Strimzi Quick Start

Table of Contents

Chapter 1. Overview of Strimzi

Strimzi simplifies the process of running Apache Kafka in a Kubernetes cluster.

This guide provides instructions for evaluating a working environment of Strimzi. The steps describe how to get a Strimzi deployment up-and-running as quickly as possible.

Before trying Strimzi, it is useful to understand its capabilities and how you might wish to use it. This chapter introduces some of the key concepts behind Kafka, and also provides a brief overview of the Strimzi Operators.

Operators are a method of packaging, deploying, and managing a Kubernetes application. Strimzi Operators extend Kubernetes functionality, automating common and complex tasks related to a Kafka deployment. By implementing knowledge of Kafka operations in code, Kafka administration tasks are simplified and require less manual intervention.

1.1. Kafka capabilities

The underlying data stream-processing capabilities and component architecture of Kafka can deliver:

- Microservices and other applications to share data with extremely high throughput and low latency
- Message ordering guarantees
- Message rewind/replay from data storage to reconstruct an application state
- Message compaction to remove old records when using a key-value log
- Horizontal scalability in a cluster configuration
- Replication of data to control fault tolerance
- Retention of high volumes of data for immediate access

1.2. Kafka use cases

Kafka’s capabilities make it suitable for:

- Event-driven architectures
- Event sourcing to capture changes to the state of an application as a log of events
- Message brokering
- Website activity tracking
- Operational monitoring through metrics
- Log collection and aggregation
- Commit logs for distributed systems
- Stream processing so that applications can respond to data in real time

1.3. How Strimzi supports Kafka

Strimzi provides container images and Operators for running Kafka on Kubernetes. Strimzi Operators are fundamental to the running of Strimzi. The Operators provided with Strimzi are purpose-built with specialist operational knowledge to effectively manage Kafka.

Operators simplify the process of:

- Deploying and running Kafka clusters
- Deploying and running Kafka components
- Configuring access to Kafka
- Securing access to Kafka
- Upgrading Kafka
- Managing brokers
- Creating and managing topics
- Creating and managing users

1.4. Operators

Strimzi provides Operators for managing a Kafka cluster running within a Kubernetes cluster.

Cluster Operator

Deploys and manages Apache Kafka clusters, Kafka Connect, Kafka MirrorMaker, Kafka Bridge, Kafka Exporter, Cruise Control, and the Entity Operator

