OpenShift Container Platform 4.4

Jaeger

Jaeger installation, usage, and release notes
OpenShift Container Platform 4.4 Jaeger

Jaeger installation, usage, and release notes
Abstract

This document provides information on how to use Jaeger in OpenShift Container Platform
Table of Contents

CHAPTER 1. JAEGER RELEASE NOTES ............................................. 3
  1.1. JAEGER OVERVIEW ....................................................... 3
  1.2. GETTING SUPPORT ...................................................... 3
  1.3. NEW FEATURES OPENShift JAEGER 1.17.1 ......................... 3
  1.4. JAEGER KNOWN ISSUES .............................................. 3

CHAPTER 2. JAEGER ARCHITECTURE ........................................... 5
  2.1. JAEGER ARCHITECTURE .............................................. 5
      2.1.1. Jaeger overview ............................................. 5
      2.1.2. Jaeger features ............................................ 5
      2.1.3. Jaeger architecture ........................................ 6

CHAPTER 3. JAEGER INSTALLATION ........................................... 7
  3.1. INSTALLING JAEGER .................................................. 7
      3.1.1. Jaeger installation overview .............................. 7
      3.1.2. Installing the Elasticsearch Operator .................. 7
      3.1.3. Installing the Jaeger Operator ......................... 8
  3.2. DEPLOYING JAEGER .................................................... 9
      3.2.1. Deploying the default Jaeger strategy from the web console ........................................ 10
      3.2.1.1. Deploying default Jaeger from the CLI ................ 11

CHAPTER 4. DEPLOYING THE JAEGER PRODUCTION STRATEGY FROM THE WEB CONSOLE ................................. 13
  4.1. DEPLOYING JAEGER PRODUCTION FROM THE CLI ................. 14

CHAPTER 5. DEPLOYING THE JAEGER STREAMING STRATEGY FROM THE WEB CONSOLE ................................. 16
  5.1. DEPLOYING JAEGER STREAMING FROM THE CLI .................. 17
  5.2. CUSTOMIZING JAEGER DEPLOYMENT .............................. 18
      5.2.1. Jaeger default configuration options .................. 18
      5.2.2. Jaeger Collector configuration options ................ 21
      5.2.3. Jaeger sampling configuration options ................ 21
      5.2.4. Jaeger storage configuration options .................. 23
      5.2.5. Jaeger Query configuration options .................... 26
      5.2.6. Jaeger Ingester configuration options ................. 27
  5.3. INJECTING SIDECARs ................................................. 29

CHAPTER 6. AUTOMATICALLY INJECTING SIDECARs ............................. 30

CHAPTER 7. MANUALLY INJECTING SIDECARs ................................. 31
  7.1. UPGRAADING JAEGER ............................................... 31
  7.2. REMOVING JAEGER ................................................... 32
      7.2.1. Removing a Jaeger instance using the web console .... 32
      7.2.2. Removing a Jaeger instance from the CLI .............. 32
      7.2.3. Removing the Jaeger Operator ............................ 33
CHAPTER 1. JAEGGER RELEASE NOTES

1.1. JAEGGER 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.

1.2. GETTING SUPPORT

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal. Through the Customer Portal, you can:

- Search or browse through the Red Hat Knowledgebase of technical support articles about Red Hat products.
- Submit a support case to Red Hat Support.

**NOTE**

When submitting a support case, it is recommended to provide the following information about your cluster to Red Hat Support to aid in troubleshooting:

- Data gathered using the `oc adm must-gather` command
- The unique cluster ID. Navigate to (?) Help → Open Support Case to have the cluster ID autofilled when you submit the case.

- Access other product documentation.

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.

1.3. NEW FEATURES OPENSHIFT JAEGGER 1.17.1

This release of OpenShift Jaeger adds support for installing Jaeger as a standalone solution, rather than as a component of Red Hat OpenShift Service Mesh.

1.4. JAEGGER KNOWN ISSUES

These limitations exist in Jaeger:
• While Kafka publisher is included as part of Jaeger, it is not supported.

