OpenShift Container Platform 4.9

Distributed tracing

Jaeger installation, usage, and release notes

Last Updated: 2021-12-16

OpenShift Container Platform 4.9 Distributed tracing

Jaeger installation, usage, and release notes

Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

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. OPENSHIFT 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.

Using distributed tracing lets you perform the following functions:

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

Red Hat OpenShift distributed tracing consists of two 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 Red Hat 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, 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.

1.3.1. 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 effort includes the following:

- Updates distributed tracing 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 OpenTelemetry Operator based on OpenTelemetry 0.33. Note that this Operator is a Technology Preview.
- 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.

1.3.2. Component versions supported in Red Hat OpenShift distributed tracing version 2.0

Component	Version
Jaeger	1.28.0
OpenTelemetry	0.33.0

1.4. 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/.

1.4.1. Red Hat OpenShift distributed tracing 2.0.0 Technology Preview

This release includes the addition of the distributed tracing data collection, which you install using the OpenTelemetry Operator.

Red Hat OpenShift distributed tracing data collection is based on 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 the OpenShift distributed tracing platform. 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.

1.5. DISTRIBUTED TRACING KNOWN ISSUES

The following limitations exist in Red Hat OpenShift distributed tracing platform:

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

These are the known issues in Red Hat OpenShift distributed tracing platform:

- 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 does not upgrade, 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**.
- BZ-1918920 The Elasticsearch pods do not get restarted automatically after an update. As a workaround, restart the pods manually.
- TRACING-809 Jaeger Ingester is incompatible with Kafka 2.3. When there are two or more instances of the Jaeger Ingester and enough traffic, the Ingester generates continuous rebalancing messages in the logs. The Ingester generates these logs because of a regression in Kafka 2.3. This regression was fixed in Kafka 2.3.1. For more information, see Jaegertracing-1819.

1.6. DISTRIBUTED TRACING FIXED ISSUES

- 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, you get a 500 error because the oauth-proxy sidecar does not trust the custom CA bundle defined at installation time with the **additionalTrustBundle**.

This was due to the config map with the **additionalTrustBundle** not being created in the Jaeger namespace.

- 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**.

CHAPTER 2. DISTRIBUTED TRACING ARCHITECTURE

2.1. JAEGER 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. Jaeger 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's valuable for understanding serialization, parallelism, and sources of latency.

Jaeger 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 Jaeger 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. Jaeger overview

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

Using Jaeger lets you perform the following functions:

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

Jaeger is based on the vendor-neutral OpenTracing APIs and instrumentation.

2.1.2. Jaeger features

Jaeger tracing provides the following capabilities:

- Integration with Kiali – When properly configured, you can view Jaeger data from the Kiali console.
- High scalability – The Jaeger backend is designed to have no single points of failure and to scale with the business needs.
- Distributed Context Propagation – Lets you connect data from different components together to create a complete end-to-end trace.
- Backwards compatibility with Zipkin – Jaeger 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. Jaeger architecture

Jaeger is made up of several components that work together to collect, store, and display tracing data.

- **Jaeger Client** (Tracer, Reporter, instrumented application, client libraries)- Jaeger 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.

- **Jaeger Agent** (Server Queue, Processor Workers) - The Jaeger 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 like Kubernetes.

- **Jaeger Collector** (Queue, Workers) - Similar to the Agent, the Collector is able to receive spans and place them in an internal queue for processing. This allows the 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. Jaeger 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) - Jaeger can use Apache Kafka as a buffer between the collector and the actual backing storage (Elasticsearch). Ingester is a service that reads data from Kafka and writes to another storage backend (Elasticsearch).

- **Jaeger Console** – Jaeger provides a user interface that lets you 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.

