How to deploy Jaeger on Kubernetes

A beginner's guide to Jaeger (5 Part Series)

Magsther · Follow
Published in FAUN—Developer Community 🐾
7 min read · Aug 30, 2022

In this **5 part series of Jaeger**, we will cover the following topics:

1. A beginner's guide to Jaeger
2. How to deploy Jaeger (and Hot R.O.D) with Docker Compose
3. How to deploy Jaeger and Elasticsearch
4. **How to deploy Jaeger on Kubernetes (this one)**
5. How to deploy Jaeger on Kubernetes with Terraform

Getting Started

Just a recap of what we have been doing so far in this series.

In part 1, we presented you the concepts of Jaeger and Hot ROD.

In part 2, we deployed the **Jaeger All-in-One** image and the **Hot ROD** image using **Docker Compose**.

In part 3, we added **Elasticsearch** to the mix. Elasticsearch is one of the recommend storage backends for tracing.

In this part (4), we will talk about:

- Jaeger Operators
- Installing the Jaeger Operator
- Installing Cert-Manager
- Deploying the Jaeger All-One image
- NodeJS demo application

Jaeger Operators

To deploy Jaeger on Kubernetes cluster, we can make use of the Jaeger operator.

> *Operators are pieces of software that ease the operational complexity of running another piece of software.*

What you do is that you first install the **Jaeger Operator** on Kubernetes.

This operator will then **watch** for new **Jaeger** custom resources (CR) in specific namespaces, or across the entire cluster.

Installing the Jaeger Operator

There are different ways of installing the Jaeger Operator on Kubernetes:

- using **Helm**
- using **Deployment** files

Before you start, pay attention to the Prerequisite section.

> *Since version 1.31 the Jaeger Operator uses webhooks to validate Jaeger custom resources (CRs). This requires an installed version of the cert-manager.*

Installing Cert-Manager

The installation of cert-manager is very simple as you can see from the **Getting Started** page.

To install all `cert-manager` components, just run:

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.9.0/cert-manager.yaml
```

By default, **cert-manager** will be installed into the **cert-manager namespace**.

You can verify the installation with:

```
kubectl get pods -n cert-manager
```

You should see the `cert-manager`, `cert-manager-cainjector`, and `cert-manager-webhook` pods in a `Running` state. The webhook might take a little longer to successfully provision than the others.

Installing Jaeger Operator using Helm

Jump over to Artifact Hub and search for **jaeger-operator.** Add the chart to the Helm repository.

```
helm repo add jaegertracing https://jaegertracing.github.io/helm-charts
```

To **install** the chart with the release name `my-release` in **observability** namespace

```
kubectl create ns observability

helm install my-release jaegertracing/jaeger-operator -n
observability
```

You can also install a **specific version** of the helm chart:

```
helm install my-jaeger-operator jaegertracing/jaeger-operator --
version 2.25.0 -n observability
```

Verify that it's installed on Kubernetes:

```
helm list -A
```

Installing Jaeger Operator using deployment files

The Jaeger Tracing provides instructions on how to install the operator on Kubernetes using deployment files. Again, create a new namespace:

```
kubectl create ns observability
```

To install the Customer Resource Definition:

```
kubectl create -f https://github.com/jaegertracing/jaeger-
operator/releases/download/v1.36.0/jaeger-operator.yaml -n
observability
```

At this point, there should be a **jaeger-operator** deployment available.

```
kubectl get deployment jaeger-operator -n observability
```

```
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
```

```
jaeger-operator   1/1     1            1           29s
```

The operator is now ready to create Jaeger instances.

Deploying the Jaeger All-One image

The operator (that we just installed) doesn't do anything itself, it just means that we can create jaeger resources/instances that we want the Jaeger Operator to manage.

The simplest possible way to create a **Jaeger** instance is by creating a YAML file like the following.

```
vim simplest.yml
```

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simplest
```

The YAML file can then be used with kubectl:

```
kubectl apply -f simplest.yaml
```

This will install the default **All-In-One** strategy, which deploys the all-in-one image, that includes all the following components in a single pod using in-memory storage by default.