• Apache Spark is not supported.

• Only self-provisioned Elasticsearch instances are supported. External Elasticsearch instances are not supported in this release.

These are the known issues in Jaeger:

• **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 an 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.
CHAPTER 2. JAEGER 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 in order 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.
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.

- If you do not want to install a service mesh, you can use the Jaeger Operator to install the Red Hat build of Jaeger by itself. To install Jaeger without a service mesh, use the following instructions.

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 4.4 overview.

- Install OpenShift Container Platform 4.4.
  - Install OpenShift Container Platform 4.4 on AWS
  - Install OpenShift Container Platform 4.4 on user-provisioned AWS
  - Install OpenShift Container Platform 4.4 on bare metal
  - Install OpenShift Container Platform 4.4 on vSphere

- Install the version of the OpenShift Container Platform command line utility (the oc client tool) that matches your OpenShift Container Platform version and add it to your path.

- An account with the cluster-admin role.

3.1.1. 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 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.

3.1.2. Installing the 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 a persistent storage option, in this case, Elasticsearch.

**Prerequisites**

- Access to the OpenShift Container Platform web console.
- An account with the `cluster-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.


3. Type Elasticsearch into the filter box to locate the Elasticsearch Operator.

4. Click the Elasticsearch Operator provided by Red Hat to display information about the Operator.

5. Click Install.


7. Select the Update Channel that matches your OpenShift Container Platform installation. For example, if you are installing on OpenShift Container Platform version 4.4, select the 4.4 update channel.

8. Select the Automatic Approval Strategy.

   ![NOTE]

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

9. Click Subscribe.

10. On the Installed Operators page, select the openshift-operators-redhat project. Wait until you see that the Elasticsearch Operator shows a status of “InstallSucceeded” before continuing.

### 3.1.3. 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 require persistent storage, you must also install the 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.
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 Create Operator Subscription page, 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.
7. 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.
   - 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.
8. Click Subscribe.
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. DEPLOYING JAEGER

The Jaeger Operator includes a custom resource definition (CRD) file that defines the architecture and configuration settings for 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

```yaml
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
```

### The Jaeger Operator currently supports the following deployment strategies:

- **allInOne (Default)** - This strategy is intended for development, testing, and demo purposes. 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 or as a daemonset. 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**

  There are two ways to install 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 must configure and deploy Jaeger as part of 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.

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.
   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 Container Platform Command-line Interface (CLI) commonly known as `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**

   ```yaml
   apiVersion: jaegertracing.io/v1
   kind: Jaeger
   metadata:
     name: jaeger-all-in-one-inmemory
   
   $ oc create -n jaeger-system -f jaeger.yaml

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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>jaeger-all-in-one-inmemory-cdf7897b-qhfdx</td>
<td>2/2</td>
<td>Running 0</td>
<td>0</td>
<td>24s</td>
</tr>
</tbody>
</table>
CHAPTER 4. 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 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.
   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

```yaml
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-production
namespace:
spec:
  strategy: production
ingress:
  security: oauth-proxy
storage:
  type: elasticsearch
```
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.

### 4.1. 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 Container Platform Command-line Interface (CLI) commonly known as **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.

   ```bash
   $ oc login https://{HOSTNAME}:8443
   ```

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

   ```bash
   $ 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:

   ```bash
   $ 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:

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

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

```yaml
elasticsearch:
  nodeCount: 3
  redundancyPolicy: SingleRedundancy
esIndexCleaner:
  enabled: true
  numberOfDays: 7
  schedule: 55 23 * * *
esRollover:
  schedule: "*/30 * * * *"
```
<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>elasticsearch-cdm-jaegersystemjaegerproduction-1-6676cf568gwhlw</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>10m</td>
</tr>
<tr>
<td>elasticsearch-cdm-jaegersystemjaegerproduction-2-bcd4c8bf5l6g6w</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>10m</td>
</tr>
<tr>
<td>elasticsearch-cdm-jaegersystemjaegerproduction-3-844d6d9694hhst</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>10m</td>
</tr>
<tr>
<td>jaeger-production-collector-94cd847d-jwjlj</td>
<td>1/1</td>
<td>Running</td>
<td>3</td>
<td>8m32s</td>
</tr>
<tr>
<td>jaeger-production-query-5cbfbd499d-tv8zf</td>
<td>3/3</td>
<td>Running</td>
<td>3</td>
<td>8m32s</td>
</tr>
</tbody>
</table>
CHAPTER 5. 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).

**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.

**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**

