OpenShift Container Platform 4.6

Distributed tracing

Jaeger installation, usage, and release notes

Last Updated: 2022-08-04

OpenShift Container Platform 4.6 Distributed tracing

Jaeger installation, usage, and release notes

Legal Notice

Abstract

This document provides information on how to use Jaeger in OpenShift Container Platform.

Table of Contents

CHAPTER 1. DISTRIBUTED TRACING RELEASE NOTES

1.1. DISTRIBUTED TRACING OVERVIEW

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

With distributed tracing you can perform the following functions:

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

Red Hat OpenShift distributed tracing consists of two main components:

- **Red Hat OpenShift distributed tracing platform** - This component is based on the open source Jaeger project.
- **Red Hat OpenShift distributed tracing data collection** - This component is based on the open source OpenTelemetry project.

Both of these components are based on the vendor-neutral OpenTracing APIs and instrumentation.

1.2. MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

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 Insights in OpenShift Cluster Manager. Insights 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, please submit a Bugzilla report against the **OpenShift Container Platform** product for the **Documentation** component. Please provide specific details, such as the section name and OpenShift Container Platform version.

1.4. NEW FEATURES AND ENHANCEMENTS

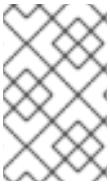
This release adds improvements related to the following components and concepts.

1.4.1. New features and enhancements Red Hat OpenShift distributed tracing 2.4

This release of Red Hat OpenShift distributed tracing addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

This release also adds support for auto-provisioning certificates using the Red Hat Elasticsearch Operator.

- Self-provisioning, which means using the Red Hat OpenShift distributed tracing platform Operator to call the Red Hat Elasticsearch Operator during installation. Self provisioning is fully supported with this release.
- Creating the Elasticsearch instance and certificates first and then configuring the distributed tracing platform to use the certificate is a Technology Preview for this release.



NOTE

When upgrading to Red Hat OpenShift distributed tracing 2.4, the Operator recreates the Elasticsearch instance, which might take five to ten minutes. Distributed tracing will be down and unavailable for that period.

1.4.1.1. Component versions supported in Red Hat OpenShift distributed tracing version 2.4

Operator	Component	Version
Red Hat OpenShift distributed tracing platform	Jaeger	1.34.1
Red Hat OpenShift distributed tracing data collection	OpenTelemetry	0.49

1.4.2. New features and enhancements Red Hat OpenShift distributed tracing 2.3.1

This release of Red Hat OpenShift distributed tracing addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.4.2.1. Component versions supported in Red Hat OpenShift distributed tracing version 2.3.1

Operator	Component	Version
Red Hat OpenShift distributed tracing platform	Jaeger	1.30.2
Red Hat OpenShift distributed tracing data collection	OpenTelemetry	0.44.1-1

1.4.3. New features and enhancements Red Hat OpenShift distributed tracing 2.3.0

This release of Red Hat OpenShift distributed tracing addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

With this release, the Red Hat OpenShift distributed tracing platform Operator is now installed to the **openshift-distributed-tracing** namespace by default. Previously the default installation had been in the **openshift-operator** namespace.

1.4.3.1. Component versions supported in Red Hat OpenShift distributed tracing version 2.3.0

Operator	Component	Version
Red Hat OpenShift distributed tracing platform	Jaeger	1.30.1
Red Hat OpenShift distributed tracing data collection	OpenTelemetry	0.44.0

1.4.4. New features and enhancements Red Hat OpenShift distributed tracing 2.2.0

This release of Red Hat OpenShift distributed tracing addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.4.4.1. Component versions supported in Red Hat OpenShift distributed tracing version 2.2.0

Operator	Component	Version
Red Hat OpenShift distributed tracing platform	Jaeger	1.30.0
Red Hat OpenShift distributed tracing data collection	OpenTelemetry	0.42.0

1.4.5. New features and enhancements Red Hat OpenShift distributed tracing 2.1.0

This release of Red Hat OpenShift distributed tracing addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.4.5.1. Component versions supported in Red Hat OpenShift distributed tracing version 2.1.0

Operator	Component	Version
Red Hat OpenShift distributed tracing platform	Jaeger	1.29.1

Operator	Component	Version
Red Hat OpenShift distributed tracing data collection	OpenTelemetry	0.41.1

1.4.6. New features and enhancements Red Hat OpenShift distributed tracing 2.0.0

This release marks the rebranding of Red Hat OpenShift Jaeger to Red Hat OpenShift distributed tracing. This release consists of the following changes, additions, and improvements:

- Red Hat OpenShift distributed tracing now consists of the following two main components:
  - **Red Hat OpenShift distributed tracing platform** - This component is based on the open source Jaeger project.
  - **Red Hat OpenShift distributed tracing data collection** - This component is based on the open source OpenTelemetry project.
- Updates Red Hat OpenShift distributed tracing platform Operator to Jaeger 1.28. Going forward, Red Hat OpenShift distributed tracing will only support the **stable** Operator channel. Channels for individual releases are no longer supported.
- Introduces a new Red Hat OpenShift distributed tracing data collection Operator based on OpenTelemetry 0.33. Note that this Operator is a Technology Preview feature.
- Adds support for OpenTelemetry protocol (OTLP) to the Query service.
- Introduces a new distributed tracing icon that appears in the OpenShift OperatorHub.
- Includes rolling updates to the documentation to support the name change and new features.

This release also addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.4.6.1. Component versions supported in Red Hat OpenShift distributed tracing version 2.0.0

Operator	Component	Version
Red Hat OpenShift distributed tracing platform	Jaeger	1.28.0
Red Hat OpenShift distributed tracing data collection	OpenTelemetry	0.33.0

1.5. RED HAT OPENSHIFT DISTRIBUTED TRACING TECHNOLOGY PREVIEW

IMPORTANT

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 https://access.redhat.com/support/offerings/techpreview/.

1.5.1. Red Hat OpenShift distributed tracing 2.4.0 Technology Preview

This release also adds support for auto-provisioning certificates using the Red Hat Elasticsearch Operator.

- Self-provisioning, which means using the Red Hat OpenShift distributed tracing platform Operator to call the Red Hat Elasticsearch Operator during installation. Self provisioning is fully supported with this release.
- Creating the Elasticsearch instance and certificates first and then configuring the distributed tracing platform to use the certificate is a Technology Preview for this release.

1.5.2. Red Hat OpenShift distributed tracing 2.2.0 Technology Preview

Unsupported OpenTelemetry Collector components included in the 2.1 release have been removed.

1.5.3. Red Hat OpenShift distributed tracing 2.1.0 Technology Preview

This release introduces a breaking change to how to configure certificates in the OpenTelemetry custom resource file. In the new version, the **ca_file** moves under **tls** in the custom resource, as shown in the following examples.

CA file configuration for OpenTelemetry version 0.33

```
spec:
  mode: deployment
  config: |
    exporters:
      jaeger:
        endpoint: jaeger-production-collector-headless.tracing-system.svc:14250
        ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
```

CA file configuration for OpenTelemetry version 0.41.1

```
spec:
  mode: deployment
  config: |
    exporters:
      jaeger:
        endpoint: jaeger-production-collector-headless.tracing-system.svc:14250
        tls:
          ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
```

1.5.4. Red Hat OpenShift distributed tracing 2.0.0 Technology Preview

This release includes the addition of the Red Hat OpenShift distributed tracing data collection, which you install using the Red Hat OpenShift distributed tracing data collection Operator. Red Hat OpenShift distributed tracing data collection is based on the OpenTelemetry APIs and instrumentation.

Red Hat OpenShift distributed tracing data collection includes the OpenTelemetry Operator and Collector. The Collector can be used to receive traces in either the OpenTelemetry or Jaeger protocol and send the trace data to Red Hat OpenShift distributed tracing. Other capabilities of the Collector are not supported at this time.

The OpenTelemetry Collector allows developers to instrument their code with vendor agnostic APIs, avoiding vendor lock-in and enabling a growing ecosystem of observability tooling.

1.6. RED HAT OPENSHIFT DISTRIBUTED TRACING KNOWN ISSUES

These limitations exist in Red Hat OpenShift distributed tracing:

- Apache Spark is not supported.
- The streaming deployment via AMQ/Kafka is unsupported on IBM Z and IBM Power Systems.

These are the known issues for Red Hat OpenShift distributed tracing:

- TRACING-2057 The Kafka API has been updated to **v1beta2** to support the Strimzi Kafka Operator 0.23.0. However, this API version is not supported by AMQ Streams 1.6.3. If you have the following environment, your Jaeger services will not be upgraded, and you cannot create new Jaeger services or modify existing Jaeger services:
  - Jaeger Operator channel: **1.17.x stable** or **1.20.x stable**
  - AMQ Streams Operator channel: **amq-streams-1.6.x**
    To resolve this issue, switch the subscription channel for your AMQ Streams Operator to either **amq-streams-1.7.x** or **stable**.

1.7. RED HAT OPENSHIFT DISTRIBUTED TRACING FIXED ISSUES

- TRACING-2337 Jaeger is logging a repetitive warning message in the Jaeger logs similar to the following:

  ```
  {"level":"warn","ts":1642438880.918793,"caller":"channelz/logging.go:62","msg":"[core]grpc: Server.Serve failed to create ServerTransport: connection error: desc = \"transport: http2Server.HandleStreams received bogus greeting from client: \\\"\\\\x16\\\\x03\\\\x01\\\\x02\\\\x00\\\\x01\\\\x00\\\\x01\\\\xfc\\\\x03\\\\x03vw\\\\x1a\\\\xc9T\\\\xe7\\\\xdaCj\\\\xb7\\\\x8dK\\\\xa6\\\"\"","system":"grpc","grpc_log":true}
  ```

  This issue was resolved by exposing only the HTTP(S) port of the query service, and not the gRPC port.

- TRACING-2009 The Jaeger Operator has been updated to include support for the Strimzi Kafka Operator 0.23.0.
- TRACING-1907 The Jaeger agent sidecar injection was failing due to missing config maps in the application namespace. The config maps were getting automatically deleted due to an incorrect **OwnerReference** field setting and as a result, the application pods were not moving past the "ContainerCreating" stage. The incorrect settings have been removed.