- **agent**
- **collector**
- **query**
- **ingester**
- **Jaeger UI**

Source

After a little while, a new in-memory all-in-one instance of **Jaeger** will be available, **suitable for quick demos and development purposes.**

To check the instances that were created, list the **Jaeger** objects:

```
kubectl get jaegers
```

To **get the pod name**, query for the pods belonging to the simplest Jaeger instance:

```
kubectl get pods -l app.kubernetes.io/instance=simplest
```

Query the logs from the pod:

```
kubectl logs -l app.kubernetes.io/instance=simplest
```

Verify Jaeger instance created:

```
kubectl get services -n default | grep jaeger
```

The output should looks like this (some of the output is omitted to fit), but you can see the names and ports in the output.

```
NAME
kubernetes          443/TCP
simplest-agent      5775/UDP,5778/TCP,6831/UDP,6832/UDP
simplest-collector
9411/TCP,14250/TCP,14267/TCP,14268/TCP,4317/TCP,4318/TCP
simplest-collector-headless
9411/TCP,14250/TCP,14267/TCP,14268/TCP,4317/TCP,4318/TCP
simplest-query      16686/TCP,16685/TCP
```

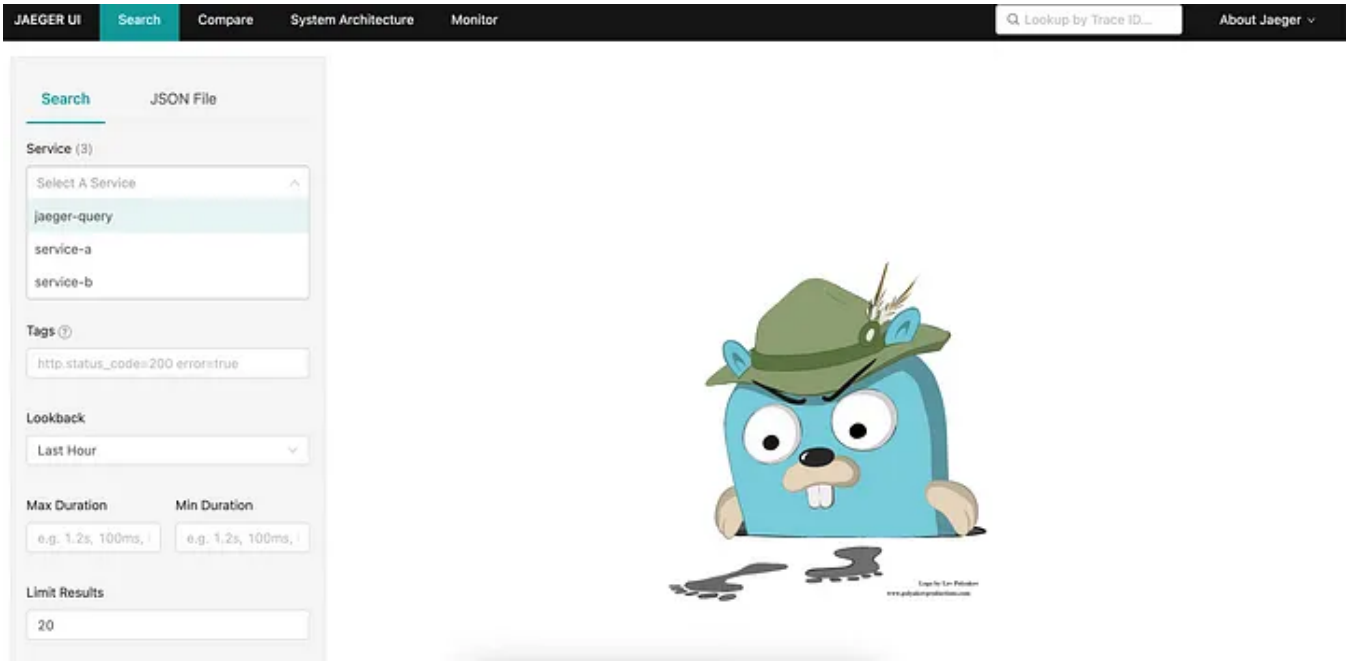
Pay extra attention to **simplest-collector** on port **14268/TCP**

This is the **service** that will be used later on in the **demo application** to send the Jaeger **traces** containing the spans.

