Contents

Table of Contents 1
Description 1
Requirements 2
Option A. Plain Kubernetes manifests 2
Option B. Official Kafka Helm chart 5
Option C. Strimzi Kafka Operator 6
Conclusion 6

Table of Contents

Table of Contents
Description
Requiremens
Option A. Plain Kubernetes manifests
  Benefits and Cautions
  Create Zookeeper
  Expose Zookeeper service
  Create Broker
  Expose Broker service
Option B. Official Kafka Helm chart
  Benefits and Cautions
  Setup by Helm
Option C. Strimzi Kafka Operator
  Benefits and Cautions
  Applying Strimzi installation files
  Provision Apache Kafka cluster
  Wait for pods starts
  Try to send and receive messages
Conclusion

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 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 Namespace

```
kubectl create ns kafka
```

Create Zookeeper

- Save snippet below to `zookeeper-statefullset.yml` file:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: zookeeper
spec:
  selector:
    matchLabels:
      app: zookeeper
  serviceName: zookeeper
  replicas: 1
  template:
    metadata:
      labels:
        app: zookeeper
    spec:
      containers:
      - name: zoo1
        image: zookeeper
        imagePullPolicy: IfNotPresent
        resources:
          requests:
            cpu: 128m
            memory: 500Mi
          limits:
            cpu: 128m
            memory: 500Mi
        ports:
        - containerPort: 2181
        env:
        - name: ZK_SERVER_HEAP
          value: "256"
        - name: ZOOKEEPER_ID
          value: "1"
        - name: ZOOKEEPER_SERVER_1
          value: zoo1
```

- Apply changes by `kubectl apply -f zookeeper-statefullset.yml`

Expose Zookeeper service

- Save snippet below to `zookeeper-service.yml` file:

```
apiVersion: v1
kind: Service
metadata:
  name: zookeeper
  labels:
    app: zookeeper
spec:
  ports:
  - name: client
    port: 2181
    protocol: TCP
  - name: follower
    port: 2888
    protocol: TCP
  - name: leader
    port: 3888
    protocol: TCP
  selector:
    app: zookeeper
```

- Apply changes by `kubectl apply -f zookeeper-service.yml`

Create Broker

- Save snippet below to `broker-statefullset.yml` file:

```
apiVersion: apps/v1
kind: StatefulSet
```

```
metadata:
  name: broker
spec:
  selector:
    matchLabels:
      app: broker
  serviceName: broker
  replicas: 1
  template:
    metadata:
      labels:
        app: broker
    spec:
      containers:
      - name: kafka
        image: wurstmeister/kafka
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9092
        - containerPort: 9094
        resources:
          requests:
            cpu: 128m
            memory: 1Gi
          limits:
            cpu: 128m
            memory: 1Gi
        env:
        - name: "KAFKA_HEAP_OPTS"
          value: "-Xmx512M -Xms512M"
        - name: KAFKA_LISTENERS
          value: "INSIDE://:9094,OUTSIDE://localhost:9092"
        - name: KAFKA_ADVERTISED_LISTENERS
          value: "INSIDE://:9094,OUTSIDE://localhost:9092"
        - name: KAFKA_LISTENER_SECURITY_PROTOCOL_MAP
          value: "INSIDE:PLAINTEXT,OUTSIDE:PLAINTEXT"
        - name: KAFKA_INTER_BROKER_LISTENER_NAME
          value: INSIDE
        - name: KAFKA_ZOOKEEPER_CONNECT
          value: zookeeper:2181
        - name: KAFKA_BROKER_ID
          value: "0"
```

- Apply changes by `kubectl apply -f broker-statefullset.yml`

Expose Broker service

- Save snippet below to `kafka-service.yml` file:

  ```
  apiVersion: v1
  kind: Service
  metadata:
    name: broker
    labels:
      app: broker
  spec:
    ports:
    - port: 9092
      name: broker-port
      protocol: TCP
    selector:
      app: broker
    type: ClusterIP
  ```

  Apply changes by `kubectl apply -f kafka-service.yml`

Try to send and receive messages

- Forward Broker service to your local machine

  ```
  kubectl port-forward service/broker -n kafka 9092:9092
  ```

- Produce something like

  ```
  kcat -b localhost:9092 -t test-topic -P <<EOF
  hello
  world
  EOF
  ```

- Consume it by

  ```
  kcat -b localhost:9092 -t test-topic -C
  ```

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.

Setup by Helm

- 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 overrided you can find here

- 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

Provision Apache Kafka cluster

Wait for pods starts

Try to send and receive messages

Conclusion

Here must be some conclusion