How to setup Apache Kafka in OTC CCE

# Table of Contents

[1 Table of Contents 1](#_Toc84437757)

[2 Description 2](#_Toc84437758)

[3 Requirements 2](#_Toc84437759)

[4 Option A. Plain Kubernetes manifests 3](#_Toc84437760)

[4.1 Benefits and Cautions 3](#_Toc84437761)

[4.2 Create Zookeeper deployment 3](#_Toc84437762)

[4.3 Expose Zookeeper service 4](#_Toc84437763)

[4.4 Create Kafka service 4](#_Toc84437764)

[4.5 Create Kafka-Broker deployment 5](#_Toc84437765)

[5 Option B. Official Kafka Helm chart 6](#_Toc84437766)

[5.1 Benefits and Cautions 6](#_Toc84437767)

[5.2 Add Helm chart repository 6](#_Toc84437768)

[5.3 Override default variables as (if) you need 6](#_Toc84437769)

[5.4 Install Helm chart with your variables 6](#_Toc84437770)

[6 Option C. Strimzi Kafka Operator 7](#_Toc84437771)

[6.1 Benefits and Cautions 7](#_Toc84437772)

[6.2 Applying Strimzi installation files 7](#_Toc84437773)

[6.3 Provision Apache Kafka cluster 7](#_Toc84437774)

[6.4 Wait for pods starts 7](#_Toc84437775)

[6.5 Try to send and receive messages 7](#_Toc84437776)

[7 Conclusion 8](#_Toc84437777)

# Description

This is step-by-step guide, which will help you to start with Apache Kafka in OTC Cloud Container Engine. There are 3 different options described in this document. You can choose by your own which exact you need, depending on your use-case.

All tools and their versions described in this article you can find in the [Requirements](#_Requirements) section below.

Please keep in mind that we are not pretending to have production-ready guide that you should follow up without worries. Production systems setup and configuration must be done by persons who have enough experience in Cloud Technologies and Kafka platform.

# Requirements

* OTC CCE cluster
* Kubectl configured for your Kubernetes cluster context properly
* Helm package manager

# **Option A**. Plain Kubernetes manifests

## Benefits and Cautions

This option should be used for testing purposes. No additional tools and pre-configuration steps needed. You are using plain Kubernetes manifests with standard API objects. Configuration as transparent as possible.

Negative side – you don’t have elasticity in terms of configuration. Since there is no any packaging (like helm) you cannot use benefits of versioning and templating. If you need to apply these manifests in different environments with different configuration – you should duplicate your code below.

## Create Zookeeper deployment

zookeeper-deployment.yml

Save snippet below to file

kind: Deployment

apiVersion: apps/v1

metadata:

  name: zookeeper-deploy

spec:

  replicas: 2

  selector:

    matchLabels:

      app: zookeeper-1

  template:

    metadata:

      labels:

        app: zookeeper-1

    spec:

      containers:

      - name: zoo1

        image: zookeeper

        ports:

        - containerPort: 2181

        env:

        - name: ZOOKEEPER\_ID

          value: "1"

        - name: ZOOKEEPER\_SERVER\_1

          value: zoo1

kubectl create -f zookeeper-deployment.yml

Apply changes by

## Expose Zookeeper service

zookeeper-service.yml

Save snippet below to file

apiVersion: v1

kind: Service

metadata:

  name: zoo1

  labels:

    app: zookeeper-1

spec:

  ports:

  - name: client

    port: 2181

    protocol: TCP

  - name: follower

    port: 2888

    protocol: TCP

  - name: leader

    port: 3888

    protocol: TCP

  selector:

    app: zookeeper-1

kubectl create -f zookeeper-service.yml

Apply changes by

## Create Kafka service

kafka-service.yml

Save snippet below to file

apiVersion: v1

kind: Service

metadata:

  name: kafka-service

  labels:

    name: kafka

spec:

  ports:

  - port: 9092

    name: kafka-port

    protocol: TCP

  selector:

    app: kafka

    id: "0"

  type: LoadBalancer

kubectl create -f kafka-service.yml

Apply changes by

## Create Kafka-Broker deployment

broker-deployment.yml

Save snippet below to file

kind: Deployment

apiVersion: apps/v1

metadata:

  name: kafka-broker0

spec:

  replicas: 2

  selector:

    matchLabels:

        app: kafka

        id: "0"

  template:

    metadata:

      labels:

        app: kafka

        id: "0"

    spec:

      containers:

      - name: kafka

        image: wurstmeister/kafka

        ports:

        - containerPort: 9092

        env:

        - name: KAFKA\_ADVERTISED\_PORT

          value: "30718"

        - name: KAFKA\_ADVERTISED\_HOST\_NAME

          value: 192.168.1.240

        - name: KAFKA\_ZOOKEEPER\_CONNECT

          value: zoo1:2181

        - name: KAFKA\_BROKER\_ID

          value: "0"

        - name: KAFKA\_CREATE\_TOPICS

          value: admintome-test:1:1

kubectl create -f broker-deployment.yml

Apply changes by

# **Option B**. Official Kafka Helm chart

## Benefits and Cautions

Most of the things that you usually need with Apache Kafka already present in Helm chart. There are a lot of variables that can help you to get exact configuration you need. Using Helm can simplify transition to GitOps for you.

By the other hand entry level for maintaining this solution a bit bigger, because of templating mechanism complexity. Usually, it does not take much time to sort out with Helm templating mechanism.

## Add Helm chart repository

helm repo add bitnami https://charts.bitnami.com/bitnami

## Override default variables as (if) you need

More information about variables, that can be specified you can find [here](https://github.com/bitnami/charts/tree/master/bitnami/kafka#parameters)

## Install Helm chart with your variables

helm install my-release bitnami/kafka

# Option C. Strimzi Kafka Operator

## Benefits and Cautions

Operators are quite smart in how they manage applications in Kubernetes. Usually, you need to define only high-level parameters like CPU, Memory, Storage, Authentication, Encryption etc. Operator will take care about Kubernetes resources by your requirements. It can automate certificate management.

You have additional abstraction level - complexity of the system potentially can bring problems. Engineers need to have additional knowledge. Besides Cloud Technologies, Kubernetes, Helm they need to know how this exact operator works.

## Applying Strimzi installation files

kubectl create -f 'https://strimzi.io/install/latest?namespace=kafka' -n kafka

## Provision Apache Kafka cluster

kubectl apply -f https://strimzi.io/examples/latest/kafka/kafka-persistent-single.yaml -n kafka

## Wait for pods starts

kubectl wait kafka/my-cluster --for=condition=Ready --timeout=300s -n kafka

## Try to send and receive messages

kubectl -n kafka run kafka-producer -ti --image=quay.io/strimzi/kafka:0.25.0-kafka-2.8.0 --rm=true --restart=Never -- bin/kafka-console-producer.sh --broker-list my-cluster-kafka-bootstrap:9092 --topic my-topic

kubectl -n kafka run kafka-consumer -ti --image=quay.io/strimzi/kafka:0.25.0-kafka-2.8.0 --rm=true --restart=Never -- bin/kafka-console-consumer.sh --bootstrap-server my-cluster-kafka-bootstrap:9092 --topic my-topic --from-beginning

# Conclusion

Here must be some conclusion