Jaeger UI

Open now the Jaeger UI, you can do that by using **port-forwarding**

```
kubectl port-forward svc/simplest-query 16686:16686
```



NodeJS demo application

This repository contains a **NodeJS** demo application, which we can use to deploy on Kubernetes.

```
vim jaeger-nodejs.yaml
```

Make sure to change the name of the **Jaeger collector** to match the one we deployed above.

```
---
apiVersion: v1
kind: Service
metadata:
  name: service-a
  labels:
    app: service-a
spec:
  ports:
    - port: 8080
```

```
      name: http
  selector:
    app: service-a
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-a
  labels:
    app: service-a
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: service-a
  template:
    metadata:
      labels:
        app: service-a
        version: v1
    spec:
      containers:
        - name: app
          image: csantanapr/service-a-nodejs
          env:
            - name: JAEGER_ENDPOINT
              value: http://simplest-collector:14268/api/traces
            - name: SERVICE_FORMATTER
              value: service-b
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: service-b
  labels:
    app: service-b
spec:
  ports:
    - port: 8081
      name: http
  selector:
    app: service-b
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-b
  labels:
    app: service-b
    version: v1
spec:
```

```
  replicas: 1
  selector:
    matchLabels:
      app: service-b
  template:
    metadata:
      labels:
        app: service-b
        version: v1
    spec:
      containers:
        - name: app
          image: csantanapr/service-b-nodejs
          env:
            - name: JAEGER_ENDPOINT
              value: http://simplest-collector:14268/api/traces
          imagePullPolicy: Always
          ports:
            - containerPort: 8081
```

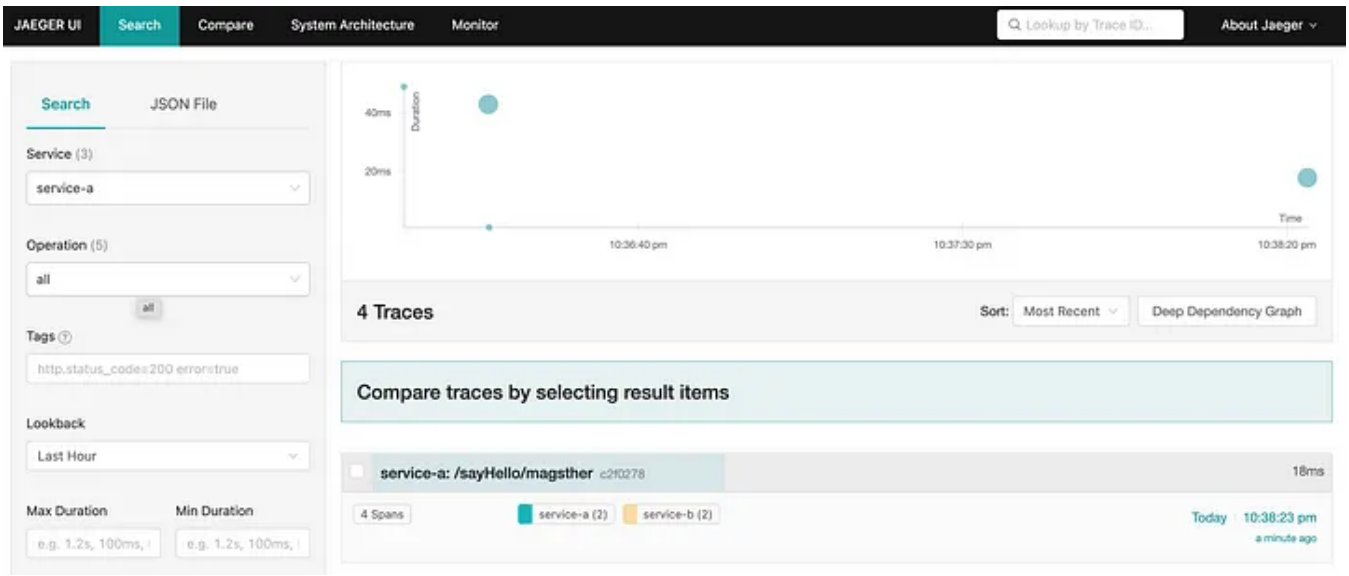
Apply it with : `kubectl apply -f jaeger-nodejs.yaml`

Open the demo application (again using port-forwarding) to access the Application UI

```
kubectl port-forward svc/service-a 8080:8080
```

Make a few requests.

Back in the Jaeger UI, you will now find the **traces**.