- TRACING-1725 Follow-up to TRACING-1631. Additional fix to ensure that Elasticsearch certificates are properly reconciled when there are multiple Jaeger production instances, using same name but within different namespaces. See also BZ-1918920.

- TRACING-1631 Multiple Jaeger production instances, using same name but within different namespaces, causing Elasticsearch certificate issue. When multiple service meshes were installed, all of the Jaeger Elasticsearch instances had the same Elasticsearch secret instead of individual secrets, which prevented the OpenShift Elasticsearch Operator from communicating with all of the Elasticsearch clusters.

- TRACING-1300 Failed connection between Agent and Collector when using Istio sidecar. An update of the Jaeger Operator enabled TLS communication by default between a Jaeger sidecar agent and the Jaeger Collector.

- TRACING-1208 Authentication "500 Internal Error" when accessing Jaeger UI. When trying to authenticate to the UI using OAuth, I get a 500 error because oauth-proxy sidecar doesn't trust the custom CA bundle defined at installation time with the **additionalTrustBundle**.

- TRACING-1166 It is not currently possible to use the Jaeger streaming strategy within a disconnected environment. When a Kafka cluster is being provisioned, it results in a error: **Failed to pull image registry.redhat.io/amq7/amq-streams-kafka-24-rhel7@sha256:f9ceca004f1b7dccb3b82d9a8027961f9fe4104e0ed69752c0bdd8078b4a1076**.

- TRACING-809 Jaeger Ingester is incompatible with Kafka 2.3. When there are two or more instances of the Jaeger Ingester and enough traffic it will continuously generate rebalancing messages in the logs. This is due to a regression in Kafka 2.3 that was fixed in Kafka 2.3.1. For more information, see Jaegertracing-1819.

- BZ-1918920/LOG-1619 The Elasticsearch pods does not get restarted automatically after an update.
  Workaround: Restart the pods manually.

CHAPTER 2. DISTRIBUTED TRACING ARCHITECTURE

2.1. DISTRIBUTED TRACING ARCHITECTURE

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 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 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 is comprised of one or more spans.

A *span* represents a logical unit of work in Red Hat OpenShift distributed tracing 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.

2.1.1. Distributed tracing overview

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

With distributed tracing you can perform the following functions:

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

Red Hat OpenShift distributed tracing consists of two main components:

- **Red Hat OpenShift distributed tracing platform** - This component is based on the open source Jaeger project.
- **Red Hat OpenShift distributed tracing data collection** - This component is based on the open source OpenTelemetry project.

Both of these components are based on the vendor-neutral OpenTracing APIs and instrumentation.

2.1.2. Red Hat OpenShift distributed tracing features

Red Hat OpenShift distributed tracing provides the following capabilities:

- Integration with Kiali – When properly configured, you can view distributed tracing data from the Kiali console.

- High scalability – The distributed tracing 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 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.

2.1.3. Red Hat OpenShift distributed tracing architecture

Red Hat OpenShift distributed tracing 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 Jaeger project.
  - **Client** (Jaeger client, Tracer, Reporter, instrumented application, client libraries)- The distributed tracing platform clients are language-specific implementations of the OpenTracing API. They can be used to instrument applications for distributed tracing either manually or with a variety of existing open source frameworks, such as Camel (Fuse), Spring Boot (RHOAR), MicroProfile (RHOAR/Thorntail), Wildfly (EAP), and many more, that are already integrated with OpenTracing.
  - **Agent** (Jaeger agent, Server Queue, Processor Workers) - The distributed tracing platform agent is a network daemon that listens for spans sent over User Datagram Protocol (UDP), which it batches and sends to the Collector. The agent is meant to be placed on the same host as the instrumented application. This is typically accomplished by having a sidecar in container environments such as Kubernetes.
  - **Jaeger Collector** (Collector, Queue, Workers) - Similar to the Jaeger agent, the Jaeger Collector receives spans and places them in an internal queue for processing. This allows the Jaeger Collector to return immediately to the client/agent instead of waiting for the span to make its way to the storage.
  - **Storage** (Data Store) - Collectors require a persistent storage backend. Red Hat OpenShift distributed tracing platform has a pluggable mechanism for span storage. Note that for this release, the only supported storage is Elasticsearch.
  - **Query** (Query Service) - Query is a service that retrieves traces from storage.
  - **Ingester** (Ingester Service) - Red Hat OpenShift distributed tracing can use Apache Kafka as a buffer between the Collector and the actual Elasticsearch backing storage. Ingester is a service that reads data from Kafka and writes to the Elasticsearch storage backend.
  - **Jaeger Console** – With the Red Hat OpenShift distributed tracing platform user interface, you can visualize your distributed tracing data. On the Search page, you can find traces and explore details of the spans that make up an individual trace.
- **Red Hat OpenShift distributed tracing data collection**- 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. DISTRIBUTED TRACING INSTALLATION

3.1. INSTALLING DISTRIBUTED TRACING

You can install Red Hat OpenShift distributed tracing on OpenShift Container Platform in either of two ways:

- You can install Red Hat OpenShift distributed tracing as part of Red Hat OpenShift Service Mesh. Distributed tracing is included by default in the Service Mesh installation. To install Red Hat OpenShift distributed tracing as part of a service mesh, follow the Red Hat Service Mesh Installation instructions. You must install Red Hat OpenShift distributed tracing in the same namespace as your service mesh, that is, the **ServiceMeshControlPlane** and the Red Hat OpenShift distributed tracing resources must be in the same namespace.
- If you do not want to install a service mesh, you can use the Red Hat OpenShift distributed tracing Operators to install distributed tracing by itself. To install Red Hat OpenShift distributed tracing without a service mesh, use the following instructions.

3.1.1. Prerequisites

Before you can install Red Hat OpenShift distributed tracing, review the installation activities, and ensure that you meet the prerequisites:

- Possess an active OpenShift Container Platform subscription on your Red Hat account. If you do not have a subscription, contact your sales representative for more information.
- Review the OpenShift Container Platform 4.6 overview .
- Install OpenShift Container Platform 4.6.
  - Install OpenShift Container Platform 4.6 on AWS
  - Install OpenShift Container Platform 4.6 on user-provisioned AWS
  - Install OpenShift Container Platform 4.6 on bare metal
  - Install OpenShift Container Platform 4.6 on vSphere
- Install the version of the OpenShift CLI (**oc**) that matches your OpenShift Container Platform version and add it to your path.
- An account with the **cluster-admin** role.

3.1.2. Red Hat OpenShift distributed tracing installation overview

The steps for installing Red Hat OpenShift distributed tracing are as follows:

- Review the documentation and determine your deployment strategy.
- If your deployment strategy requires persistent storage, install the OpenShift Elasticsearch Operator via the OperatorHub.
- Install the Red Hat OpenShift distributed tracing platform Operator via the OperatorHub.
- Modify the custom resource YAML file to support your deployment strategy.

- Deploy one or more instances of Red Hat OpenShift distributed tracing platform to your OpenShift Container Platform environment.

3.1.3. Installing the OpenShift Elasticsearch Operator

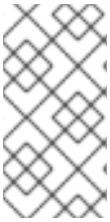
The default Red Hat OpenShift distributed tracing platform deployment uses in-memory storage because it is designed to be installed quickly for those evaluating Red Hat OpenShift distributed tracing, giving demonstrations, or using Red Hat OpenShift distributed tracing platform in a test environment. If you plan to use Red Hat OpenShift distributed tracing platform in production, you must install and configure a persistent storage option, in this case, Elasticsearch.

Prerequisites

- You have access to the OpenShift Container Platform web console.
- You have access to the cluster as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.

WARNING

Do not install Community versions of the Operators. Community Operators are not supported.



NOTE

If you have already installed the OpenShift Elasticsearch Operator as part of OpenShift Logging, you do not need to install the OpenShift Elasticsearch Operator again. The Red Hat OpenShift distributed tracing platform Operator creates the Elasticsearch instance using the installed OpenShift Elasticsearch Operator.

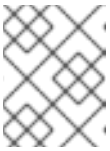
Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.
2. Navigate to **Operators → OperatorHub**.
3. Type **Elasticsearch** into the filter box to locate the OpenShift Elasticsearch Operator.
4. Click the **OpenShift Elasticsearch Operator** provided by Red Hat to display information about the Operator.
5. Click **Install**.
6. On the **Install Operator** page, select the **stable** Update Channel. This automatically updates your Operator as new versions are released.
7. Accept the default **All namespaces on the cluster (default)** This installs the Operator in the default **openshift-operators-redhat** project and makes the Operator available to all projects in the cluster.

NOTE

The Elasticsearch installation requires the **openshift-operators-redhat** namespace for the OpenShift Elasticsearch Operator. The other Red Hat OpenShift distributed tracing Operators are installed in the **openshift-operators** namespace.

- Accept the default **Automatic** approval strategy. By accepting the default, when a new version of this Operator is available, Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without human intervention. If you select **Manual** updates, when a newer version of an Operator is available, OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.



NOTE

The **Manual** approval strategy requires a user with appropriate credentials to approve the Operator install and subscription process.

8. Click **Install**.

9. On the **Installed Operators** page, select the **openshift-operators-redhat** project. Wait until you see that the OpenShift Elasticsearch Operator shows a status of "InstallSucceeded" before continuing.

3.1.4. Installing the Red Hat OpenShift distributed tracing platform Operator

To install Red Hat OpenShift distributed tracing platform, you use the OperatorHub to install the Red Hat OpenShift distributed tracing platform Operator.

By default, the Operator is installed in the **openshift-operators** project.

Prerequisites

- You have access to the OpenShift Container Platform web console.
- You have access to the cluster as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.
- If you require persistent storage, you must also install the OpenShift Elasticsearch Operator before installing the Red Hat OpenShift distributed tracing platform Operator.