CHAPTER 3. DISTRIBUTED TRACING INSTALLATION

3.1. INSTALLING JAEGER

You can install Jaeger on OpenShift Container Platform in either of two ways:

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

3.1.1. Prerequisites

Before you can install OpenShift Jaeger, 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 4.9 overview .
- Install OpenShift Container Platform 4.9.
  - Install OpenShift Container Platform 4.9 on AWS
  - Install OpenShift Container Platform 4.9 on user-provisioned AWS
  - Install OpenShift Container Platform 4.9 on bare metal
  - Install OpenShift Container Platform 4.9 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. Jaeger installation overview

The steps for installing OpenShift Jaeger 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 Jaeger Operator via the OperatorHub.
- Modify the Jaeger YAML file to support your deployment strategy.
- Deploy one or more instances of Jaeger to your OpenShift Container Platform environment.

3.1.3. Installing the OpenShift Elasticsearch Operator

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

Prerequisites

- Access to the OpenShift Container Platform web console.
- An account with the **cluster-admin** role. If you use {product-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 Jaeger Operator will create 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 {product-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, under **Installation Mode** select **All namespaces on the cluster (default)**. This makes the Operator available to all projects in the cluster.
7. Under **Installed Namespaces** select **openshift-operators-redhat** from the menu.

   NOTE

   The Elasticsearch installation requires the **openshift-operators-redhat** namespace for the OpenShift Elasticsearch Operator. The other OpenShift Jaeger operators are installed in the **openshift-operators** namespace.

8. Select **stable-5.x** as the **Update Channel**.
9. Select the **Automatic** Approval Strategy.

   NOTE

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

10. Click **Install**.
11. 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 Jaeger Operator

To install Jaeger you use the OperatorHub to install the Jaeger Operator.

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

Prerequisites

- Access to the OpenShift Container Platform web console.
- An account with the **cluster-admin** role. If you use {product-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 Jaeger 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 {product-dedicated}, you must have an account with the **dedicated-admin** role.
2. Navigate to **Operators** → **OperatorHub**.
3. Type **Jaeger** into the filter to locate the Jaeger Operator.
4. Click the **Jaeger 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 will automatically update

Jaeger as new versions are released. If you select a maintenance channel, for example, **1.17-stable**, you will receive bug fixes and security patches for the length of the support cycle for that version.

7. Select **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.

   - Select an Approval Strategy. You can select **Automatic** or **Manual** updates. If you choose Automatic updates for an installed Operator, when a new version of that Operator is available, the 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, the 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 **Subscription Overview** page, select the **openshift-operators** project. Wait until you see that the Jaeger Operator shows a status of "InstallSucceeded" before continuing.

3.2. CONFIGURING AND DEPLOYING JAEGER

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

Jaeger has predefined deployment strategies. You specify a deployment strategy in the custom resource file. When you create a Jaeger 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: simple-prod
spec:
  strategy: production (1)
```

(1) The Jaeger 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 Jaeger 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 backend storage (Elasticsearch). 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**
>
> There are two ways to install and use Jaeger, as part of a service mesh or as a stand alone component. If you have installed Jaeger as part of Red Hat OpenShift Service Mesh, you can configure and deploy Jaeger as part of the ServiceMeshControlPlane or configure Jaeger and then reference your Jaeger configuration in the ServiceMeshControlPlane .

3.2.1. Deploying the default Jaeger strategy from the web console

The custom resource definition (CRD) defines the configuration used when you deploy an instance of Jaeger. The default CR for Jaeger 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 Jaeger instance that uses the **AllInOne** deployment strategy, or you can define your own custom resource file.

> **NOTE**
>
> In-memory storage is not persistent, which means that if the Jaeger pod shuts down, restarts, or is replaced, that 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 Jaeger Operator must be installed.

- Review the instructions for how to customize the Jaeger installation.
- An account with the **cluster-admin** role.

NOTE

Jaeger streaming is currently unsupported on IBM Z.

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 **jaeger-system**.
   a. Navigate to **Home** → **Projects**.
   b. Click **Create Project**.
   c. Enter **jaeger-system** in the **Name** field.
   d. Click **Create**.
3. Navigate to **Operators** → **Installed Operators**.
4. If necessary, select **jaeger-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 OpenShift Jaeger 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, to install using the defaults, click **Create** to create the Jaeger instance.
8. On the **Jaegers** page, click the name of the Jaeger 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 default Jaeger from the CLI

Follow this procedure to create an instance of Jaeger from the command line.

Prerequisites

- An installed, verified OpenShift Jaeger Operator.
- Access to the OpenShift CLI (**oc**) that matches your OpenShift Container Platform version.
- An account 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 **jaeger-system**.