```yaml
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-streaming
spec:
```

OpenShift Container Platform 4.4 Jaeger
strategy: streaming
collector:
  options:
  kafka:
    producer:
      topic: jaeger-spans
      # Note: If brokers are not defined, AMQ Streams 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.

---

### 5.1. 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 Container Platform Command-line Interface (CLI) commonly known as **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-1-697b66d6fcztcn 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
```

5.2. CUSTOMIZING JAEGER DEPLOYMENT

5.2.1. 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

```yaml
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: {}
```
Table 5.1. Jaeger parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Values</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiVersion</td>
<td>Version of the Application Program Interface to use when creating the object.</td>
<td>jaegertracing.io/v1</td>
<td>jaegertracing.io/v1</td>
</tr>
<tr>
<td>kind:</td>
<td>Defines the kind of Kubernetes object to create.</td>
<td>jaeger</td>
<td></td>
</tr>
<tr>
<td>metadata:</td>
<td>Data that helps uniquely identify the object, including a name string, UID, and optional namespace.</td>
<td></td>
<td>OpenShift automatically generates the UID and completes the namespace with the name of the project where the object is created.</td>
</tr>
<tr>
<td>name:</td>
<td>Name for the object.</td>
<td>The name of your Jaeger instance.</td>
<td>jaeger-all-in-one-inmemory</td>
</tr>
<tr>
<td>spec:</td>
<td>Specification for the object to be created.</td>
<td>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/&lt;component&gt; node.</td>
<td>N/A</td>
</tr>
<tr>
<td>strategy:</td>
<td>Jaeger deployment strategy</td>
<td>allInOne, production, or streaming</td>
<td>allInOne</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Values</td>
<td>Default value</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>allInOne:</td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>agent:</td>
<td>Configuration options that define the Jaeger agent.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>collector:</td>
<td>Configuration options that define the Jaeger Collector.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sampling:</td>
<td>Configuration options that define the sampling strategies for tracing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>storage:</td>
<td>Configuration options that define the storage. All storage related options should be placed under <code>storage</code>, rather than under the <code>allInOne</code> or other component options.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>query:</td>
<td>Configuration options that define the Query service.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ingester:</td>
<td>Configuration options that define the Ingester service.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

```yaml
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```
5.2.2. 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 5.2. Jaeger collector parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>spec:</td>
<td>Configuration options that define the Jaeger Collector.</td>
<td></td>
</tr>
<tr>
<td>collector:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>options:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kafka:</td>
<td>The topic parameter identifies the Kafka configuration used by the collector to produce the messages, and the ingester to consume the messages.</td>
<td>Label for the producer</td>
</tr>
<tr>
<td>producer:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>topic:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jaeger-spans</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kafka:</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>producer:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>brokers:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>my-cluster-kafka-brokers.kafka:9092</td>
<td></td>
<td></td>
</tr>
<tr>
<td>log-level:</td>
<td>Logging level for the collector.</td>
<td>Possible values: trace, debug, info, warning, error, fatal, panic.</td>
</tr>
</tbody>
</table>

5.2.3. 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:

- **Constant** - The sampler always makes the same decision for all traces. It either samples all traces (sampling.param=1) or none of them (sampling.param=0).

- **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.

- **Remote** - The sampler consults the Jaeger agent for the appropriate sampling strategy to use in the current service. This allows controlling the sampling strategies in the services from a central configuration in the Jaeger backend.

### Table 5.3. Jaeger sampling parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Values</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spec:sampling:options:[]</code></td>
<td>Configuration options that define the sampling strategies for tracing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sampling:type:</code></td>
<td>Sampling strategy to use. (See descriptions above.)</td>
<td>Valid values are <code>const</code>, <code>probabilistic</code>, <code>ratelimiting</code>, and <code>remote</code>.</td>
<td><em>remote</em></td>
</tr>
<tr>
<td><code>sampling:options:type:param:</code></td>
<td>Parameters for the selected sampling strategy (See examples above)</td>
<td>Decimal and integer values (0, .1, 1, 10)</td>
<td><em>N/A</em></td>
</tr>
</tbody>
</table>

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

**Probabilistic sampling example**