WARNING

Do not install Community versions of the Operators. Community Operators are not supported.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.
2. Navigate to **Operators → OperatorHub**.
3. Type **distributing tracing platform** into the filter to locate the Red Hat OpenShift distributed tracing platform Operator.
4. Click the **Red Hat OpenShift distributed tracing platform Operator** provided by Red Hat to display information about the Operator.
5. Click **Install**.
6. On the **Install Operator** page, select the **stable** Update Channel. This automatically updates your Operator as new versions are released.
7. Accept the default **All namespaces on the cluster (default)** This installs the Operator in the default **openshift-operators** project and makes the Operator available to all projects in the cluster.
   - Accept the default **Automatic** approval strategy. By accepting the default, when a new version of this Operator is available, Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without human intervention. If you select **Manual** updates, when a newer version of an Operator is available, OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.

NOTE

The **Manual** approval strategy requires a user with appropriate credentials to approve the Operator install and subscription process.

8. Click **Install**.
9. Navigate to **Operators → Installed Operators**.
10. On the **Installed Operators** page, select the **openshift-operators** project. Wait until you see that the Red Hat OpenShift distributed tracing platform Operator shows a status of "Succeeded" before continuing.

3.1.5. Installing the Red Hat OpenShift distributed tracing data collection Operator

IMPORTANT

The Red Hat OpenShift distributed tracing data collection Operator 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 https://access.redhat.com/support/offerings/techpreview/.

To install Red Hat OpenShift distributed tracing data collection, you use the OperatorHub to install the Red Hat OpenShift distributed tracing data collection Operator.

By default, the Operator is installed in the **openshift-operators** project.

Prerequisites

- You have access to the OpenShift Container Platform web console.
- You have access to the cluster as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.

WARNING

Do not install Community versions of the Operators. Community Operators are not supported.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.
2. Navigate to **Operators → OperatorHub**.
3. Type **distributing tracing data collection** into the filter to locate the Red Hat OpenShift distributed tracing data collection Operator.
4. Click the **Red Hat OpenShift distributed tracing data collection Operator** provided by Red Hat to display information about the Operator.
5. Click **Install**.
6. On the **Install Operator** page, accept the default **stable** Update channel. This automatically updates your Operator as new versions are released.
7. Accept the default **All namespaces on the cluster (default)** This installs the Operator in the default **openshift-operators** project and makes the Operator available to all projects in the cluster.
8. Accept the default **Automatic** approval strategy. By accepting the default, when a new version of this Operator is available, Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without human intervention. If you select **Manual** updates, when a newer version of an Operator is available, OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.

NOTE

The **Manual** approval strategy requires a user with appropriate credentials to approve the Operator install and subscription process.

9. Click **Install**.

10. Navigate to **Operators → Installed Operators**.

11. On the **Installed Operators** page, select the **openshift-operators** project. Wait until you see that the Red Hat OpenShift distributed tracing data collection Operator shows a status of "Succeeded" before continuing.

3.2. CONFIGURING AND DEPLOYING DISTRIBUTED TRACING

The Red Hat OpenShift distributed tracing platform Operator uses a custom resource definition (CRD) file that defines the architecture and configuration settings to be used when creating and deploying the distributed tracing platform resources. You can either install the default configuration or modify the file to better suit your business requirements.

Red Hat OpenShift distributed tracing platform has predefined deployment strategies. You specify a deployment strategy in the custom resource file. When you create a distributed tracing platform instance the Operator uses this configuration file to create the objects necessary for the deployment.

Jaeger custom resource file showing deployment strategy

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: MyConfigFile
spec:
  strategy: production (1)
```

(1) The Red Hat OpenShift distributed tracing platform Operator currently supports the following deployment strategies:

- **allInOne** (Default) - This strategy is intended for development, testing, and demo purposes; it is not intended for production use. The main backend components, Agent, Collector, and Query service, are all packaged into a single executable which is configured, by default. to use in-memory storage.

NOTE

In-memory storage is not persistent, which means that if the distributed tracing platform instance shuts down, restarts, or is replaced, that your trace data will be lost. And in-memory storage cannot be scaled, since each pod has its own memory. For persistent storage, you must use the **production** or **streaming** strategies, which use Elasticsearch as the default storage.

- **production** - The production strategy is intended for production environments, where long term storage of trace data is important, as well as a more scalable and highly available architecture is required. Each of the backend components is therefore deployed separately. The Agent can be injected as a sidecar on the instrumented application. The Query and Collector services are configured with a supported storage type - currently Elasticsearch. Multiple instances of each of these components can be provisioned as required for performance and resilience purposes.

- **streaming** - The streaming strategy is designed to augment the production strategy by providing a streaming capability that effectively sits between the Collector and the

Elasticsearch backend storage. This provides the benefit of reducing the pressure on the backend storage, under high load situations, and enables other trace post-processing capabilities to tap into the real time span data directly from the streaming platform (AMQ Streams/ Kafka).

NOTE

The streaming strategy requires an additional Red Hat subscription for AMQ Streams.

NOTE

The streaming deployment strategy is currently unsupported on IBM Z.

NOTE

There are two ways to install and use Red Hat OpenShift distributed tracing, as part of a service mesh or as a stand alone component. If you have installed distributed tracing as part of Red Hat OpenShift Service Mesh, you can perform basic configuration as part of the ServiceMeshControlPlane but for completely control you should configure a Jaeger CR and then reference your distributed tracing configuration file in the ServiceMeshControlPlane.

3.2.1. Deploying the distributed tracing default strategy from the web console

The custom resource definition (CRD) defines the configuration used when you deploy an instance of Red Hat OpenShift distributed tracing. The default CR is named **jaeger-all-in-one-inmemory** and it is configured with minimal resources to ensure that you can successfully install it on a default OpenShift Container Platform installation. You can use this default configuration to create a Red Hat OpenShift distributed tracing platform instance that uses the **AllInOne** deployment strategy, or you can define your own custom resource file.

NOTE

In-memory storage is not persistent. If the Jaeger pod shuts down, restarts, or is replaced, your trace data will be lost. For persistent storage, you must use the **production** or **streaming** strategies, which use Elasticsearch as the default storage.

Prerequisites

- The Red Hat OpenShift distributed tracing platform Operator has been installed.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.
2. Create a new project, for example **tracing-system**.

NOTE

If you are installing as part of Service Mesh, the distributed tracing resources must be installed in the same namespace as the **ServiceMeshControlPlane** resource, for example **istio-system**.

a. Navigate to **Home → Projects**.

b. Click **Create Project**.

c. Enter **tracing-system** in the **Name** field.

d. Click **Create**.

3. Navigate to **Operators → Installed Operators**.

4. If necessary, select **tracing-system** from the **Project** menu. You may have to wait a few moments for the Operators to be copied to the new project.

5. Click the Red Hat OpenShift distributed tracing platform Operator. On the **Details** tab, under **Provided APIs**, the Operator provides a single link.

6. Under **Jaeger**, click **Create Instance**.

7. On the **Create Jaeger** page, to install using the defaults, click **Create** to create the distributed tracing platform instance.

8. On the **Jaegers** page, click the name of the distributed tracing platform instance, for example, **jaeger-all-in-one-inmemory**.

9. On the **Jaeger Details** page, click the **Resources** tab. Wait until the pod has a status of "Running" before continuing.

3.2.1.1. Deploying the distributed tracing default strategy from the CLI

Follow this procedure to create an instance of distributed tracing platform from the command line.

Prerequisites

- The Red Hat OpenShift distributed tracing platform Operator has been installed and verified.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the OpenShift CLI (**oc**) that matches your OpenShift Container Platform version.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role.

```
$ oc login https://<HOSTNAME>:8443
```

2. Create a new project named **tracing-system**.

```
$ oc new-project tracing-system
```

3. Create a custom resource file named **jaeger.yaml** that contains the following text:

   Example jaeger-all-in-one.yaml

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

4. Run the following command to deploy distributed tracing platform:

```
$ oc create -n tracing-system -f jaeger.yaml
```

5. Run the following command to watch the progress of the pods during the installation process:

```
$ oc get pods -n tracing-system -w
```

   After the installation process has completed, you should see output similar to the following example:

```
NAME                                  READY   STATUS    RESTARTS   AGE
jaeger-all-in-one-inmemory-cdff7897b-qhfdx   2/2     Running   0          24s
```

3.2.2. Deploying the distributed tracing production strategy from the web console

The **production** deployment strategy is intended for production environments that require a more scalable and highly available architecture, and where long-term storage of trace data is important.

Prerequisites

- The OpenShift Elasticsearch Operator has been installed.
- The Red Hat OpenShift distributed tracing platform Operator has been installed.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.
2. Create a new project, for example **tracing-system**.

NOTE

If you are installing as part of Service Mesh, the distributed tracing resources must be installed in the same namespace as the **ServiceMeshControlPlane** resource, for example **istio-system**.

   a. Navigate to **Home → Projects**.

b. Click **Create Project**.

c. Enter **tracing-system** in the **Name** field.

d. Click **Create**.

3. Navigate to **Operators → Installed Operators**.

4. If necessary, select **tracing-system** from the **Project** menu. You may have to wait a few moments for the Operators to be copied to the new project.

5. Click the Red Hat OpenShift distributed tracing platform Operator. On the **Overview** tab, under **Provided APIs**, the Operator provides a single link.

6. Under **Jaeger**, click **Create Instance**.

7. On the **Create Jaeger** page, replace the default **all-in-one** YAML text with your production YAML configuration, for example:

Example jaeger-production.yaml file with Elasticsearch

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-production
  namespace:
spec:
  strategy: production
  ingress:
    security: oauth-proxy
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 3
      redundancyPolicy: SingleRedundancy
    esIndexCleaner:
      enabled: true
      numberOfDays: 7
      schedule: 55 23 * * *
    esRollover:
      schedule: '*/30 * * * *'
```

8. Click **Create** to create the distributed tracing platform instance.

9. On the **Jaegers** page, click the name of the distributed tracing platform instance, for example, **jaeger-prod-elasticsearch**.

10. On the **Jaeger Details** page, click the **Resources** tab. Wait until all the pods have a status of "Running" before continuing.

