//the producer configuration

spring.kafka.bootstrap-servers=my-cluster-kafka-bootstrap:9092

//this binding is defined without function to be used with streambridge

spring.cloud.stream.bindings.order-flow-binding.destination=order-flow

spring.cloud.stream.bindings.order-flow-binding.content-type: application/json

//very important to encode json correctly

spring.cloud.stream.bindings.order-flow-binding.producer.use-native-encoding=true

#spring.cloud.stream.bindings.order-flow-binding.producer.header-mode: embeddedHeaders

#spring.kafka.consumer.group-id=order

#spring.kafka.consumer.key-deserializer: org.apache.kafka.common.serialization.StringDeserializer

#spring.kafka.consumer.value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer

#spring.kafka.producer.key-serializer: org.apache.kafka.common.serialization.StringSerializer

#spring.kafka.producer.properties.spring.json.add.type.headers=false

spring.kafka.producer.value-serializer: org.springframework.kafka.support.serializer.JsonSerializer

//very important to map type in the header and allow converting it in the consumer

spring.kafka.producer.properties.spring.json.type.mapping=OrderCreatedEvent:com.beshara.fintech.order.domain.events.OrderCreatedEvent

#spring.kafka.consumer.properties.spring.json.trusted.packages=\*

//the consumer configuration

spring.kafka.bootstrap-servers=my-cluster-kafka-bootstrap:9091,my-cluster-kafka-bootstrap:9092,my-cluster-kafka-bootstrap:9093

#spring.cloud.stream.kafka.binder.brokers=my-cluster-kafka-brokers

#spring.cloud.stream.kafka.binder.auto-add-partitions=false

#spring.cloud.stream.kafka.binder.consider-down-when-any-partition-has-no-leader=false

##spring.cloud.stream.kafka.binder.brokers=my-cluster-kafka-brokers:9092

#spring.cloud.stream.kafka.binder.brokers=9092

##spring.cloud.stream.kafka.binder.zkNodes=my-cluster-zookeeper-nodes

#spring.cloud.stream.kafka.binder.defaultZkPort=2181

#spring.kafka.brokers=zookeeper:2181

#spring.kafka.admin.auto-create=false

# spring.cloud.stream.default.group=product

spring.cloud.function.definition= handleOrderCreation

spring.cloud.stream.bindings.handleOrderCreation-in-0.destination: order-flow

##spring.cloud.stream.bindings.handleOrderCreation-in-0.consumer.

#spring.cloud.stream.kafka.binder.auto-create-topics=false

spring.cloud.stream.bindings.handleOrderCreation-in-0.group=product

#spring.cloud.stream.bindings.handleOrderCreation-in-0.content-type=application/json

spring.kafka.consumer.properties.spring.json.type.mapping=OrderCreatedEvent:com.beshara.fintech.product.domain.events.OrderCreatedEvent

#spring.cloud.stream.bindings.handleOrderCreation-in-0.consumer.value-deserializer= org.springframework.kafka.support.serializer.JsonDeserializer

#spring.kafka.consumer.key-deserializer: org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.consumer.value-deserializer= org.springframework.kafka.support.serializer.JsonDeserializer

#spring.kafka.producer.key-serializer: org.apache.kafka.common.serialization.StringSerializer

spring.kafka.producer.value-serializer: org.springframework.kafka.support.serializer.JsonSerializer

spring.kafka.consumer.properties.spring.json.trusted.packages=\*

check <https://stackoverflow.com/questions/47397603/kafka-producer-json-serialization>

installing kafka stimzi on kubernetes

// run the commands from the powershell to work

Minikube provides a local Kubernetes, designed to make it easy to learn and develop for Kubernetes. The Kubernetes cluster is started either inside a virtual machine, a container or on bare-metal, depending on the minikube driver you choose.

**Installing the dependencies**

This quickstart assumes that you have the latest version of the minikube binary, which you can get from the [minikube website](https://minikube.sigs.k8s.io/docs/start/).

Minikube requires a container or virtual machine manager. The Minikube documentation includes a list of suggested options in the [getting started guide](https://minikube.sigs.k8s.io/docs/start/).

