMonolithic** instance. You can do this by running the following command:

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

For more information, see "Object storage setup".

Example secret for Amazon S3 and MinIO storage

```
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. Create a **TempoMonolithic** instance in the project that you created for it.

TIP

You can create multiple **TempoMonolithic** instances in separate projects on the same cluster.

a. Customize the **TempoMonolithic** custom resource (CR).

Example TempoMonolithic CR

```
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. This CR creates a **TempoMonolithic** deployment with trace ingestion in the OTLP protocol.
2. The project that you have chosen for the **TempoMonolithic** deployment. Project names beginning with the **openshift-** prefix are not permitted.
3. Red Hat supports only the custom resource options that are available in the Red Hat OpenShift Distributed Tracing Platform documentation.
4. Specifies the storage for storing traces.
5. Type of storage for storing traces: in-memory storage, a persistent volume, or object storage. The value for a persistent volume is **pv**. The accepted values for object storage are **s3**, **gcs**, or **azure**, depending on the used object store type. The default value is **memory** for the **tmpfs** in-memory storage, which is only appropriate for development, testing, demonstrations, and proof-of-concept environments because the data does not persist when the pod is shut down.
6. Memory size: For in-memory storage, this means the size of the **tmpfs** volume, where the default is **2Gi**. For a persistent volume, this means the size of the persistent volume claim, where the default is **10Gi**. For object storage, this means the size of the persistent volume claim for the Tempo Write-Ahead Logging (WAL), where the default is **10Gi**.
7. Optional: For object storage, the type of object storage. The accepted values are **s3**, **gcs**, and **azure**, depending on the used object store type.
8. Optional: For object storage, the value of the **name** in the **metadata** of the storage secret. The storage secret must be in the same namespace as the **TempoMonolithic** instance and contain the fields specified in "Table 1. Required secret parameters" in the section "Object storage setup".
9. Optional.

10 Optional: Name of a **ConfigMap** object that contains a CA certificate.

11 Exposes the Jaeger UI, which visualizes the data, via a route at **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search**.

12 Enables creation of the route for the Jaeger UI.

13 Optional.

14 Lists the tenants.

15 The tenant name, which is used as the value for the **X-Scope-OrgId** HTTP header.

16 The unique identifier of the tenant. Must be unique throughout the lifecycle of the **TempoMonolithic** deployment. This ID will be added as a prefix to the objects in the object storage. You can reuse the value of the UUID or **tempoName** field.

b. Apply the customized CR by running the following command:

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

Verification

1. Verify that the **status** of all **TempoMonolithic components** is **Running** and the **conditions** are **type: Ready** by running the following command:

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

2. Run the following command to verify that the pod of the **TempoMonolithic** instance is running:

```
$ oc get pods
```

3. Access the Jaeger UI:

   a. Query the route details for the **tempo-<metadata_name_of_tempomonolithic_cr>-jaegerui** route by running the following command:

```
$ oc get route
```

   b. Open **https://<route_from_previous_step>** in a web browser.

4. When the pod of the **TempoMonolithic** instance is ready, you can send traces to the **tempo-<metadata_name_of_tempomonolithic_cr>:4317** (OTLP/gRPC) and **tempo-<metadata_name_of_tempomonolithic_cr>:4318** (OTLP/HTTP) endpoints inside the cluster. The Tempo API is available at the **tempo-<metadata_name_of_tempomonolithic_cr>:3200** endpoint inside the cluster.

3.6. ADDITIONAL RESOURCES

- Creating a cluster admin
- OperatorHub.io
- Accessing the web console
- Installing from the software catalog using the web console
- Creating applications from installed Operators
- Getting started with the OpenShift CLI

CHAPTER 4. CONFIGURING THE DISTRIBUTED TRACING PLATFORM

The Tempo Operator uses a custom resource definition (CRD) file that defines the architecture and configuration settings for creating and deploying the Distributed Tracing Platform resources. You can install the default configuration or modify the file.

4.1. CONFIGURING BACK-END STORAGE

For information about configuring the back-end storage, see Understanding persistent storage and the relevant configuration section for your chosen storage option.

4.2. INTRODUCTION TO TEMPOSTACK CONFIGURATION PARAMETERS

The **TempoStack** custom resource (CR) defines the architecture and settings for creating the Distributed Tracing Platform resources. You can modify these parameters to customize your implementation to your business needs.