3.2.2.1. Deploying the distributed tracing production strategy from the CLI

Follow this procedure to create an instance of distributed tracing platform from the command line.

Prerequisites

- The OpenShift Elasticsearch Operator has been installed.
- The Red Hat OpenShift distributed tracing platform Operator has been installed.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the OpenShift CLI (**oc**) that matches your OpenShift Container Platform version.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role.

```
$ oc login https://<HOSTNAME>:8443
```

2. Create a new project named **tracing-system**.

```
$ oc new-project tracing-system
```

3. Create a custom resource file named **jaeger-production.yaml** that contains the text of the example file in the previous procedure.

4. Run the following command to deploy distributed tracing platform:

```
$ oc create -n tracing-system -f jaeger-production.yaml
```

5. Run the following command to watch the progress of the pods during the installation process:

```
$ oc get pods -n tracing-system -w
```

After the installation process has completed, you should see output similar to the following example:

```
NAME                                                  READY   STATUS    RESTARTS   AGE
elasticsearch-cdm-jaegersystemjaegerproduction-1-6676cf568gwhlw   2/2     Running   0
10m
elasticsearch-cdm-jaegersystemjaegerproduction-2-bcd4c8bf5l6g6w   2/2     Running   0
10m
elasticsearch-cdm-jaegersystemjaegerproduction-3-844d6d9694hhst   2/2     Running   0
10m
jaeger-production-collector-94cd847d-jwjlj                1/1     Running   3          8m32s
jaeger-production-query-5cbfbd499d-tv8zf                  3/3     Running   3          8m32s
```

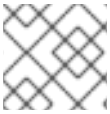
3.2.3. Deploying the distributed tracing streaming strategy from the web console

The **streaming** deployment strategy is intended for production environments that require a more scalable and highly available architecture, and where long-term storage of trace data is important.

The **streaming** strategy provides a streaming capability that sits between the Collector and the Elasticsearch storage. This reduces the pressure on the storage under high load situations, and enables other trace post-processing capabilities to tap into the real-time span data directly from the Kafka streaming platform.

NOTE

The streaming strategy requires an additional Red Hat subscription for AMQ Streams. If you do not have an AMQ Streams subscription, contact your sales representative for more information.



NOTE

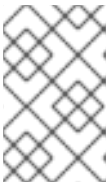
The streaming deployment strategy is currently unsupported on IBM Z.

Prerequisites

- The AMQ Streams Operator has been installed. If using version 1.4.0 or higher you can use self-provisioning. Otherwise you must create the Kafka instance.
- The Red Hat OpenShift distributed tracing platform Operator has been installed.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.
2. Create a new project, for example **tracing-system**.



NOTE

If you are installing as part of Service Mesh, the distributed tracing resources must be installed in the same namespace as the **ServiceMeshControlPlane** resource, for example **istio-system**.

   a. Navigate to **Home → Projects**.
   b. Click **Create Project**.
   c. Enter **tracing-system** in the **Name** field.
   d. Click **Create**.

3. Navigate to **Operators → Installed Operators**.
4. If necessary, select **tracing-system** from the **Project** menu. You may have to wait a few moments for the Operators to be copied to the new project.
5. Click the Red Hat OpenShift distributed tracing platform Operator. On the **Overview** tab, under **Provided APIs**, the Operator provides a single link.
6. Under **Jaeger**, click **Create Instance**.
7. On the **Create Jaeger** page, replace the default **all-in-one** YAML text with your streaming YAML configuration, for example:

Example jaeger-streaming.yaml file

```
-
```

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-streaming
spec:
  strategy: streaming
  collector:
    options:
      kafka:
        producer:
          topic: jaeger-spans
          #Note: If brokers are not defined,AMQStreams 1.4.0+ will self-provision Kafka.
          brokers: my-cluster-kafka-brokers.kafka:9092
  storage:
    type: elasticsearch
  ingester:
    options:
      kafka:
        consumer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
```

1. Click **Create** to create the distributed tracing platform instance.
2. On the **Jaegers** page, click the name of the distributed tracing platform instance, for example, **jaeger-streaming**.
3. On the **Jaeger Details** page, click the **Resources** tab. Wait until all the pods have a status of "Running" before continuing.

3.2.3.1. Deploying the distributed tracing streaming strategy from the CLI

Follow this procedure to create an instance of distributed tracing platform from the command line.

Prerequisites

- The AMQ Streams Operator has been installed. If using version 1.4.0 or higher you can use self-provisioning. Otherwise you must create the Kafka instance.
- The Red Hat OpenShift distributed tracing platform Operator has been installed.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the OpenShift CLI (**oc**) that matches your OpenShift Container Platform version.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role.

```
$ oc login https://<HOSTNAME>:8443
```

2. Create a new project named **tracing-system**.

```
$ oc new-project tracing-system
```

3. Create a custom resource file named **jaeger-streaming.yaml** that contains the text of the example file in the previous procedure.

4. Run the following command to deploy Jaeger:

```
$ oc create -n tracing-system -f jaeger-streaming.yaml
```

5. Run the following command to watch the progress of the pods during the installation process:

```
$ oc get pods -n tracing-system -w
```

After the installation process has completed, you should see output similar to the following example:

```
NAME                                                  READY   STATUS    RESTARTS   AGE
elasticsearch-cdm-jaegersystemjaegerstreaming-1-697b66d6fcztcnn   2/2     Running   0
5m40s
elasticsearch-cdm-jaegersystemjaegerstreaming-2-5f4b95c78b9gckz   2/2     Running   0
5m37s
elasticsearch-cdm-jaegersystemjaegerstreaming-3-7b6d964576nnz97   2/2     Running   0
5m5s
jaeger-streaming-collector-6f6db7f99f-rtcfm                       1/1     Running   0          80s
jaeger-streaming-entity-operator-6b6d67cc99-4lm9q                 3/3     Running   2
2m18s
jaeger-streaming-ingester-7d479847f8-5h8kc                        1/1     Running   0          80s
jaeger-streaming-kafka-0                                          2/2     Running   0          3m1s
jaeger-streaming-query-65bf5bb854-ncnc7                           3/3     Running   0          80s
jaeger-streaming-zookeeper-0                                      2/2     Running   0          3m39s
```

3.2.4. Validating your deployment

3.2.4.1. Accessing the Jaeger console

To access the Jaeger console you must have either Red Hat OpenShift Service Mesh or Red Hat OpenShift distributed tracing installed, and Red Hat OpenShift distributed tracing platform installed, configured, and deployed.

The installation process creates a route to access the Jaeger console.

If you know the URL for the Jaeger console, you can access it directly. If you do not know the URL, use the following directions.

Procedure from OpenShift console

1. Log in to the OpenShift Container Platform web console as a user with cluster-admin rights. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.

2. Navigate to **Networking → Routes**.

3. On the **Routes** page, select the control plane project, for example **tracing-system**, from the **Namespace** menu.
   The **Location** column displays the linked address for each route.

4. If necessary, use the filter to find the **jaeger** route. Click the route **Location** to launch the console.

5. Click **Log In With OpenShift**

Procedure from the CLI

1. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.

```
$ oc login https://<HOSTNAME>:6443
```

2. To query for details of the route using the command line, enter the following command. In this example, **tracing-system** is the control plane namespace.

```
$ export JAEGER_URL=$(oc get route -n tracing-system jaeger -o jsonpath='{.spec.host}')
```

3. Launch a browser and navigate to **https://<JAEGER_URL>**, where **<JAEGER_URL>** is the route that you discovered in the previous step.

4. Log in using the same user name and password that you use to access the OpenShift Container Platform console.

5. If you have added services to the service mesh and have generated traces, you can use the filters and **Find Traces** button to search your trace data.
   If you are validating the console installation, there is no trace data to display.

3.2.5. Customizing your deployment

3.2.5.1. Deployment best practices

- Red Hat OpenShift distributed tracing instance names must be unique. If you want to have multiple Red Hat OpenShift distributed tracing platform instances and are using sidecar injected agents, then the Red Hat OpenShift distributed tracing platform instances should have unique names, and the injection annotation should explicitly specify the Red Hat OpenShift distributed tracing platform instance name the tracing data should be reported to.

- If you have a multitenant implementation and tenants are separated by namespaces, deploy a Red Hat OpenShift distributed tracing platform instance to each tenant namespace.

  - Agent as a daemonset is not supported for multitenant installations or Red Hat OpenShift Dedicated. Agent as a sidecar is the only supported configuration for these use cases.

- If you are installing distributed tracing as part of Red Hat OpenShift Service Mesh, the distributed tracing resources must be installed in the same namespace as the **ServiceMeshControlPlane** resource.

3.2.5.2. Distributed tracing default configuration options

The Jaeger custom resource (CR) defines the architecture and settings to be used when creating the distributed tracing platform resources. You can modify these parameters to customize your distributed tracing platform implementation to your business needs.

Jaeger generic YAML example

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: name
spec:
  strategy: <deployment_strategy>
  allInOne:
    options: {}
    resources: {}
  agent:
    options: {}
    resources: {}
  collector:
    options: {}
    resources: {}
  sampling:
    options: {}
  storage:
    type:
    options: {}
  query:
    options: {}
    resources: {}
  ingester:
    options: {}
    resources: {}
  options: {}
```

Table 3.1. Jaeger parameters

Parameter	Description	Values	Default value
apiVersion:		API version to use when creating the object.	**jaegertracing.io/v1**
jaegertracing.io/v1	**kind:**	Defines the kind of Kubernetes object to create.	**jaeger**
	metadata:	Data that helps uniquely identify the object, including a **name** string, **UID**, and optional **namespace**.	