```
$ oc new-project jaeger-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 Jaeger:

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

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

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

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

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

3.2.2. Deploying the Jaeger production strategy from the web console

The **production** deployment 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.

Prerequisites

- The OpenShift Elasticsearch Operator must be installed.
- The Jaeger Operator must be installed.
- Review the instructions for how to customize the Jaeger installation.
- An account 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 **jaeger-system**.
   a. Navigate to **Home → Projects**.
   b. Click **Create Project**.

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

   d. Click **Create**.

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

4. If necessary, select **jaeger-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 Jaeger 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 Jaeger instance.

9. On the **Jaegers** page, click the name of the Jaeger 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 Jaeger production from the CLI

Follow this procedure to create an instance of Jaeger from the command line.

Prerequisites

- An installed, verified OpenShift Jaeger Operator.

- Access to the OpenShift CLI (**oc**).
- An account 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 **jaeger-system**.

   ```
   $ oc new-project jaeger-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 Jaeger:

   ```
   $ oc create -n jaeger-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 jaeger-system -w
   ```

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

   ```
   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 Jaeger streaming strategy from the web console

The **streaming** deployment 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.

The **streaming** strategy provides a streaming capability that sits between the collector and the storage (Elasticsearch). 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 streaming platform (Kafka).

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.

Prerequisites

- The AMQ Streams Operator must be installed. If using version 1.4.0 or higher you can use self-provisioning. If otherwise, you need to create the Kafka instance.
- The Jaeger Operator must be installed.
- Review the instructions for how to customize the Jaeger installation.
- An account with the **cluster-admin** role.

> **NOTE**
>
> Jaeger streaming is currently unsupported on IBM Z.

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 **jaeger-system**.
   a. Navigate to **Home → Projects**.
   b. Click **Create Project**.
   c. Enter **jaeger-system** in the **Name** field.
   d. Click **Create**.
3. Navigate to **Operators → Installed Operators**.
4. If necessary, select **jaeger-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 Jaeger 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 Jaeger instance.
2. On the **Jaegers** page, click the name of the Jaeger 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 Jaeger streaming from the CLI

Follow this procedure to create an instance of Jaeger from the command line.

Prerequisites

- An installed, verified OpenShift Jaeger Operator.
- Access to the OpenShift CLI (**oc**).
- An account 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 **jaeger-system**.

   ```
   $ oc new-project jaeger-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 jaeger-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 jaeger-system -w
   ```