Example TempoStack CR

```
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. API version to use when creating the object.
2. Defines the kind of Kubernetes object to create.
3. Data that uniquely identifies the object, including a **name** string, **UID**, and optional **namespace**. OpenShift Container Platform automatically generates the **UID** and completes the **namespace** with the name of the project where the object is created.
4. Name of the TempoStack instance.
5. Contains all of the configuration parameters of the TempoStack instance. When a common definition for all Tempo components is required, define it in the **spec** section. When the definition relates to an individual component, place it in the **spec.template.<component>** section.
6. Storage is specified at instance deployment. See the installation page for information about storage options for the instance.
7. Defines the compute resources for the Tempo container.
8. Integer value for the number of ingesters that must acknowledge the data from the distributors before accepting a span.
9. Configuration options for retention of traces. The default value is **48h**.
10. Configuration options for the Tempo **distributor** component.
11. Configuration options for the Tempo **ingester** component.
12. Configuration options for the Tempo **compactor** component.
13. Configuration options for the Tempo **querier** component.
14. Configuration options for the Tempo **query-frontend** component.
15. Configuration options for the Tempo **gateway** component.
16. Limits ingestion and query rates.
17. Defines ingestion rate limits.
18. Defines query rate limits.
19. Configures operands to handle telemetry data.
20. Configures search capabilities.
21. Defines whether or not this CR is managed by the Operator. The default value is **managed**.

Table 4.1. **TempoStack** CR parameters

Parameter	Description	Values	Default value
apiVersion:	API version to use when creating the object.	**tempo.grafana.com/v1alpha1**	**tempo.grafana.com/v1alpha1**

Parameter	Description	Values	Default value
kind:	Defines the kind of the Kubernetes object to create.	**tempo**	
metadata:	Data that uniquely identifies the object, including a **name** string, **UID**, and optional **namespace**.		OpenShift Container Platform automatically generates the **UID** and completes the **namespace** with the name of the project where the object is created.
name:	Name for the object.	Name of your TempoStack instance.	**tempo-all-in-one-inmemory**
spec:	Specification for the object to be created.	Contains all of the configuration parameters for your TempoStack instance. When a common definition for all Tempo components is required, it is defined under the **spec** node. When the definition relates to an individual component, it is placed under the **spec.template.<component>** node.	N/A
resources:	Resources assigned to the TempoStack instance.		
storageSize:	Storage size for ingester PVCs.		
replicationFactor:	Configuration for the replication factor.		
retention:	Configuration options for retention of traces.		
storage:	Configuration options that define the storage.		

Parameter	Description	Values	Default value
template.distributor:	Configuration options for the Tempo distributor.		
template.ingester:	Configuration options for the Tempo ingester.		
template.compactor:	Configuration options for the Tempo compactor.		
template.querier:	Configuration options for the Tempo querier.		
template.queryFrontend:	Configuration options for the Tempo query frontend.		
template.gateway:	Configuration options for the Tempo gateway.		

Additional resources

- Installing a TempoStack instance
- Installing a TempoMonolithic instance

4.3. QUERY CONFIGURATION OPTIONS

Two components of the Distributed Tracing Platform, the querier and query frontend, manage queries. You can configure both of these components.

The querier component finds the requested trace ID in the ingesters or back-end storage. Depending on the set parameters, the querier component can query both the ingesters and pull bloom or indexes from the back end to search blocks in object storage. The querier component exposes an HTTP endpoint at **GET /querier/api/traces/<trace_id>**, but it is not expected to be used directly. Queries must be sent to the query frontend.

Table 4.2. Configuration parameters for the querier component

Parameter	Description	Values
nodeSelector	The simple form of the node-selection constraint.	type: object
replicas	The number of replicas to be created for the component.	type: integer; format: int32

Parameter	Description	Values
tolerations	Component-specific pod tolerations.	type: array

The query frontend component is responsible for sharding the search space for an incoming query. The query frontend exposes traces via a simple HTTP endpoint: **GET /api/traces/<trace_id>**. Internally, the query frontend component splits the **blockID** space into a configurable number of shards and then queues these requests. The querier component connects to the query frontend component via a streaming gRPC connection to process these sharded queries.

Table 4.3. Configuration parameters for the query frontend component

Parameter	Description	Values
component	Configuration of the query frontend component.	type: object
component.nodeSelector	The simple form of the node selection constraint.	type: object
component.replicas	The number of replicas to be created for the query frontend component.	type: integer; format: int32
component.tolerations	Pod tolerations specific to the query frontend component.	type: array
jaegerQuery	The options specific to the Jaeger Query component.	type: object
jaegerQuery.enabled	When **enabled**, creates the Jaeger Query component,**jaegerQuery**.	type: boolean
jaegerQuery.ingress	The options for the Jaeger Query ingress.	type: object
jaegerQuery.ingress.annotations	The annotations of the ingress object.	type: object
jaegerQuery.ingress.host	The hostname of the ingress object.	type: string
jaegerQuery.ingress.ingressClassName	The name of an IngressClass cluster resource. Defines which ingress controller serves this ingress resource.	type: string

Parameter	Description	Values
jaegerQuery.ingress.route	The options for the OpenShift route.	type: object
jaegerQuery.ingress.route.termination	The termination type. The default is **edge**.	type: string (enum: insecure, edge, passthrough, reencrypt)
jaegerQuery.ingress.type	The type of ingress for the Jaeger Query UI. The supported types are **ingress**, **route**, and **none**.	type: string (enum: ingress, route)
jaegerQuery.monitorTab	The monitor tab configuration.	type: object
jaegerQuery.monitorTab.enabled	Enables the monitor tab in the Jaeger console. The **PrometheusEndpoint** must be configured.	type: boolean
jaegerQuery.monitorTab.prometheusEndpoint	The endpoint to the Prometheus instance that contains the span rate, error, and duration (RED) metrics. For example, **https://thanos-querier.openshift-monitoring.svc.cluster.local:9092**.	type: string

Example configuration of the query frontend component in a TempoStack CR