Parameter	Description	Values	Default value
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.	The name of your distributed tracing platform instance.
jaeger-all-in-one-inmemory	**spec:**	Specification for the object to be created.	Contains all of the configuration parameters for your distributed tracing platform instance. When a common definition for all Jaeger components is required, it is defined under the **spec** node. When the definition relates to an individual component, it is placed under the **spec/<component>** node.
N/A	**strategy:**	Jaeger deployment strategy	**allInOne**, **production**, or **streaming**
allInOne	**allInOne:**	Because the **allInOne** image deploys the Agent, Collector, Query, Ingester, and Jaeger UI in a single pod, configuration for this deployment must nest component configuration under the **allInOne** parameter.	
	agent:	Configuration options that define the Agent.	
	collector:	Configuration options that define the Jaeger Collector.	
	sampling:	Configuration options that define the sampling strategies for tracing.	

Parameter	Description	Values	Default value
	storage:	Configuration options that define the storage. All storage-related options must be placed under **storage**, rather than under the **allInOne** or other component options.	
	query:	Configuration options that define the Query service.	
	ingester:	Configuration options that define the Ingester service.	

The following example YAML is the minimum required to create a Red Hat OpenShift distributed tracing platform deployment using the default settings.

Example minimum required dist-tracing-all-in-one.yaml

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

3.2.5.3. Jaeger Collector configuration options

The Jaeger Collector is the component responsible for receiving the spans that were captured by the tracer and writing them to persistent Elasticsearch storage when using the **production** strategy, or to AMQ Streams when using the **streaming** strategy.

The Collectors are stateless and thus many instances of Jaeger Collector can be run in parallel. Collectors require almost no configuration, except for the location of the Elasticsearch cluster.

Table 3.2. Parameters used by the Operator to define the Jaeger Collector

Parameter	Description	Values
`collector:` `  replicas:`	Specifies the number of Collector replicas to create.	Integer, for example, **5**

Table 3.3. Configuration parameters passed to the Collector

Parameter	Description	Values

`spec:`
`collector:`
`options: {}`
	Configuration options that define the Jaeger Collector.	

`options:`
`collector:`
`num-workers:`
	The number of workers pulling from the queue.	Integer, for example, **50**

`options:`
`collector:`
`queue-size:`
	The size of the Collector queue.	Integer, for example, **2000**

`options:`
`kafka:`
`producer:`
`topic: jaeger-spans`
	The **topic** parameter identifies the Kafka configuration used by the Collector to produce the messages, and the Ingester to consume the messages.	Label for the producer.

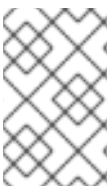
`options:`
`kafka:`
`producer:`
`brokers: my-cluster-kafka-brokers.kafka:9092`
	Identifies the Kafka configuration used by the Collector to produce the messages. If brokers are not specified, and you have AMQ Streams 1.4.0+ installed, the Red Hat OpenShift distributed tracing platform Operator will self-provision Kafka.	

`options:` `log-level:`	Logging level for the Collector.	Possible values: **debug**, **info**, **warn**, **error**, **fatal**, **panic**.

3.2.5.4. Distributed tracing sampling configuration options

The Red Hat OpenShift distributed tracing platform Operator can be used to define sampling strategies that will be supplied to tracers that have been configured to use a remote sampler.

While all traces are generated, only a few are sampled. Sampling a trace marks the trace for further processing and storage.



NOTE

This is not relevant if a trace was started by the Envoy proxy, as the sampling decision is made there. The Jaeger sampling decision is only relevant when the trace is started by an application using the client.

When a service receives a request that contains no trace context, the client starts a new trace, assigns it a random trace ID, and makes a sampling decision based on the currently installed sampling strategy. The sampling decision propagates to all subsequent requests in the trace so that other services are not making the sampling decision again.

distributed tracing platform libraries support the following samplers:

- **Probabilistic** - The sampler makes a random sampling decision with the probability of sampling equal to the value of the **sampling.param** property. For example, using **sampling.param=0.1** samples approximately 1 in 10 traces.
- **Rate Limiting** - The sampler uses a leaky bucket rate limiter to ensure that traces are sampled with a certain constant rate. For example, using **sampling.param=2.0** samples requests with the rate of 2 traces per second.

Table 3.4. Jaeger sampling options

Parameter	Description	Values	Default value

`spec:`
`sampling:`
`options: {}`
`default_strategy:`
 	
`service_strategy:`
	Configuration options that define the sampling strategies for tracing.		If you do not provide configuration, the Collectors will return the default probabilistic sampling policy with 0.001 (0.1%) probability for all services.

`default_strategy:`
`type:`
`service_strategy:`
`type:`
	Sampling strategy to use. See descriptions above.	Valid values are **probabilistic**, and **ratelimiting**.	**probabilistic**

`default_strategy:`
`param:`
`service_strategy:`
`param:`
	Parameters for the selected sampling strategy.	Decimal and integer values (0, .1, 1, 10)	1

This example defines a default sampling strategy that is probabilistic, with a 50% chance of the trace instances being sampled.

Probabilistic sampling example

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: with-sampling
spec:
  sampling:
    options:
      default_strategy:
```

```
      type: probabilistic
      param: 0.5
    service_strategies:
      - service: alpha
        type: probabilistic
        param: 0.8
        operation_strategies:
          - operation: op1
            type: probabilistic
            param: 0.2
          - operation: op2
            type: probabilistic
            param: 0.4
      - service: beta
        type: ratelimiting
        param: 5
```

If there are no user-supplied configurations, the distributed tracing platform uses the following settings:

Default sampling

```
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 1
```

3.2.5.5. Distributed tracing storage configuration options

You configure storage for the Collector, Ingester, and Query services under **spec.storage**. Multiple instances of each of these components can be provisioned as required for performance and resilience purposes.

Table 3.5. General storage parameters used by the Red Hat OpenShift distributed tracing platform Operator to define distributed tracing storage

Parameter	Description	Values	Default value

Parameter	Description	Values	Default value
spec: storage: type:	Type of storage to use for the deployment.	**memory** or **elasticsearch**. Memory storage is only appropriate for development, testing, demonstrations, and proof of concept environments as the data does not persist if the pod is shut down. For production environments distributed tracing platform supports Elasticsearch for persistent storage.	**memory**
storage: secretname:	Name of the secret, for example **tracing-secret**.		N/A
storage: options: {}	Configuration options that define the storage.		

Table 3.6. Elasticsearch index cleaner parameters

Parameter	Description	Values	Default value
storage: esIndexCleaner: enabled:	When using Elasticsearch storage, by default a job is created to clean old traces from the index. This parameter enables or disables the index cleaner job.	**true**/ **false**	**true**
storage: esIndexCleaner: numberOfDays:	Number of days to wait before deleting an index.	Integer value	**7**
storage: esIndexCleaner: schedule:	Defines the schedule for how often to clean the Elasticsearch index.	Cron expression	"55 23 * * *"

3.2.5.5.1. Auto-provisioning an Elasticsearch instance

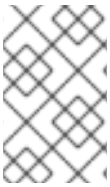
When you deploy a Jaeger custom resource, the Red Hat OpenShift distributed tracing platform Operator uses the OpenShift Elasticsearch Operator to create an Elasticsearch cluster based on the configuration provided in the **storage** section of the custom resource file. The Red Hat OpenShift distributed tracing platform Operator will provision Elasticsearch if the following configurations are set:

- **spec.storage:type** is set to **elasticsearch**
- **spec.storage.elasticsearch.doNotProvision** set to **false**
- **spec.storage.options.es.server-urls** is not defined, that is, there is no connection to an Elasticsearch instance that was not provisioned by the Red Hat Elasticsearch Operator.

When provisioning Elasticsearch, the Red Hat OpenShift distributed tracing platform Operator sets the Elasticsearch custom resource **name** to the value of **spec.storage.elasticsearch.name** from the Jaeger custom resource. If you do not specify a value for **spec.storage.elasticsearch.name**, the Operator uses **elasticsearch**.

Restrictions

- You can have only one distributed tracing platform with self-provisioned Elasticsearch instance per namespace. The Elasticsearch cluster is meant to be dedicated for a single distributed tracing platform instance.
- There can be only one Elasticsearch per namespace.



NOTE

If you already have installed Elasticsearch as part of OpenShift Logging, the Red Hat OpenShift distributed tracing platform Operator can use the installed OpenShift Elasticsearch Operator to provision storage.

The following configuration parameters are for a *self-provisioned* Elasticsearch instance, that is an instance created by the Red Hat OpenShift distributed tracing platform Operator using the OpenShift Elasticsearch Operator. You specify configuration options for self-provisioned Elasticsearch under **spec:storage:elasticsearch** in your configuration file.

Table 3.7. Elasticsearch resource configuration parameters

Parameter	Description	Values	Default value

`elasticsearch:`
`  properties:`
`  doNotProvision:`
	Use to specify whether or not an Elasticsearch instance should be provisioned by the Red Hat OpenShift distributed tracing platform Operator.	**true**/**false**	**true**

Parameter	Description	Values	Default value
elasticsearch: properties: name:	Name of the Elasticsearch instance. The Red Hat OpenShift distributed tracing platform Operator uses the Elasticsearch instance specified in this parameter to connect to Elasticsearch.	string	**elasticsearch**
elasticsearch: nodeCount:	Number of Elasticsearch nodes. For high availability use at least 3 nodes. Do not use 2 nodes as “split brain” problem can happen.	Integer value. For example, Proof of concept = 1, Minimum deployment =3	3
elasticsearch: resources: requests: cpu:	Number of central processing units for requests, based on your environment’s configuration.	Specified in cores or millicores, for example, 200m, 0.5, 1. For example, Proof of concept = 500m, Minimum deployment =1	1
elasticsearch: resources: requests: memory:	Available memory for requests, based on your environment’s configuration.	Specified in bytes, for example, 200Ki, 50Mi, 5Gi. For example, Proof of concept = 1Gi, Minimum deployment = 16Gi*	16Gi
elasticsearch: resources: limits: cpu:	Limit on number of central processing units, based on your environment’s configuration.	Specified in cores or millicores, for example, 200m, 0.5, 1. For example, Proof of concept = 500m, Minimum deployment =1	
elasticsearch: resources: limits: memory:	Available memory limit based on your environment’s configuration.	Specified in bytes, for example, 200Ki, 50Mi, 5Gi. For example, Proof of concept = 1Gi, Minimum deployment = 16Gi*	

Parameter	Description	Values	Default value
elasticsearch: redundancyPolicy:	Data replication policy defines how Elasticsearch shards are replicated across data nodes in the cluster. If not specified, the Red Hat OpenShift distributed tracing platform Operator automatically determines the most appropriate replication based on number of nodes.	**ZeroRedundancy**(no replica shards), **SingleRedundancy**(one replica shard), **MultipleRedundancy** (each index is spread over half of the Data nodes), **FullRedundancy** (each index is fully replicated on every Data node in the cluster).	
elasticsearch: useCertManagement:	Use to specify whether or not distributed tracing platform should use the certificate management feature of the Red Hat Elasticsearch Operator. This feature was added to logging subsystem for Red Hat OpenShift 5.2 in OpenShift Container Platform 4.7 and is the preferred setting for new Jaeger deployments.	**true**/**false**	**true**
	*Each Elasticsearch node can operate with a lower memory setting though this is NOT recommended for production deployments. For production use, you should have no less than 16Gi allocated to each pod by default, but preferably allocate as much as you can, up to 64Gi per pod.		

Production storage example

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 3
      resources:
        requests:
          cpu: 1
```

```
        memory: 16Gi
      limits:
        memory: 16Gi
```