Entity Operator

Comprises the Topic Operator and User Operator

Topic Operator

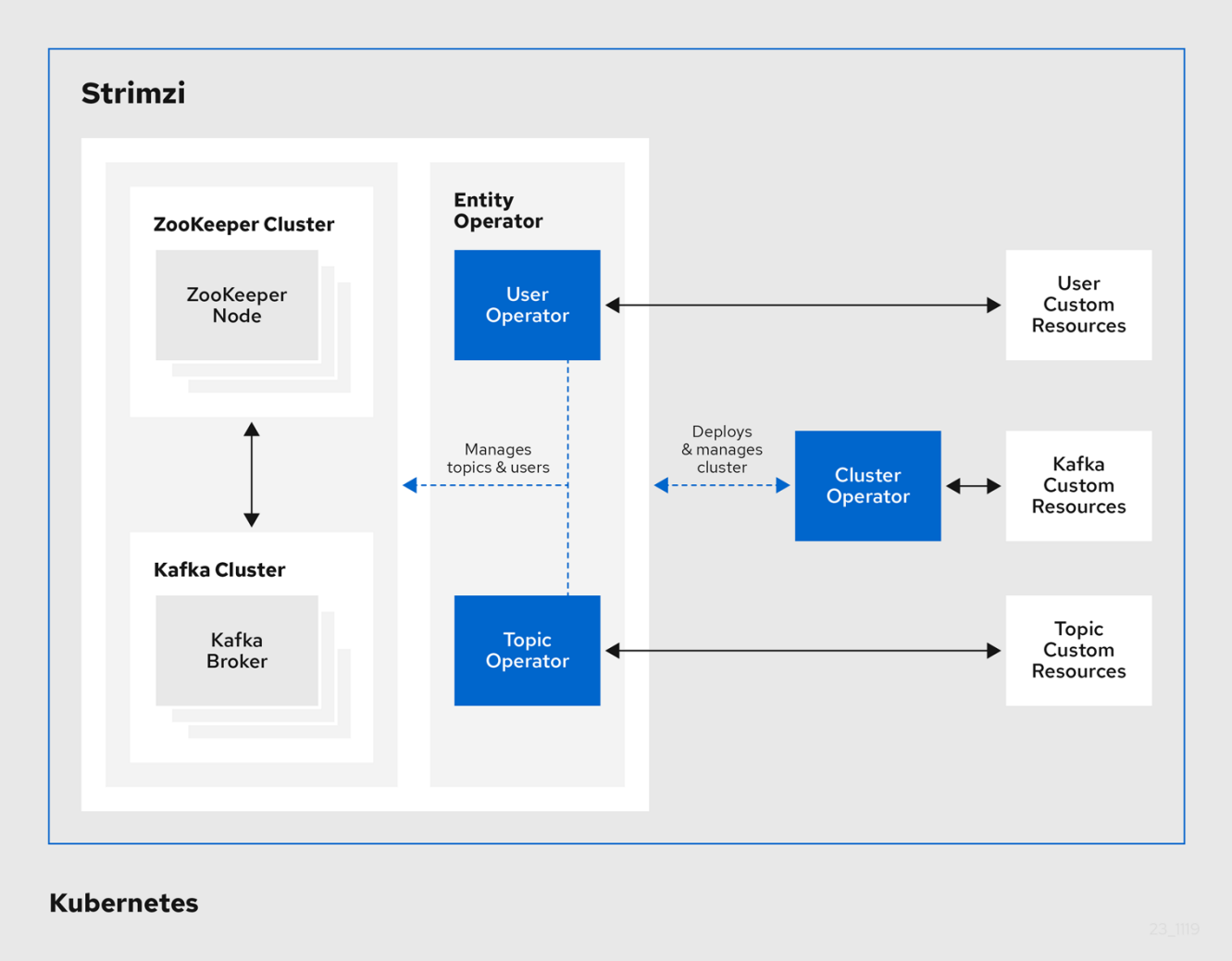
Manages Kafka topics

User Operator

Manages Kafka users

The Cluster Operator can deploy the Topic Operator and User Operator as part of an **Entity Operator** configuration at the same time as a Kafka cluster.

Operators within the Strimzi architecture



1.5. Document Conventions

Replaceables

In this document, replaceable text is styled in `monospace`, with italics, uppercase, and hyphens.

For example, in the following code, you will want to replace `MY-NAMESPACE` with the name of your namespace:

```
sed -i 's/namespace: .*/namespace: MY-NAMESPACE/' install/cluster-
operator/*RoleBinding*.yaml
```

Chapter 2. Evaluate Strimzi

The procedures in this chapter provide a quick way to evaluate the functionality of Strimzi.

Follow the steps in the order provided to install Strimzi, and start sending and receiving messages from a topic:

- Ensure you have the required prerequisites
- Install and start Minikube
- Install Strimzi
- Create a Kafka cluster
- Access the Kafka cluster to send and receive messages

2.1. Prerequisites

- Install and start Minikube.
- You need to be able to access Strimzi GitHub.

2.2. Downloading Strimzi

The resources and artifacts required to install Strimzi, along with examples for configuration, are provided in a ZIP file.

Procedure

1. Download the `strimzi-x.y.z.zip` file from GitHub.
2. Unzip the file to any destination.

    - Windows or Mac: Extract the contents of the ZIP archive by double-clicking on the ZIP file.
    - Linux: Open a terminal window in the target machine and navigate to where the ZIP file was downloaded.

    Extract the ZIP file with this command:

    ```
    unzip strimzi-xyz.zip
    ```

2.3. Installing Strimzi

Using the download files, install Strimzi with the Custom Resource Definitions (CRDs) and RBAC configuration required for deployment.

In this task you create namespaces in the cluster for your deployment. Use namespaces to separate functions.

Prerequisites

- Installation requires a Kubernetes account with cluster admin credentials.

Procedure

1. Log in to the Kubernetes cluster using an account that has cluster admin privileges.
2. Create a new `kafka` namespace for the Strimzi Kafka Cluster Operator.

```
kubectl create ns kafka
```

3. Modify the installation files to reference the `kafka` namespace where you will install the Strimzi Kafka Cluster Operator.

> **NOTE** By default, the files work in the `myproject` namespace.