```
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
```

Additional resources

- Understanding taints and tolerations

4.4. CONFIGURING THE UI

You can use the distributed tracing UI plugin of the Cluster Observability Operator (COO) as the user interface (UI) for the Red Hat OpenShift Distributed Tracing Platform. For more information about installing and using the distributed tracing UI plugin, see "Distributed tracing UI plugin" in *Cluster Observability Operator*.

Additional resources

- Distributed tracing UI plugin

4.5. CONFIGURING THE MONITOR TAB IN JAEGER UI

You can have the request rate, error, and duration (RED) metrics extracted from traces and visualized through the Jaeger Console in the **Monitor** tab of the OpenShift Container Platform web console. The metrics are derived from spans in the OpenTelemetry Collector that are scraped from the Collector by Prometheus, which you can deploy in your user-workload monitoring stack. The Jaeger UI queries these metrics from the Prometheus endpoint and visualizes them.

Prerequisites

- You have configured the permissions and tenants for the Distributed Tracing Platform. For more information, see "Configuring the permissions and tenants".

Procedure

1. In the **OpenTelemetryCollector** custom resource of the OpenTelemetry Collector, enable the Spanmetrics Connector (**spanmetrics**), which derives metrics from traces and exports the metrics in the Prometheus format.

   Example OpenTelemetryCollector custom resource for span RED

```
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. Creates the **ServiceMonitor** custom resource to enable scraping of the Prometheus exporter.
2. The Spanmetrics connector receives traces and exports metrics.
3. The OTLP receiver to receive spans in the OpenTelemetry protocol.
4. The Prometheus exporter is used to export metrics in the Prometheus format.
5. The resource attributes are dropped by default.
6. The Spanmetrics connector is configured as exporter in traces pipeline.
7. The Spanmetrics connector is configured as receiver in metrics pipeline.

2. In the **TempoStack** custom resource, enable the **Monitor** tab and set the Prometheus endpoint to the Thanos querier service to query the data from your user-defined monitoring stack.

Example TempoStack custom resource with the enabled Monitor tab

```
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. Enables the monitoring tab in the Jaeger console.
2. The service name for Thanos Querier from user-workload monitoring.
3. Optional: The metrics namespace on which the Jaeger query retrieves the Prometheus metrics. Include this line only if you are using an OpenTelemetry Collector version earlier than 0.109.0. If you are using an OpenTelemetry Collector version 0.109.0 or later, omit this line.

3. Optional: Use the span RED metrics generated by the **spanmetrics** connector with alerting rules. For example, for alerts about a slow service or to define service level objectives (SLOs), the connector creates a **duration_bucket** histogram and the **calls** counter metric. These metrics have labels that identify the service, API name, operation type, and other attributes.

Table 4.4. Labels of the metrics created in the spanmetrics connector

Label	Description	Values

Label	Description	Values
service_name	Service name set by the **otel_service_name** environment variable.	**frontend**
span_name	Name of the operation.	• / • **/customer**
span_kind	Identifies the server, client, messaging, or internal operation.	• **SPAN_KIND_SERVER** • **SPAN_KIND_CLIENT** • **SPAN_KIND_PRODUCER** • **SPAN_KIND_CONSUMER** • **SPAN_KIND_INTERNAL**

Example PrometheusRule custom resource that defines an alerting rule for SLO when not serving 95% of requests within 2000ms on the front-end service

```
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. The expression for checking if 95% of the front-end server response time values are below 2000 ms. The time range (**[5m]**) must be at least four times the scrape interval and long enough to accommodate a change in the metric.