Storage example with persistent storage:

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 1
      storage: 1
        storageClassName: gp2
        size: 5Gi
      resources:
        requests:
          cpu: 200m
          memory: 4Gi
        limits:
          memory: 4Gi
      redundancyPolicy: ZeroRedundancy
```

1 Persistent storage configuration. In this case AWS **gp2** with **5Gi** size. When no value is specified, distributed tracing platform uses **emptyDir**. The OpenShift Elasticsearch Operator provisions **PersistentVolumeClaim** and **PersistentVolume** which are not removed with distributed tracing platform instance. You can mount the same volumes if you create a distributed tracing platform instance with the same name and namespace.

3.2.5.5.2. Connecting to an existing Elasticsearch instance

You can use an existing Elasticsearch cluster for storage with distributed tracing. An existing Elasticsearch cluster, also known as an *external* Elasticsearch instance, is an instance that was not installed by the Red Hat OpenShift distributed tracing platform Operator or by the Red Hat Elasticsearch Operator.

When you deploy a Jaeger custom resource, the Red Hat OpenShift distributed tracing platform Operator will not provision Elasticsearch if the following configurations are set:

- **spec.storage.elasticsearch.doNotProvision** set to **true**
- **spec.storage.options.es.server-urls** has a value
- **spec.storage.elasticsearch.name** has a value, or if the Elasticsearch instance name is **elasticsearch**.

The Red Hat OpenShift distributed tracing platform Operator uses the Elasticsearch instance specified in **spec.storage.elasticsearch.name** to connect to Elasticsearch.

Restrictions

- You cannot share or reuse a OpenShift Container Platform logging Elasticsearch instance with distributed tracing platform. The Elasticsearch cluster is meant to be dedicated for a single distributed tracing platform instance.

NOTE

Red Hat does not provide support for your external Elasticsearch instance. You can review the tested integrations matrix on the Customer Portal.

The following configuration parameters are for an already existing Elasticsearch instance, also known as an *external* Elasticsearch instance. In this case, you specify configuration options for Elasticsearch under **spec:storage:options:es** in your custom resource file.

Table 3.8. General ES configuration parameters

Parameter	Description	Values	Default value
es: server-urls:	URL of the Elasticsearch instance.	The fully-qualified domain name of the Elasticsearch server.	**http://elasticsearch.<namespace>.svc:9200**
es: max-doc-count:	The maximum document count to return from an Elasticsearch query. This will also apply to aggregations. If you set both **es.max-doc-count** and **es.max-num-spans**, Elasticsearch will use the smaller value of the two.		10000
es: max-num-spans:	[**Deprecated** - Will be removed in a future release, use **es.max-doc-count** instead.] The maximum number of spans to fetch at a time, per query, in Elasticsearch. If you set both **es.max-num-spans** and **es.max-doc-count**, Elasticsearch will use the smaller value of the two.		10000
es: max-span-age:	The maximum lookback for spans in Elasticsearch.		72h0m0s

Parameter	Description	Values	Default value
es: sniffer:	The sniffer configuration for Elasticsearch. The client uses the sniffing process to find all nodes automatically. Disabled by default.	**true**/ **false**	**false**
es: sniffer-tls-enabled:	Option to enable TLS when sniffing an Elasticsearch Cluster. The client uses the sniffing process to find all nodes automatically. Disabled by default	**true**/ **false**	**false**
es: timeout:	Timeout used for queries. When set to zero there is no timeout.		0s
es: username:	The username required by Elasticsearch. The basic authentication also loads CA if it is specified. See also **es.password**.		
es: password:	The password required by Elasticsearch. See also, **es.username**.		
es: version:	The major Elasticsearch version. If not specified, the value will be auto-detected from Elasticsearch.		0

Table 3.9. ES data replication parameters

Parameter	Description	Values	Default value
es: num-replicas:	The number of replicas per index in Elasticsearch.		1

Parameter	Description	Values	Default value
`es:` `  num-shards:`	The number of shards per index in Elasticsearch.		5

Table 3.10. ES index configuration parameters

Parameter	Description	Values	Default value
`es:` `  create-index-templates:`	Automatically create index templates at application startup when set to **true**. When templates are installed manually, set to **false**.	**true**/ **false**	**true**
`es:` `  index-prefix:`	Optional prefix for distributed tracing platform indices. For example, setting this to "production" creates indices named "production-tracing-*".		

Table 3.11. ES bulk processor configuration parameters

Parameter	Description	Values	Default value

`es:`
`  bulk:`
`  actions:`
	The number of requests that can be added to the queue before the bulk processor decides to commit updates to disk.		1000

`es:`
`  bulk:`
`  flush-interval:`
	A **time.Duration** after which bulk requests are committed, regardless of other thresholds. To disable the bulk processor flush interval, set this to zero.		200ms

Parameter	Description	Values	Default value
es: bulk: size:	The number of bytes that the bulk requests can take up before the bulk processor decides to commit updates to disk.		5000000
es: bulk: workers:	The number of workers that are able to receive and commit bulk requests to Elasticsearch.		1

Table 3.12. ES TLS configuration parameters

Parameter	Description	Values	Default value
es: tls: ca:	Path to a TLS Certification Authority (CA) file used to verify the remote servers.		Will use the system truststore by default.
es: tls: cert:	Path to a TLS Certificate file, used to identify this process to the remote servers.		
es: tls: enabled:	Enable transport layer security (TLS) when talking to the remote servers. Disabled by default.	**true**/ **false**	**false**
es: tls: key:	Path to a TLS Private Key file, used to identify this process to the remote servers.		
es: tls: server-name:	Override the expected TLS server name in the certificate of the remote servers.		

Parameter	Description	Values	Default value
es: token-file:	Path to a file containing the bearer token. This flag also loads the Certification Authority (CA) file if it is specified.		

Table 3.13. ES archive configuration parameters

Parameter	Description	Values	Default value
es-archive: bulk: actions:	The number of requests that can be added to the queue before the bulk processor decides to commit updates to disk.		0
es-archive: bulk: flush-interval:	A **time.Duration** after which bulk requests are committed, regardless of other thresholds. To disable the bulk processor flush interval, set this to zero.		0s
es-archive: bulk: size:	The number of bytes that the bulk requests can take up before the bulk processor decides to commit updates to disk.		0
es-archive: bulk: workers:	The number of workers that are able to receive and commit bulk requests to Elasticsearch.		0
es-archive: create-index-templates:	Automatically create index templates at application startup when set to **true**. When templates are installed manually, set to **false**.	**true**/ **false**	**false**
es-archive: enabled:	Enable extra storage.	**true**/ **false**	**false**

Parameter	Description	Values	Default value
es-archive: index-prefix:	Optional prefix for distributed tracing platform indices. For example, setting this to "production" creates indices named "production-tracing-*".		
es-archive: max-doc-count:	The maximum document count to return from an Elasticsearch query. This will also apply to aggregations.		0
es-archive: max-num-spans:	[**Deprecated** - Will be removed in a future release, use **es-archive.max-doc-count** instead.] The maximum number of spans to fetch at a time, per query, in Elasticsearch.		0
es-archive: max-span-age:	The maximum lookback for spans in Elasticsearch.		0s
es-archive: num-replicas:	The number of replicas per index in Elasticsearch.		0
es-archive: num-shards:	The number of shards per index in Elasticsearch.		0
es-archive: password:	The password required by Elasticsearch. See also, **es.username**.		
es-archive: server-urls:	The comma-separated list of Elasticsearch servers. Must be specified as fully qualified URLs, for example, **http://localhost:9200**.		

Parameter	Description	Values	Default value
es-archive: sniffer:	The sniffer configuration for Elasticsearch. The client uses the sniffing process to find all nodes automatically. Disabled by default.	**true**/ **false**	**false**
es-archive: sniffer-tls-enabled:	Option to enable TLS when sniffing an Elasticsearch Cluster. The client uses the sniffing process to find all nodes automatically. Disabled by default.	**true**/ **false**	**false**
es-archive: timeout:	Timeout used for queries. When set to zero there is no timeout.		0s
es-archive: tls: ca:	Path to a TLS Certification Authority (CA) file used to verify the remote servers.		Will use the system truststore by default.
es-archive: tls: cert:	Path to a TLS Certificate file, used to identify this process to the remote servers.		
es-archive: tls: enabled:	Enable transport layer security (TLS) when talking to the remote servers. Disabled by default.	**true**/ **false**	**false**

Parameter	Description	Values	Default value
es-archive: tls: key:	Path to a TLS Private Key file, used to identify this process to the remote servers.		
es-archive: tls: server-name:	Override the expected TLS server name in the certificate of the remote servers.		
es-archive: token-file:	Path to a file containing the bearer token. This flag also loads the Certification Authority (CA) file if it is specified.		
es-archive: username:	The username required by Elasticsearch. The basic authentication also loads CA if it is specified. See also **es-archive.password**.		
es-archive: version:	The major Elasticsearch version. If not specified, the value will be auto-detected from Elasticsearch.		0

Storage example with volume mounts

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200
        index-prefix: my-prefix
        tls:
          ca: /es/certificates/ca.crt
    secretName: tracing-secret
  volumeMounts:
```

```
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public
```

The following example shows a Jaeger CR using an external Elasticsearch cluster with TLS CA certificate mounted from a volume and user/password stored in a secret.