   - On Linux, use:

```
sed -i 's/namespace: .*/namespace: kafka/' install/cluster-
operator/*RoleBinding*.yaml
```

   - On Mac, use:

```
sed -i '' 's/namespace: .*/namespace: kafka/' install/cluster-
operator/*RoleBinding*.yaml
```

4. Create a new `my-kafka-project` namespace where you will deploy your Kafka cluster.

```
kubectl create ns my-kafka-project
```

5. Edit the `install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml` file and set the `STRIMZI_NAMESPACE` environment variable to the namespace `my-kafka-project`.

```
# ...
env:
- name: STRIMZI_NAMESPACE
  value: my-kafka-project
# ...
```

6. Deploy the CRDs and role-based access control (RBAC) resources to manage the CRDs.

```
kubectl create -f install/cluster-operator/ -n kafka
```

7. Give permission to the Cluster Operator to watch the `my-kafka-project` namespace.

```
kubectl create -f install/cluster-operator/020-RoleBinding-strimzi-cluster-
operator.yaml -n my-kafka-project
```

```
kubectl create -f install/cluster-operator/031-RoleBinding-strimzi-cluster-
operator-entity-operator-delegation.yaml -n my-kafka-project
```

The commands create role bindings that grant permission for the Cluster Operator to access the Kafka cluster.

2.4. Creating a cluster

With Strimzi installed, you create a Kafka cluster, then a topic within the cluster.

When you create a cluster, the Cluster Operator you deployed when installing Strimzi watches for new Kafka resources.

Prerequisites

- For the Kafka cluster, ensure a Cluster Operator is deployed.
- For the topic, you must have a running Kafka cluster.

Procedure

1. Log in to the Kubernetes cluster as a non-privileged user.
2. Create a new `my-cluster` Kafka cluster with one ZooKeeper and one Kafka broker.
    - Use `persistent-claim` storage
    - Expose the Kafka cluster outside of the Kubernetes cluster using an external listener configured to use a `nodeport`.

```
cat << EOF | kubectl create -n my-kafka-project -f -
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 1
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
      - name: external
        port: 9094
        type: nodeport
        tls: false
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
    config:
      offsets.topic.replication.factor: 1
      transaction.state.log.replication.factor: 1
      transaction.state.log.min.isr: 1
      default.replication.factor: 1
      min.insync.replicas: 1
  zookeeper:
    replicas: 1
    storage:
      type: persistent-claim
      size: 100Gi
      deleteClaim: false
  entityOperator:
    topicOperator: {}
    userOperator: {}
EOF
```

3. Wait for the cluster to be deployed:

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

4. When your cluster is ready, create a topic to publish and subscribe from your external client.

   Create the following `my-topic` custom resource definition with 3 partitions and replication factor 1 in the `my-cluster` Kafka cluster:

```
cat << EOF | kubectl create -n my-kafka-project -f -
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: my-topic
  labels:
    strimzi.io/cluster: "my-cluster"
spec:
  partitions: 3
  replicas: 1
EOF
```

2.5. Sending and receiving messages from a topic

You can test your Strimzi installation by sending and receiving messages to `my-topic` from outside the cluster.

Use a terminal to run a Kafka producer and consumer on a local machine.

Prerequisites

- Ensure Strimzi is installed on the Kubernetes cluster.
- ZooKeeper and Kafka must be running to be able to send and receive messages.

Procedure

1. Download the latest Kafka binaries and install Kafka on your local machine.

   Apache Kafka download

2. Find the port of the bootstrap service:

```
kubectl get service my-cluster-kafka-external-bootstrap -n my-kafka-project
-o=jsonpath='{.spec.ports[0].nodePort}{"\n"}'
```

3. Find the IP address of the Minikube node:

```
kubectl get nodes --output=jsonpath='{range
.items[*]}{.status.addresses[?(@.type=="InternalIP")].address}{"\n"}{end}'
```

4. Open a terminal, and start the Kafka console producer with the topic `my-topic`:

```
bin/kafka-console-producer.sh --broker-list <node-address>:_<node-port>_ --topic
my-topic
```

5. Type your message into the console where the producer is running.
6. Open a new terminal tab or window, and start the consumer to receive the messages:

```
bin/kafka-console-consumer.sh --bootstrap-server <node-address>:_<node-port>_
--topic my-topic --from-beginning
```

7. Verify that you see the incoming messages in the consumer console.
8. Press Crtl+C to exit the Kafka console producer and consumer.