Additional resources

- Configuring the permissions and tenants

4.6. CONFIGURING THE RECEIVER TLS

The custom resource of your TempoStack or TempoMonolithic instance supports configuring the TLS for receivers by using user-provided certificates or OpenShift's service serving certificates.

4.6.1. Receiver TLS configuration for a TempoStack instance

You can provide a TLS certificate in a secret or use the service serving certificates that are generated by OpenShift Container Platform.

- To provide a TLS certificate in a secret, configure it in the **TempoStack** custom resource.

NOTE

This feature is not supported with the enabled Tempo Gateway.

TLS for receivers and using a user-provided certificate in a secret

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

1. TLS enabled at the Tempo Distributor.
2. Secret containing a **tls.key** key and **tls.crt** certificate that you apply in advance.
3. Optional: CA in a config map to enable mutual TLS authentication (mTLS).

- Alternatively, you can use the service serving certificates that are generated by OpenShift Container Platform.

NOTE

Mutual TLS authentication (mTLS) is not supported with this feature.

TLS for receivers and using the service serving certificates that are generated by OpenShift Container Platform

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

```
# ...
spec:
# ...
  template:
    distributor:
      tls:
        enabled: true (1)
# ...
```

(1) Sufficient configuration for the TLS at the Tempo Distributor.

Additional resources

- Understanding service serving certificates
- Service CA certificates

4.6.2. Receiver TLS configuration for a TempoMonolithic instance

You can provide a TLS certificate in a secret or use the service serving certificates that are generated by OpenShift Container Platform.

- To provide a TLS certificate in a secret, configure it in the **TempoMonolithic** custom resource.

NOTE

This feature is not supported with the enabled Tempo Gateway.

TLS for receivers and using a user-provided certificate in a secret

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

(1) TLS enabled at the Tempo Distributor.

(2) Secret containing a **tls.key** key and **tls.crt** certificate that you apply in advance.

(3) Optional: CA in a config map to enable mutual TLS authentication (mTLS).

- Alternatively, you can use the service serving certificates that are generated by OpenShift Container Platform.

NOTE

Mutual TLS authentication (mTLS) is not supported with this feature.

TLS for receivers and using the service serving certificates that are generated by OpenShift Container Platform

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

1 Minimal configuration for the TLS at the Tempo Distributor.

Additional resources

- Understanding service serving certificates
- Service CA certificates

4.7. CONFIGURING THE QUERY RBAC

As an administrator, you can set up the query role-based access control (RBAC) to filter the span attributes for your users by the namespaces for which you granted them permissions.

NOTE

When you enable the query RBAC, users can still access traces from all namespaces, and the **service.name** and **k8s.namespace.name** attributes are also visible to all users.

Prerequisites

- An active OpenShift CLI (**oc**) session by a cluster administrator with the **cluster-admin** role.

  TIP

  - Ensure that your OpenShift CLI (**oc**) version is up to date and matches your OpenShift Container Platform version.
  - Run **oc login**:

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

Procedure

1. Enable multitenancy and query RBAC in the **TempoStack** custom resource (CR), for example:

```
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) Always set to **true**.

(2) Always set to **true**.

(3) Always set to **false**.

2. Create a cluster role and cluster role binding to grant the target users the permissions to access the tenant that you specified in the **TempoStack** CR, for example:

```
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. Tenant name in the **TempoStack** CR.
2. Means all authenticated OpenShift users.

3. Grant the target users the permissions to read attributes for the project. You can do this by running the following command:

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

4.8. USING TAINTS AND TOLERATIONS

To schedule the TempoStack pods on dedicated nodes, see How to deploy the different TempoStack components on infra nodes using nodeSelector and tolerations in OpenShift 4.

4.9. CONFIGURING MONITORING AND ALERTS

The Tempo Operator supports monitoring and alerts about each TempoStack component such as distributor, ingester, and so on, and exposes upgrade and operational metrics about the Operator itself.

4.9.1. Configuring the TempoStack metrics and alerts

You can enable metrics and alerts of TempoStack instances.

Prerequisites

- Monitoring for user-defined projects is enabled in the cluster.