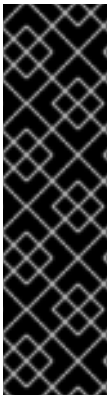
External Elasticsearch example:

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200 (1)
        index-prefix: my-prefix
        tls: (2)
          ca: /es/certificates/ca.crt
    secretName: tracing-secret (3)
  volumeMounts: (4)
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public
```

1. URL to Elasticsearch service running in default namespace.
2. TLS configuration. In this case only CA certificate, but it can also contain es.tls.key and es.tls.cert when using mutual TLS.
3. Secret which defines environment variables ES_PASSWORD and ES_USERNAME. Created by kubectl create secret generic tracing-secret --from-literal=ES_PASSWORD=changeme --from-literal=ES_USERNAME=elastic
4. Volume mounts and volumes which are mounted into all storage components.

3.2.5.6. Managing certificates with Elasticsearch

You can create and manage certificates using the Red Hat Elasticsearch Operator. Managing certificates using the Red Hat Elasticsearch Operator also lets you use a single Elasticsearch cluster with multiple Jaeger Collectors.



IMPORTANT

Managing certificates with Elasticsearch 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 https://access.redhat.com/support/offerings/techpreview/.

Starting with version 2.4, the Red Hat OpenShift distributed tracing platform Operator delegates certificate creation to the Red Hat Elasticsearch Operator by using the following annotations in the Elasticsearch custom resource:

- **logging.openshift.io/elasticsearch-cert-management: "true"**
- **logging.openshift.io/elasticsearch-cert.jaeger-<shared-es-node-name>: "user.jaeger"**
- **logging.openshift.io/elasticsearch-cert.curator-<shared-es-node-name>: "system.logging.curator"**

Where the **<shared-es-node-name>** is the name of the Elasticsearch node. For example, if you create an Elasticsearch node named **custom-es**, your custom resource might look like the following example.

Example Elasticsearch CR showing annotations

```
apiVersion: logging.openshift.io/v1
kind: Elasticsearch
metadata:
  annotations:
    logging.openshift.io/elasticsearch-cert-management: "true"
    logging.openshift.io/elasticsearch-cert.jaeger-custom-es: "user.jaeger"
    logging.openshift.io/elasticsearch-cert.curator-custom-es: "system.logging.curator"
  name: custom-es
spec:
  managementState: Managed
  nodeSpec:
    resources:
      limits:
        memory: 16Gi
      requests:
        cpu: 1
        memory: 16Gi
  nodes:
    - nodeCount: 3
      proxyResources: {}
      resources: {}
      roles:
        - master
        - client
        - data
      storage: {}
  redundancyPolicy: ZeroRedundancy
```

Prerequisites

- OpenShift Container Platform 4.7
- logging subsystem for Red Hat OpenShift 5.2
- The Elasticsearch node and the Jaeger instances must be deployed in the same namespace. For example, **tracing-system**.

You enable certificate management by setting **spec.storage.elasticsearch.useCertManagement** to **true** in the Jaeger custom resource.

Example showing useCertManagement

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      name: custom-es
      doNotProvision: true
      useCertManagement: true
```

The Red Hat OpenShift distributed tracing platform Operator sets the Elasticsearch custom resource **name** to the value of **spec.storage.elasticsearch.name** from the Jaeger custom resource when provisioning Elasticsearch.

The certificates are provisioned by the Red Hat Elasticsearch Operator and the Red Hat OpenShift distributed tracing platform Operator injects the certificates.

3.2.5.7. Query configuration options

Query is a service that retrieves traces from storage and hosts the user interface to display them.

Table 3.14. Parameters used by the Red Hat OpenShift distributed tracing platform Operator to define Query

Parameter	Description	Values	Default value

`spec:`
`  query:`
`  replicas:`
	Specifies the number of Query replicas to create.	Integer, for example, **2**	

Table 3.15. Configuration parameters passed to Query

Parameter	Description	Values	Default value

`spec:`
`  query:`
`  options: {}`
	Configuration options that define the Query service.		

`options:` `  log-level:`	Logging level for Query.	Possible values: **debug**, **info**, **warn**, **error**, **fatal**, **panic**.	

`options:`
`  query:`
`  base-path:`
	The base path for all jaeger-query HTTP routes can be set to a non-root value, for example, **/jaeger** would cause all UI URLs to start with **/jaeger**. This can be useful when running jaeger-query behind a reverse proxy.	/<path>	

Sample Query configuration

```
apiVersion: jaegertracing.io/v1
kind: "Jaeger"
metadata:
  name: "my-jaeger"
spec:
  strategy: allInOne
  allInOne:
    options:
      log-level: debug
      query:
        base-path: /jaeger
```

3.2.5.8. Ingester configuration options

Ingester is a service that reads from a Kafka topic and writes to the Elasticsearch storage backend. If you are using the **allInOne** or **production** deployment strategies, you do not need to configure the Ingester service.

Table 3.16. Jaeger parameters passed to the Ingester

Parameter	Description	Values

`spec:`
`  ingester:`
`  options: {}`
	Configuration options that define the Ingester service.	

`options:` `  deadlockInterval:`	Specifies the interval, in seconds or minutes, that the Ingester must wait for a message before terminating. The deadlock interval is disabled by default (set to **0**), to avoid terminating the Ingester when no messages arrive during system initialization.	Minutes and seconds, for example, **1m0s**. Default value is **0**.

`options:`
`  kafka:`
`  consumer:`
`  topic:`
	The **topic** parameter identifies the Kafka configuration used by the collector to produce the messages, and the Ingester to consume the messages.	Label for the consumer. For example, **jaeger-spans**.

`options:`
`  kafka:`
`  consumer:`
`  brokers:`
	Identifies the Kafka configuration used by the Ingester to consume the messages.	Label for the broker, for example, **my-cluster-kafka-brokers.kafka:9092**.

`options:` `  log-level:`	Logging level for the Ingester.	Possible values: **debug**, **info**, **warn**, **error**, **fatal**, **dpanic**, **panic**.

Streaming Collector and Ingester example

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-streaming
spec:
  strategy: streaming
  collector:
    options:
      kafka:
        producer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
  ingester:
    options:
      kafka:
        consumer:
```

```
        topic: jaeger-spans
        brokers: my-cluster-kafka-brokers.kafka:9092
    ingester:
      deadlockInterval: 5
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: http://elasticsearch:9200
```

3.2.6. Injecting sidecars

Red Hat OpenShift distributed tracing platform relies on a proxy sidecar within the application's pod to provide the agent. The Red Hat OpenShift distributed tracing platform Operator can inject Agent sidecars into Deployment workloads. You can enable automatic sidecar injection or manage it manually.

3.2.6.1. Automatically injecting sidecars

The Red Hat OpenShift distributed tracing platform Operator can inject Jaeger Agent sidecars into Deployment workloads. To enable automatic injection of sidecars, add the **sidecar.jaegertracing.io/inject** annotation set to either the string **true** or to the distributed tracing platform instance name that is returned by running **$ oc get jaegers**. When you specify **true**, there should be only a single distributed tracing platform instance for the same namespace as the deployment, otherwise, the Operator cannot determine which distributed tracing platform instance to use. A specific distributed tracing platform instance name on a deployment has a higher precedence than **true** applied on its namespace.

The following snippet shows a simple application that will inject a sidecar, with the agent pointing to the single distributed tracing platform instance available in the same namespace:

Automatic sidecar injection example

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    "sidecar.jaegertracing.io/inject": "true" (1)
spec:
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: acme/myapp:myversion
```

1. Set to either the string **true** or to the Jaeger instance name.

When the sidecar is injected, the agent can then be accessed at its default location on **localhost**.

3.2.6.2. Manually injecting sidecars

The Red Hat OpenShift distributed tracing platform Operator can only automatically inject Jaeger Agent sidecars into Deployment workloads. For controller types other than **Deployments**, such as **StatefulSets`and `DaemonSets**, you can manually define the Jaeger agent sidecar in your specification.

The following snippet shows the manual definition you can include in your containers section for a Jaeger agent sidecar:

Sidecar definition example for a StatefulSet

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: example-statefulset
  namespace: example-ns
  labels:
    app: example-app
spec:

    spec:
      containers:
        - name: example-app
          image: acme/myapp:myversion
          ports:
            - containerPort: 8080
              protocol: TCP
        - name: jaeger-agent
          image: registry.redhat.io/distributed-tracing/jaeger-agent-rhel7:<version>
           # The agent version must match the Operator version
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 5775
              name: zk-compact-trft
              protocol: UDP
            - containerPort: 5778
              name: config-rest
              protocol: TCP
            - containerPort: 6831
              name: jg-compact-trft
              protocol: UDP
            - containerPort: 6832
              name: jg-binary-trft
              protocol: UDP
            - containerPort: 14271
              name: admin-http
              protocol: TCP
          args:
            - --reporter.grpc.host-port=dns:///jaeger-collector-headless.example-ns:14250
            - --reporter.type=grpc
```

The agent can then be accessed at its default location on localhost.

3.3. CONFIGURING AND DEPLOYING DISTRIBUTED TRACING DATA COLLECTION

The Red Hat OpenShift distributed tracing data collection Operator uses a custom resource definition (CRD) file that defines the architecture and configuration settings to be used when creating and deploying the Red Hat OpenShift distributed tracing data collection resources. You can either install the default configuration or modify the file to better suit your business requirements.

3.3.1. OpenTelemetry Collector configuration options

IMPORTANT

The Red Hat OpenShift distributed tracing data collection Operator 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 https://access.redhat.com/support/offerings/techpreview/.

The OpenTelemetry Collector consists of three components that access telemetry data:

- **Receivers** - A receiver, which can be push or pull based, is how data gets into the Collector. Generally, a receiver accepts data in a specified format, translates it into the internal format and passes it to processors and exporters defined in the applicable pipelines. By default, no receivers are configured. One or more receivers must be configured. Receivers may support one or more data sources.

- **Processors** - (Optional) Processors are run on data between being received and being exported. By default, no processors are enabled. Processors must be enabled for every data source. Not all processors support all data sources. Depending on the data source, it may be recommended that multiple processors be enabled. In addition, it is important to note that the order of processors matters.

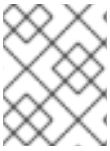
- **Exporters** - An exporter, which can be push or pull based, is how you send data to one or more backends/destinations. By default, no exporters are configured. One or more exporters must be configured. Exporters may support one or more data sources. Exporters may come with default settings, but many require configuration to specify at least the destination and security settings.

You can define multiple instances of components in a custom resource YAML file. Once configured, these components must be enabled through pipelines defined in the **spec.config.service** section of the YAML file. As a best practice you should only enable the components that you need.

sample OpenTelemetry collector custom resource file

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: cluster-collector
  namespace: tracing-system
spec:
  mode: deployment
  config: |
    receivers:
      otlp:
        protocols:
          grpc:
```

```
      http:
processors:
exporters:
  jaeger:
    endpoint: jaeger-production-collector-headless.tracing-system.svc:14250
    tls:
      ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: []
      exporters: [jaeger]
```



NOTE

If a component is configured, but not defined within the **service** section then it is not enabled.

Table 3.17. Parameters used by the Operator to define the OpenTelemetry Collector

Parameter	Description	Values	Default
`receivers:`	A receiver is how data gets into the Collector. By default, no receivers are configured. There must be at least one enabled receiver for a configuration to be considered valid. Receivers are enabled by being added to a pipeline.	**otlp**, **jaeger**	None
`receivers:` `  otlp:`	The **oltp** and **jaeger** receivers come with default settings, specifying the name of the receiver is enough to configure it.		
`processors:`	Processors run on data between being received and being exported. By default, no processors are enabled.		None

Parameter	Description	Values	Default
exporters:	An exporter sends data to one or more backends/destinations. By default, no exporters are configured. There must be at least one enabled exporter for a configuration to be considered valid. Exporters are enabled by being added to a pipeline. Exporters may come with default settings, but many require configuration to specify at least the destination and security settings.	**logging**, **jaeger**	None
exporters: jaeger: endpoint:	The **jaeger** exporter's endpoint must be of the form **<name>-collector-headless.<namespace>.svc**, with the name and namespace of the Jaeger deployment, for a secure connection to be established.		
exporters: jaeger: tls: ca_file:	Path to the CA certificate. For a client this verifies the server certificate. For a server this verifies client certificates. If empty uses system root CA.		
service: pipelines:	Components are enabled by adding them to a pipeline under **services.pipeline**.		
service: pipelines: traces: receivers:	You enable receivers for tracing by adding them under **service.pipelines.traces**.		None

Parameter	Description	Values	Default

`service:`
`  pipelines:`
`  traces:`
`  processors:`
	You enable processors for tracing by adding them under **service.pipelines.traces**.		None

`service:`
`  pipelines:`
`  traces:`
`  exporters:`
	You enable exporters for tracing by adding them under **service.pipelines.traces**.		None

3.3.2. Validating your deployment

3.3.3. Accessing the Jaeger console

To access the Jaeger console you must have either Red Hat OpenShift Service Mesh or Red Hat OpenShift distributed tracing installed, and Red Hat OpenShift distributed tracing platform installed, configured, and deployed.

The installation process creates a route to access the Jaeger console.

If you know the URL for the Jaeger console, you can access it directly. If you do not know the URL, use the following directions.

Procedure from OpenShift console

1. Log in to the OpenShift Container Platform web console as a user with cluster-admin rights. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.
2. Navigate to **Networking → Routes**.
3. On the **Routes** page, select the control plane project, for example **tracing-system**, from the **Namespace** menu.
   The **Location** column displays the linked address for each route.
4. If necessary, use the filter to find the **jaeger** route. Click the route **Location** to launch the console.
5. Click **Log In With OpenShift**

Procedure from the CLI

1. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.

```
$ oc login https://<HOSTNAME>:6443
```

2. To query for details of the route using the command line, enter the following command. In this example, **tracing-system** is the control plane namespace.

```
$ export JAEGER_URL=$(oc get route -n tracing-system jaeger -o jsonpath='{.spec.host}')
```

3. Launch a browser and navigate to **https://<JAEGER_URL>**, where **<JAEGER_URL>** is the route that you discovered in the previous step.

4. Log in using the same user name and password that you use to access the OpenShift Container Platform console.

5. If you have added services to the service mesh and have generated traces, you can use the filters and **Find Traces** button to search your trace data.
   If you are validating the console installation, there is no trace data to display.

3.4. UPGRADING DISTRIBUTED TRACING

Operator Lifecycle Manager (OLM) controls the installation, upgrade, and role-based access control (RBAC) of Operators in a cluster. The OLM runs by default in OpenShift Container Platform. OLM queries for available Operators as well as upgrades for installed Operators. For more information about how OpenShift Container Platform handles upgrades, see the Operator Lifecycle Manager documentation.

During an update, the Red Hat OpenShift distributed tracing Operators upgrade the managed distributed tracing instances to the version associated with the Operator. Whenever a new version of the Red Hat OpenShift distributed tracing platform Operator is installed, all the distributed tracing platform application instances managed by the Operator are upgraded to the Operator's version. For example, after upgrading the Operator from 1.10 installed to 1.11, the Operator scans for running distributed tracing platform instances and upgrades them to 1.11 as well.

For specific instructions on how to update the OpenShift Elasticsearch Operator, see Updating OpenShift Logging.

3.4.1. Changing the Operator channel for 2.0

Red Hat OpenShift distributed tracing 2.0.0 made the following changes:

- Renamed the Red Hat OpenShift Jaeger Operator to the Red Hat OpenShift distributed tracing platform Operator.
- Stopped support for individual release channels. Going forward, the Red Hat OpenShift distributed tracing platform Operator will only support the **stable** Operator channel. Maintenance channels, for example **1.24-stable**, will no longer be supported by future Operators.

As part of the update to version 2.0, you must update your OpenShift Elasticsearch and Red Hat OpenShift distributed tracing platform Operator subscriptions.

Prerequisites

- The OpenShift Container Platform version is 4.6 or later.
- You have updated the OpenShift Elasticsearch Operator.
- You have backed up the Jaeger custom resource file.

- An account with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.

IMPORTANT

If you have not already updated your OpenShift Elasticsearch Operator as described in Updating OpenShift Logging complete that update before updating your Red Hat OpenShift distributed tracing platform Operator.

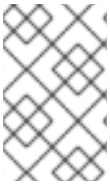
For instructions on how to update the Operator channel, see Upgrading installed Operators.

3.5. REMOVING DISTRIBUTED TRACING

The steps for removing Red Hat OpenShift distributed tracing from an OpenShift Container Platform cluster are as follows:

1. Shut down any Red Hat OpenShift distributed tracing pods.
2. Remove any Red Hat OpenShift distributed tracing instances.
3. Remove the Red Hat OpenShift distributed tracing platform Operator.
4. Remove the Red Hat OpenShift distributed tracing data collection Operator.

3.5.1. Removing a Red Hat OpenShift distributed tracing platform instance using the web console



NOTE

When deleting an instance that uses the in-memory storage, all data is permanently lost. Data stored in a persistent storage such as Elasticsearch is not be deleted when a Red Hat OpenShift distributed tracing platform instance is removed.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Operators → Installed Operators**.
3. Select the name of the project where the Operators are installed from the **Project** menu, for example, **openshift-operators**.
4. Click the Red Hat OpenShift distributed tracing platform Operator.
5. Click the **Jaeger** tab.
6. Click the Options menu  next to the instance you want to delete and select **Delete Jaeger**.
7. In the confirmation message, click **Delete**.

3.5.2. Removing a Red Hat OpenShift distributed tracing platform instance from the CLI

1. Log in to the OpenShift Container Platform CLI.

```
$ oc login
```

2. To display the distributed tracing platform instances run the command:

```
$ oc get deployments -n <jaeger-project>
```

For example,

```
$ oc get deployments -n openshift-operators
```

The names of Operators have the suffix **-operator**. The following example shows two Red Hat OpenShift distributed tracing platform Operators and four distributed tracing platform instances:

```
$ oc get deployments -n openshift-operators
```

You should see output similar to the following:

```
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
elasticsearch-operator   1/1     1            1           93m
jaeger-operator          1/1     1            1           49m
jaeger-test              1/1     1            1           7m23s
jaeger-test2             1/1     1            1           6m48s
tracing1                 1/1     1            1           7m8s
tracing2                 1/1     1            1           35m
```

3. To remove an instance of distributed tracing platform, run the following command:

```
$ oc delete jaeger <deployment-name> -n <jaeger-project>
```

For example:

```
$ oc delete jaeger tracing2 -n openshift-operators
```

4. To verify the deletion, run the **oc get deployments** command again:

```
$ oc get deployments -n <jaeger-project>
```

For example:

```
$ oc get deployments -n openshift-operators
```

You should see generated output that is similar to the following example:

```
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
elasticsearch-operator   1/1     1            1           94m
jaeger-operator          1/1     1            1           50m
```

```
jaeger-test          1/1   1         1         8m14s
jaeger-test2         1/1   1         1         7m39s
tracing1             1/1   1         1         7m59s
```

3.5.3. Removing the Red Hat OpenShift distributed tracing Operators

Procedure

1. Follow the instructions for Deleting Operators from a cluster .
    - Remove the Red Hat OpenShift distributed tracing platform Operator.
- After the Red Hat OpenShift distributed tracing platform Operator has been removed, if appropriate, remove the OpenShift Elasticsearch Operator.