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

   ```
   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. Customizing Jaeger deployment

3.2.4.1. Deployment best practices

- Jaeger instance names must be unique. If you want to have multiple Jaeger instances and are using sidecar injected Jaeger agents, then the Jaeger instances should have unique names, and the injection annotation should explicitly specify the Jaeger instance name the tracing data should be reported to.
- If you have a multitenant implementation and tenants are separated by namespaces, deploy a Jaeger instance to each tenant namespace.
  - Jaeger agent as a daemonset is not supported for multitenant installations or OpenShift Dedicated. Jaeger agent as a sidecar is the only supported configuration for these use cases.
- If you are installing Jaeger as part of Red Hat OpenShift Service Mesh, Jaeger resources must be installed in the same namespace as the **ServiceMeshControlPlane** resource.

3.2.4.2. Jaeger default configuration options

The Jaeger custom resource (CR) defines the architecture and settings to be used when creating the Jaeger resources. You can modify these parameters to customize your Jaeger 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:	Version of the Application Program Interface 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**.		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 Jaeger instance.	**jaeger-all-in-one-inmemory**
spec:	Specification for the object to be created.	Contains all of the configuration parameters for your Jaeger 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

Parameter	Description	Values	Default value
strategy:	Jaeger deployment strategy	**allInOne**, **production**, or **streaming**	**allInOne**
allInOne:	Because the allInOne image deploys the agent, collector, query, ingester, Jaeger UI in a single pod, configuration for this deployment should nest component configuration under the allInOne parameter.		
agent:	Configuration options that define the Jaeger agent.		
collector:	Configuration options that define the Jaeger Collector.		
sampling:	Configuration options that define the sampling strategies for tracing.		
storage:	Configuration options that define the storage. All storage related options should 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 Jaeger instance using the default settings.

Example minimum required jaeger-all-in-one.yaml

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
```

```
metadata:
  name: jaeger-all-in-one-inmemory
```

3.2.4.3. 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 a persistent storage (Elasticsearch) 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. Jaeger 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

Parameter	Description	Values

`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, Jaeger will self-provision Kafka.	

`log-level:`	Logging level for the collector.	**trace**, **debug**, **info**, **warning**, **error**, **fatal**, **panic**
`maxReplicas:`	Specifies the maximum number of replicas to create when autoscaling the Collector.	Integer, for example, **100**
`num-workers:`	The number of workers pulling from the queue.	Integer, for example, **50**
`queue-size:`	The size of the Collector queue.	Integer, for example, **2000**
`replicas:`	Specifies the number of Collector replicas to create.	Integer, for example, **5**

3.2.4.3.1. Configuring the Collector for autoscaling

NOTE

Autoscaling is only supported for Jaeger 1.20 or later.

You can configure the Collector to autoscale; the Collector will scale up or down based on the CPU and/or memory consumption. Configuring the Collector to autoscale can help you ensure your Jaeger environment scales up during times of increased load, and scales down when less resources are needed, saving on costs. You configure autoscaling by setting the **autoscale** parameter to **true** and specifying a value for **.spec.collector.maxReplicas** along with a reasonable value for the resources that you expect the Collector's pod to consume. If you do not set a value for **.spec.collector.maxReplicas** the Operator will set it to **100**.

By default, when there is no value provided for **.spec.collector.replicas**, the Jaeger Operator creates a horizontal pod autoscaler (HPA) configuration for the Collector. For more information about HPA, refer to the Kubernetes documentation.

The following is an example autoscaling configuration, setting the Collector's limits as well as the maximum number of replicas:

Collector autoscaling example

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  collector:
    maxReplicas: 5
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
```

3.2.4.4. Jaeger sampling configuration options

The 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 Istio 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 Jaeger tracer.

When a service receives a request that contains no trace context, the Jaeger tracer will start a new trace, assign it a random trace ID, and make a sampling decision based on the currently installed sampling strategy. The sampling decision is propagated to all subsequent requests in the trace, so that other services are not making the sampling decision again.

Jaeger 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, with sampling.param=0.1 approximately 1 in 10 traces will be sampled.
- **Rate Limiting** - The sampler uses a leaky bucket rate limiter to ensure that traces are sampled with a certain constant rate. For example, when sampling.param=2.0 it will sample 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 probability 0.001 (0.1%) for all services.

Parameter	Description	Values	Default value

`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, Jaeger uses the following settings.

default sampling