You’ll also need the kubectl binary, which you can get by following the [kubectl installation instructions](https://kubernetes.io/docs/tasks/tools/) from the Kubernetes website.

Once you have all the binaries installed, make sure everything works:

# Validate minikube

minikube version

# Validate kubectl

kubectl version

**Starting the Kubernetes cluster**

Start a local development cluster of [Minikube](https://minikube.sigs.k8s.io/docs/start/) that runs in a container or virtual machine manager.

**minikube** start --memory=4096 # 2GB default memory isn't always enough

**Deploy Strimzi using installation files**

Before deploying the Strimzi cluster operator, create a namespace called kafka:

kubectl create **namespace** kafka

Apply the Strimzi install files, including ClusterRoles, ClusterRoleBindings and some **Custom Resource Definitions** (CRDs). The CRDs define the schemas used for the custom resources (CRs, such as Kafka, KafkaTopic and so on) you will be using to manage Kafka clusters, topics and users.

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

The YAML files for ClusterRoles and ClusterRoleBindings downloaded from strimzi.io contain a default namespace of myproject. The query parameter namespace=kafka updates these files to use kafka instead. By specifying -n kafka when running kubectl create, the definitions and configurations without a namespace reference are also installed in the kafka namespace. If there is a mismatch between namespaces, then the Strimzi cluster operator will not have the necessary permissions to perform its operations.

Follow the deployment of the Strimzi cluster operator:

kubectl **get** pod -n kafka --watch

You can also follow the operator’s log:

kubectl logs deployment/strimzi-cluster-operator -n kafka -f

Once the operator is running it will watch for new custom resources and create the Kafka cluster, topics or users that correspond to those custom resources.

**Create an Apache Kafka cluster**

Create a new Kafka custom resource to get a single node Apache Kafka cluster:

# Apply the `Kafka` Cluster CR file

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

Wait while Kubernetes starts the required pods, services, and so on:

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

The above command might timeout if you’re downloading images over a slow connection. If that happens you can always run it again.

**Send and receive messages**

With the cluster running, run a simple producer to send messages to a Kafka topic (the topic is automatically created):

kubectl -n kafka run kafka-producer -ti --image=quay.io/strimzi/kafka:0.42.0-kafka-3.7.1 --rm=true --restart=Never -- bin/kafka-console-producer.sh --bootstrap-server my-cluster-kafka-bootstrap:9092 --topic my-topic

Once everything is set up correctly, you’ll see a prompt where you can type in your messages:

If you don't see a command prompt, try pressing enter.

>Hello Strimzi!

And to receive them in a different terminal, run:

kubectl -n kafka run kafka-consumer -ti --image=quay.io/strimzi/kafka:0.42.0-kafka-3.7.1 --rm=true --restart=Never -- bin/kafka-console-consumer.sh --bootstrap-server my-cluster-kafka-bootstrap:9092 --topic my-topic --**from**-beginning

If everything works as expected, you’ll be able to see the message you produced in the previous step:

If you don't see a command prompt, try pressing enter.

>Hello Strimzi!

Enjoy your Apache Kafka cluster, running on Minikube!

**Deleting your Apache Kafka cluster**

When you are finished with your Apache Kafka cluster, you can delete it by running:

kubectl -n kafka **delete** $(kubectl **get** strimzi -o **name** -n kafka)

This will remove all Strimzi custom resources, including the Apache Kafka cluster and any KafkaTopic custom resources but leave the Strimzi cluster operator running so that it can respond to new Kafka custom resources.

Next, delete the Persistent Volume Claim (PVC) that was used by the cluster:

kubectl **delete** pvc -l strimzi.io/name=**my**-cluster-kafka -n kafka

Without deleting the PVC, the next Kafka cluster you might start will fail as it will try to use the volume that belonged to the previous Apache Kafka cluster.

**Deleting the Strimzi cluster operator**

When you want to fully remove the Strimzi cluster operator and associated definitions, you can run:

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

**Deleting the**kafka**namespace**

Once it is not used, you can also delete the Kubernetes namespace:

kubectl **delete** **namespace** kafka