```yaml
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: with-sampling
spec:
  strategy: allInOne
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 50
```
5.2.4. Jaeger storage configuration options

The query and collector services are configured with a supported storage type; Elasticsearch. Multiple instances of each of these components can be provisioned as required for performance and resilience purposes.

When there is no `es.server-urls` option as part of a Jaeger production instance and Elasticsearch is set as the storage type, the Jaeger Operator creates an Elasticsearch cluster via the Elasticsearch Operator by creating a Custom Resource based on the configuration provided in storage section. The Elasticsearch cluster is meant to be dedicated for a single Jaeger instance.

Restrictions

- There can be only one Jaeger with self-provisioned Elasticsearch instance per namespace.
- There can be only one Elasticsearch per namespace.
- You cannot share or reuse a OpenShift Jaeger logging Elasticsearch instance with Jaeger.

Table 5.4. Jaeger storage parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Values</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>spec:</td>
<td>Configuration options that define the storage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>storage:</td>
<td>Type of storage to use for the deployment</td>
<td><code>memory</code> or <code>elasticsearch</code></td>
<td><code>memory</code></td>
</tr>
<tr>
<td>options:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>server-urls:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>elasticsearch:</td>
<td>Number of Elasticsearch nodes. For high availability use at least 3 nodes. Do not use 2 nodes as “split brain” problem can happen.</td>
<td>Integer value. For example, Proof of concept = 1, Minimum deployment =3</td>
<td>1</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Values</td>
<td>Default value</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
<td>---------------</td>
</tr>
<tr>
<td>storage: elasticsearch: redundancyPolicy:</td>
<td>Controls number of Elasticsearch replica shards.</td>
<td><strong>ZeroRedundancy</strong> (no replica shards), <strong>SingleRedundancy</strong> (one replica shard), <strong>MultipleRedundancy</strong> (each index is spread over half of the Data nodes), <strong>FullRedundancy</strong> (each index is fully replicated on every Data node in the cluster)</td>
<td></td>
</tr>
<tr>
<td>storage: elasticsearch: resources: requests: cpu:</td>
<td>Number of central processing units for requests, based on your environment’s configuration.</td>
<td>Specified in cores or millicores (for example, 200m, 0.5, 1). For example, Proof of concept = 500m, Minimum deployment = 1</td>
<td>1Gi</td>
</tr>
<tr>
<td>storage: elasticsearch: resources: requests: memory:</td>
<td>Available memory for requests, based on your environment’s configuration.</td>
<td>Specified in bytes (for example, 200Ki, 50Mi, 5Gi). For example, Proof of concept = 1Gi, Minimum deployment = 16Gi*</td>
<td>500m</td>
</tr>
<tr>
<td>storage: elasticsearch: resources: limits: cpu:</td>
<td>Limit on number of central processing units, based on your environment’s configuration.</td>
<td>Specified in cores or millicores (for example, 200m, 0.5, 1). For example, Proof of concept = 500m, Minimum deployment = 1</td>
<td></td>
</tr>
<tr>
<td>storage: elasticsearch: resources: limits: memory:</td>
<td>Available memory limit based on your environment’s configuration.</td>
<td>Specified in bytes (for example, 200Ki, 50Mi, 5Gi). For example, Proof of concept = 1Gi, Minimum deployment = 16Gi*</td>
<td></td>
</tr>
<tr>
<td>storage: elasticsearch: secretname:</td>
<td>Name of the secret, for example jaeger-secret.</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Values</td>
<td>Default value</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>storage: esIndexCleaner: enabled</td>
<td>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.</td>
<td>true/ false</td>
<td>true</td>
</tr>
<tr>
<td>storage: esIndexCleaner: numberOfDays</td>
<td>Number of days to wait before deleting an index.</td>
<td>Integer value</td>
<td>7</td>
</tr>
<tr>
<td>storage: esIndexCleaner: schedule</td>
<td>Defines the schedule for how often to clean the Elasticsearch index.</td>
<td>Cron expression</td>
<td>&quot;55 23 * * *&quot;</td>
</tr>
<tr>
<td>storage: esRollover: schedule</td>
<td>Defines the schedule for how often to rollover to a new Elasticsearch index.</td>
<td>Cron expression</td>
<td>'*/30 * * *'</td>
</tr>
</tbody>
</table>

* 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

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

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
storage:
  type: elasticsearch
  options:
    es:
      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
```