```
spec:
  sampling:
    options:
```

```
default_strategy:
  type: probabilistic
  param: 1
```

3.2.4.5. Jaeger 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 Operator to define Jaeger storage

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 Jaeger supports Elasticsearch for persistent storage.	**memory**

`storage:`
`  secretname:`
	Name of the secret, for example **jaeger-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**

Parameter	Description	Values	Default value

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 * * *"

3.2.4.5.1. Auto-provisioning an Elasticsearch instance

When the **storage:type** is set to **elasticsearch** but there is no value set for **spec:storage:options:es:server-urls**, the Jaeger 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.

Restrictions

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

NOTE

If you already have installed Elasticsearch as part of OpenShift Logging, the Jaeger 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 Jaeger 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: 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

Parameter	Description	Values	Default value

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*	

elasticsearch: redundancyPolicy:	Data replication policy defines how Elasticsearch shards are replicated across data nodes in the cluster. If not specified, the Jaeger 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).	
	*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, Jaeger uses **emptyDir**. The OpenShift Elasticsearch Operator provisions **PersistentVolumeClaim** and **PersistentVolume** which are not removed with Jaeger instance. You can mount the same volumes if you create a Jaeger instance with the same name and namespace.

3.2.4.5.2. Connecting to an existing Elasticsearch instance

You can use an existing Elasticsearch cluster for storage with Jaeger, that is, an instance that was not auto-provisioned by the Jaeger Operator. You do this by specifying the URL of the existing cluster as the **spec:storage:options:es:server-urls** value in your configuration.

Restrictions

- You cannot share or reuse a OpenShift Jaeger logging Elasticsearch instance with Jaeger. The Elasticsearch cluster is meant to be dedicated for a single Jaeger 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
`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 Jaeger indices. For example, setting this to "production" creates indices named "production-jaeger-*".		

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
`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 server(s).		Will use the system truststore by default.
es: tls: cert:	Path to a TLS Certificate file, used to identify this process to the remote server(s).		
es: tls: enabled:	Enable transport layer security (TLS) when talking to the remote server(s). Disabled by default.	**true**/ **false**	**false**
es: tls: key:	Path to a TLS Private Key file, used to identify this process to the remote server(s).		
es: tls: server-name:	Override the expected TLS server name in the certificate of the remote server(s).		
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

Parameter	Description	Values	Default value

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**
es-archive: index-prefix:	Optional prefix for Jaeger indices. For example, setting this to "production" creates indices named "production-jaeger-*".		
es-archive: max-doc-count:	The maximum document count to return from an Elasticsearch query. This will also apply to aggregations.		0

Parameter	Description	Values	Default value
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**.		
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**

Parameter	Description	Values	Default value
`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 server(s).		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 server(s).		
`es-archive: tls: enabled:`	Enable transport layer security (TLS) when talking to the remote server(s). Disabled by default.	**true**/ **false**	**false**
`es-archive: tls: key:`	Path to a TLS Private Key file, used to identify this process to the remote server(s).		
`es-archive: tls: server-name:`	Override the expected TLS server name in the certificate of the remote server(s).		
`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**.		

Parameter	Description	Values	Default value
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: jaeger-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: jaeger-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 jaeger-secret --from-literal=ES_PASSWORD=changeme --from-literal=ES_USERNAME=elastic
4. Volume mounts and volumes which are mounted into all storage components.

3.2.4.6. Jaeger 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 Operator to define Jaeger Query

Parameter	Description	Values	Default value

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

Table 3.15. Jaeger 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: **trace**, **debug**, **info**, **warning**, **error**, **fatal**, **panic**.	

Parameter	Description	Values	Default value

`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
```

3.2.4.7. Jaeger Ingester configuration options

Ingester is a service that reads from a Kafka topic and writes to another storage backend (Elasticsearch). 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 should wait for a message before terminating. The deadlock interval is disabled by default (set to 0), to avoid the Ingester being terminated when no messages arrive while the system is being initialized.	Minutes and seconds, for example, **1m0s**. Default value is **0**.

Parameter	Description	Values

```
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**.

```
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**.

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

`log-level:`	Logging level for the Ingester.	Possible values: **trace**, **debug**, **info**, **warning**, **error**, **fatal**, **panic**.
`maxReplicas:`	Specifies the maximum number of replicas to create when autoscaling the Ingester.	Integer, for example, **100**.