All-in-one complex instance

For reference, here's how you can create a more complex all-in-one instance:

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: my-jaeger
spec:
  strategy: allInOne # <1>
  allInOne:
    image: jaegertracing/all-in-one:latest # <2>
    options: # <3>
      log-level: debug # <4>
  storage:
    type: memory # <5>
    options: # <6>
      memory: # <7>
        max-traces: 100000
  ingress:
    enabled: false # <8>
  agent:
    strategy: DaemonSet # <9>
  annotations:
    scheduler.alpha.kubernetes.io/critical-pod: "" # <10>
```

where:

<1> = The default strategy is allInOne. (can also be set to production or streaming)

<2> = The image to use

<3> = Non-storage related options to be passed

<4> = –log-level=debug is passed to the binary.

<5> = The storage type to be used. (default `memory`)

<6> = Storage related options should be placed here

<7> = Some options are namespaced

<8> = By default, an ingress object is created for the query service

<9> = Makes the operator deploy the agent as DaemonSet. (default is `sidecars`)

<10> = Define annotations to be applied to all deployments (not services).

Production Strategy

What if you want to deploy this to **Production**?

The documentation recommends you to use the **production strategy** 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 separately deployed.

In the below configuration, we use **Elasticsearch** as the backend.

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  collector:
    maxReplicas: 5
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: http://elasticsearch:9200
        username: elastic
        password: changeme
```

Deploy it with `kubectl apply -f simple-prod.yaml`

What's next?

In part 1, we presented you the concepts of **Jaeger** (and Hot ROD), we deployed the Jaeger All-in-One image and the Hot ROD images separately. In part 2 we ran everything together with **Docker Compose**.

In part 3 part, we added Elasticsearch to the mix. Elasticsearch is one of the recommend storage backend for tracing.

In this part (4) we deployed Jaeger on Kubernetes.

In the next and final part we will deploy **Jaeger** on **Kubernetes** using **Terraform**.

If you found this useful, please hit that **clap** button and **follow me** to get more articles on your feed.

Follow

Written by Magsther

631 Followers · Writer for FAUN—Developer Community 🐾

Creator of "Awesome OpenTelemetry". https://twitter.com/magsther

More from Magsther and FAUN—Developer Community 🐾

Magsther

GitHub Pages + Hugo

How to create a website using Hugo and host it on GitHub

7 min read · Jun 18, 2023

-- 3

Aymen El Amri in FAUN—Developer Community 🐾

The Hottest Open Source Projects Of 2023

This article was originally posted on faun.dev.

· 14 min read · 5 days ago

-- 4

Sanjit Mohanty in FAUN—Developer Community 🐾

Upgrading from Ingress to Kubernetes Gateway API

Your guide with ingress2gateway!

2 min read · Dec 3, 2023

-- 1

Magsther

JupyterHub on Kubernetes

An introduction to JupyterHub

6 min read · Dec 30, 2022

Pavel Glukhikh

Deploying a Production Kubernetes Cluster in 2023—A Complete Guide

A batteries included guide on deploying a production-ready Kubernetes cluster with external ETCD, load balacing, OCFS2, and Longhorn.

21 min read · Aug 25, 2023

-- 1

Gabriel Dantas

Backstage and Terraform—A Powerful Combination for Ops, Wonderful for Devs

Even as the DevOps/Platform world is in constant evolution, classical challenges persist in many companies, especially those with...

8 min read · Dec 22, 2023

Grafana Authentication with Azure AD and Okta

A recent study on cyber security revealed that over 82% of data breaches were caused by authentication issues—stolen or weak credentials...

5 min read · Dec 21, 2023

 --  

Dipta Roy in AppsCode

Deploy Production-Grade MongoDB Cluster in Rancher Using KubeDB

Overview

7 min read · Jul 21, 2023

 --  

Darío Blanco Iturriaga

Streamline Kubernetes Management: GitOps with Flux and Kustomize

In this blog post, we'll explore the powerful combination of GitOps, Flux, and Kustomize, which together form a winning trio for Kubernetes...

10 min read · Aug 12, 2023

-- 1

Emre Çakmak in AWS in Plain English

Kubernetes Management: Dashboard Setup in AWS EKS, ELB & Route53

Mastering Kubernetes management

5 min read · Jul 13, 2023

--

See more recommendations