5.2.5. Jaeger Query configuration options

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

Table 5.5. Jaeger Query parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Values</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>spec:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>query:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>options:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resources:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Configuration options that define the Query service.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| query: | additional-headers: |        | Format: "Key: Value" |
|        |                     |        |               |
|        | Additional HTTP response headers. Can be specified multiple times. |        |               |
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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Values</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>query: base-path:</td>
<td>The base path for all jaeger-query HTTP routes can be set to a non-root value, for example, <code>/jaeger</code> would cause all UI URLs to start with <code>/jaeger</code>. This can be useful when running jaeger-query behind a reverse proxy.</td>
<td>/{path}</td>
<td></td>
</tr>
<tr>
<td>query: port:</td>
<td>The port for the query service.</td>
<td></td>
<td>16686</td>
</tr>
<tr>
<td>options: log-level:</td>
<td>Logging level for Query. Possible values: <code>trace</code>, <code>debug</code>, <code>info</code>, <code>warning</code>, <code>error</code>, <code>fatal</code>, <code>panic</code>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sample Query configuration

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

5.2.6. Jaeger Ingester configuration options

Ingester is a service that reads from 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 5.6. Jaeger default parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Values</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>spec:</td>
<td>Configuration options that define the Ingester service.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>strategy:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>streaming</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ingester:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>options:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kafka:</td>
<td>The <strong>topic</strong> parameter identifies the Kafka configuration used by the collector to produce the messages, and the ingester to consume the messages.</td>
<td></td>
<td>Label for the consumer</td>
</tr>
<tr>
<td>consumer:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>topic:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jaeger-spans</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ingester:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>options:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kafka:</td>
<td>Identifies the Kafka configuration used by the Ingester to consume the messages.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>consumer:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>brokers:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>my-cluster-kafka-brokers.kafka:9092</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ingester:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>options:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ingester:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deadlockInterval: 5</td>
<td>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.</td>
<td>Minutes and seconds, for example, 1m0s.</td>
<td>0</td>
</tr>
<tr>
<td>options:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Streaming Collector and Ingester example**

```yaml
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-streaming
```
5.3. 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.
CHAPTER 6. 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:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
annotations:
  "sidecar.jaegertracing.io/inject": "true"
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`. 

OpenShift Container Platform 4.4 Jaeger
CHAPTER 7. 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

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: example-statefulset
  namespace: example-ns
labels:
  app: example-app
spec:
  containers:
    - name: example-app
      image: acme/myapp:myversion
    - 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: 8080
          protocol: TCP
        - 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.

7.1. 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.

7.2. REMOVING JAEGER

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

1. Shut down any Jaeger pods.
2. Remove any Jaeger instances.
3. Remove the Jaeger Operator.

7.2.1. 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.

7.2.2. 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:

```bash
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:

```
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:

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

For example,

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

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

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

For 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
```

### 7.2.3. 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 Elasticsearch Operator.