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
```

3.2.4.7.1. Configuring Ingester for autoscaling

NOTE

Autoscaling is only supported for Jaeger 1.20 or later.

You can configure the Ingester to autoscale; the Ingester will scale up or down based on the CPU and/or memory consumption. Configuring the Ingester to autoscale can help you ensure your Jaeger environment scales up during times of increased load, and scales down when less resources are needed, saving on costs. You configure autoscaling by setting the **autoscale** parameter to **true** and specifying a value for **.spec.ingester.maxReplicas** along with a reasonable value for the resources that you expect the Ingester's pod to consume. If you do not set a value for **.spec.ingester.maxReplicas** the Operator will set it to **100**.

By default, when there is no value provided for **.spec.ingester.replicas**, the Jaeger Operator creates a horizontal pod autoscaler (HPA) configuration for the Ingester. For more information about HPA, refer to the Kubernetes documentation.

The following is an example autoscaling configuration, setting the Ingester's limits as well as the maximum number of replicas:

Ingester autoscaling example

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-streaming
spec:
  strategy: streaming
  ingester:
    maxReplicas: 8
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
```

3.2.5. Injecting sidecars

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

3.2.5.1. Automatically injecting sidecars

To enable this feature, you add the annotation **sidecar.jaegertracing.io/inject** set to either the string

true or the Jaeger instance name as returned by **oc get jaegers**. When you specify **true**, there should be only a single Jaeger instance for the same namespace as the deployment, otherwise, the Operator cannot determine which Jaeger instance to use. A specific Jaeger 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 Jaeger Agent pointing to the single Jaeger instance available in the same namespace:

sample automatic sidecar injection

```
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
```

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

3.2.5.2. Manually injecting sidecars

For controller types other than **Deployments** (for example, **StatefulSets**, **DaemonSets**, etc.), 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:

example sidecar definition 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 Jaeger Agent can then be accessed at its default location on localhost.

3.3. UPGRADING JAEGER

The 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. The OLM queries for available Operators as well as upgrades for installed Operators. For more information about how OpenShift Container Platform handled upgrades, refer to the Operator Lifecycle Manager documentation.

The update approach used by the Jaeger Operator upgrades the managed Jaeger instances to the version associated with the Operator. Whenever a new version of the Jaeger Operator is installed, all the Jaeger application instances managed by the Operator will be upgraded to the Operator's version. For example, if version 1.10 is installed (both Operator and backend components) and the Operator is upgraded to version 1.11, then as soon as the Operator upgrade has completed, the Operator will scan for running Jaeger instances and upgrade them to 1.11 as well.

For specific instructions for how to update the OpenShift Elasticsearch Operator, refer to Updating OpenShift Logging.

3.4. REMOVING JAEGER

The steps for removing Jaeger from an OpenShift Container Platform cluster are as follows:

1. Shut down any Jaeger pods.

2. Remove any Jaeger instances.

3. Remove the Jaeger Operator.

3.4.1. Removing a Jaeger instance using the web console

> **NOTE**
>
> When deleting an instance that uses the in-memory storage, all data will be permanently lost. Data stored in a persistent storage (such as Elasticsearch) will not be deleted when a Jaeger 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, **jaeger-system**.
4. Click the Jaeger 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**.

3.4.2. Removing a Jaeger instance from the CLI

1. Log in to the OpenShift Container Platform CLI.

```
$ oc login
```

2. To display the Jaeger instances run the command:

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

The names of operators have the suffix **-operator**. The following example shows two Jaeger Operators and four Jaeger instances:

```
$ oc get deployments -n jaeger-system
```

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 Jaeger, run the command:

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

For example,

```
$ oc delete jaeger tracing2 -n jaeger-system
```

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

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

For example,

```
$ oc get deployments -n jaeger-system
```

Should generate output similar to the following:

```
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
```

3.4.3. Removing the Jaeger Operator

Procedure

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