Procedure

1. To enable metrics of a TempoStack instance, set the **spec.observability.metrics.createServiceMonitors** field to **true**:

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

2. To enable alerts for a TempoStack instance, set the **spec.observability.metrics.createPrometheusRules** field to **true**:

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

Verification

You can use the **Administrator** view of the web console to verify successful configuration:

1. Go to **Observe → Targets**, filter for **Source: User**, and check that **ServiceMonitors** in the format **tempo-<instance_name>-<component>** have the **Up** status.

2. To verify that alerts are set up correctly, go to **Observe → Alerting → Alerting rules**, filter for **Source: User**, and check that the **Alert rules** for the TempoStack instance components are available.

Additional resources

- Enabling monitoring for user-defined projects

4.9.2. Configuring the Tempo Operator metrics and alerts

When installing the Tempo Operator from the web console, you can select the **Enable Operator recommended cluster monitoring on this Namespace** checkbox, which enables creating metrics and alerts of the Tempo Operator.

If the checkbox was not selected during installation, you can manually enable metrics and alerts even after installing the Tempo Operator.

Procedure

- Add the **openshift.io/cluster-monitoring: "true"** label in the project where the Tempo Operator is installed, which is **openshift-tempo-operator** by default.

Verification

You can use the **Administrator** view of the web console to verify successful configuration:

1. Go to **Observe → Targets**, filter for **Source: Platform**, and search for **tempo-operator**, which must have the **Up** status.

2. To verify that alerts are set up correctly, go to **Observe → Alerting → Alerting rules**, filter for **Source: Platform**, and locate the **Alert rules** for the **Tempo Operator**.

CHAPTER 5. TROUBLESHOOTING THE DISTRIBUTED TRACING PLATFORM

You can diagnose and fix issues in TempoStack or TempoMonolithic instances by using various troubleshooting methods.

5.1. COLLECTING DIAGNOSTIC DATA FROM THE COMMAND LINE

When submitting a support case, it is helpful to include diagnostic information about your cluster to Red Hat Support. You can use the **oc adm must-gather** tool to gather diagnostic data for resources of various types, such as **TempoStack** or **TempoMonolithic**, and the created resources like **Deployment**, **Pod**, or **ConfigMap**. The **oc adm must-gather** tool creates a new pod that collects this data.

Procedure

- From the directory where you want to save the collected data, run the **oc adm must-gather** command to collect the data:

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

1 The default namespace where the Operator is installed is **openshift-tempo-operator**.

Verification

- Verify that the new directory is created and contains the collected data.

CHAPTER 6. UPGRADING

For version upgrades, the Tempo Operator uses the Operator Lifecycle Manager (OLM), which controls installation, upgrade, and role-based access control (RBAC) of Operators in a cluster.

The OLM runs in the OpenShift Container Platform by default. The OLM queries for available Operators as well as upgrades for installed Operators.

When the Tempo Operator is upgraded to the new version, it scans for running TempoStack instances that it manages and upgrades them to the version corresponding to the Operator's new version.

6.1. ADDITIONAL RESOURCES

- Operator Lifecycle Manager concepts and resources
- Updating installed Operators

CHAPTER 7. REMOVING THE DISTRIBUTED TRACING PLATFORM

The steps for removing the Red Hat OpenShift Distributed Tracing Platform from an OpenShift Container Platform cluster are as follows:

1. Shut down all Distributed Tracing Platform pods.
2. Remove any TempoStack instances.
3. Remove the Tempo Operator.

7.1. REMOVING BY USING THE WEB CONSOLE

You can remove a TempoStack instance in the **Administrator** view of the web console.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a cluster administrator with the **cluster-admin** role.
- For Red Hat OpenShift Dedicated, you must be logged in using an account with the **dedicated-admin** role.

Procedure

1. Go to **Ecosystem → Installed Operators → Tempo Operator → TempoStack**.
2. To remove the TempoStack instance, select  **→ Delete TempoStack → Delete**.
3. Optional: Remove the Tempo Operator.

7.2. REMOVING BY USING THE CLI

You can remove a TempoStack instance on the command line.

Prerequisites

- An active OpenShift CLI (**oc**) session by a cluster administrator with the **cluster-admin** role.

  TIP

  - Ensure that your OpenShift CLI (**oc**) version is up to date and matches your OpenShift Container Platform version.
  - Run **oc login**:

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

Procedure

1. Get the name of the TempoStack instance by running the following command:

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

2. Remove the TempoStack instance by running the following command:

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

3. Optional: Remove the Tempo Operator.

Verification

1. Run the following command to verify that the TempoStack instance is not found in the output, which indicates its successful removal:

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

7.3. ADDITIONAL RESOURCES

- Deleting Operators from a cluster
- Getting started with the OpenShift CLI