Release H: Quarkus with Kafka

- Introduction
  - Code
    - KafkaUser
    - JKS secret
    - kafka-streams-quickstart-producer application.properties
    - kafka-streams-quickstart-aggregator application.properties
    - quarkus-producer
    - quarkus-aggregator
- Links

Introduction

Quarkus along with GraalVM allows developers to maximize the performance of their java applications while minimizing the size of the images used.

This is done by compiling the code into a native executable.

Code

This code is based on the USING APACHE KAFKA STREAMS example.

Create the 2 java projects specified in the code above: kafka-streams-quickstart-producer and kafka-streams-quickstart-aggregator.

Start Strimzi and make sure you have setup port 9093 to authenticate with tls.

Strimzi listener

```
      listeners:
      - name: plain
        port: 9092
        tls: false
        type: internal
      - name: tls
        port: 9093
        tls: true
        type: internal
        authentication:
          type: tls
```

KafkaUser

Once Strimzi is up and running we need to create a user for our applications, we'll call them quarkus.

quarkus user

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: quarkus
  namespace: kafka
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
  authorization:
    type: simple
    acls:
      # access to the topic
      - resource:
          type: topic
          name: temperature-values
        operations:
          - Create
          - Describe
          - Read
          - Write
          - AlterConfigs
        host: "*"
      - resource:
          type: topic
          name: weather-stations
        operations:
          - Create
          - Describe
          - Read
          - Write
          - AlterConfigs
        host: "*"
      - resource:
          type: topic
          name: kafka-streams-quickstart-aggregator-weather-stations-store-changelog
        operations:
          - Create
          - Describe
          - Read
          - Write
          - AlterConfigs
        host: "*"
      - resource:
          type: topic
          name: temperatures-aggregated
        operations:
          - Create
          - Describe
          - Read
          - Write
          - AlterConfigs
        host: "*"
      # access to the group
      - resource:
          type: group
          name: kafka-streams-quickstart-aggregator
        operations:
          - Describe
          - Read
        host: "*"
```

This will give the quarkus user access to the following topics: temperature-values, weather-stations, kafka-streams-quickstart-aggregator-weather-stations-store-changelog, temperatures-aggregated and the following group: kafka-streams-quickstart-aggregator.

JKS secret

Once the user is created we can create a secret to store our keystore, truststore and passwords.

JKS secret script

```
#!/bin/sh
kafkauser="quarkus"

rm ca.crt user.crt  user.key  user-keystore.jks  user.p12  user.password  user-truststore.jks 2>/dev/null

kubectl delete secret ${kafkauser}-jks -n kafka 2>/dev/null

kubectl get secret my-cluster-cluster-ca-cert -n kafka -o jsonpath='{.data.ca\.crt}' | base64 --decode > ca.crt

kubectl get secret ${kafkauser} -n kafka -o jsonpath='{.data.user\.key}' | base64 --decode > user.key

kubectl get secret ${kafkauser} -n kafka -o jsonpath='{.data.user\.crt}' | base64 --decode > user.crt

kubectl get secret ${kafkauser} -n kafka -o jsonpath='{.data.user\.p12}' | base64 --decode > user.p12

kubectl get secret ${kafkauser} -n kafka -o jsonpath='{.data.user\.password}' | base64 --decode > user.password

export PASSWORD=`cat user.password`
echo $PASSWORD

keytool -import -trustcacerts -file ca.crt -keystore user-truststore.jks -storepass $PASSWORD -noprompt

keytool -importkeystore -srckeystore user.p12 -srcstorepass ${PASSWORD} -srcstoretype pkcs12 -destkeystore user-
keystore.jks -deststorepass ${PASSWORD} -deststoretype jks

kubectl create secret generic ${kafkauser}-jks -n kafka --from-literal=keystore_password=$PASSWORD --from-
file=user-keystore.jks=user-keystore.jks --from-literal=truststore_password=$PASSWORD --from-file=user-
truststore.jks=user-truststore.jks --from-literal=key_password=$PASSWORD
```

We need to make some changes to the application.properties file to enable communication with Strimzi over SSL.

kafka-streams-quickstart-producer application.properties

application.properties

```
# Configure the Kafka broker location
kafka.bootstrap.servers=my-cluster-kafka-bootstrap.kafka:9093
kafka.security.protocol=SSL
kafka.ssl.keystore.location=/etc/ssl/user-keystore.jks
kafka.ssl.keystore.password=${KEYSTORE_PASSWORD}
kafka.ssl.key.password=${KEY_PASSWORD}
kafka.ssl.keystore.type=JKS
kafka.ssl.truststore.location=/etc/ssl/user-truststore.jks
kafka.ssl.truststore.password=${TRUSTSTORE_PASSWORD}
kafka.ssl.truststore.type=JKS

mp.messaging.outgoing.temperature-values.connector=smallrye-kafka
mp.messaging.outgoing.temperature-values.key.serializer=org.apache.kafka.common.serialization.IntegerSerializer
mp.messaging.outgoing.temperature-values.value.serializer=org.apache.kafka.common.serialization.StringSerializer

mp.messaging.outgoing.weather-stations.connector=smallrye-kafka
mp.messaging.outgoing.weather-stations.key.serializer=org.apache.kafka.common.serialization.IntegerSerializer
mp.messaging.outgoing.weather-stations.value.serializer=org.apache.kafka.common.serialization.StringSerializer
```

kafka-streams-quickstart-aggregator application.properties

application.properties

```
kafka.bootstrap.servers=my-cluster-kafka-bootstrap.kafka:9093
kafka.security.protocol=SSL
kafka.ssl.keystore.location=/etc/ssl/user-keystore.jks
kafka.ssl.keystore.password=${KEYSTORE_PASSWORD}
kafka.ssl.key.password=${KEY_PASSWORD}
kafka.ssl.keystore.type=JKS
kafka.ssl.truststore.location=/etc/ssl/user-truststore.jks
kafka.ssl.truststore.password=${TRUSTSTORE_PASSWORD}
kafka.ssl.truststore.type=JKS

quarkus.kafka-streams.application-server=${hostname}:8080
quarkus.kafka-streams.topics=weather-stations,temperature-values

# pass-through options
kafka-streams.cache.max.bytes.buffering=10240
kafka-streams.commit.interval.ms=1000
kafka-streams.metadata.max.age.ms=500
```

We can then compile our code.

Change to the project directory (cd kafka-streams-quickstart-aggregator)

We can create a normal jar file and image using the following commands:

mvn clean package

docker build -f src/main/docker/Dockerfile.jvm . -t <namespace tag>/quarkus-aggregator

or a native build and image using the following commands:

./mvnw package -Pnative -Dquarkus.native.remote-container-build=true

docker build -f src/main/docker/Dockerfile.native . -t <namespace tag>/quarkus-aggregator-native

Run both builds for both projects.

The native build is very slow and takes a long time to complete.

When we are finished we should have 4 docker images:

Quarkus images

```
docker images | grep quarkus
ktimoney/quarkus-aggregator-native                                                                  latest
            92c052f92b58   10 minutes ago      169MB
ktimoney/quarkus-producer-native                                                                    latest
            66cc3a1e35f2   About an hour ago   149MB
ktimoney/quarkus-producer                                                                           latest
            d6703ba8b500   3 hours ago         421MB
ktimoney/quarkus-aggregator                                                                         latest
            22712c462e7b   3 hours ago         478MB
```

As you can see the native images are less than half the size of the standard jar images.

Lastly we'll create our deployment files.

quarkus-producer

quarkus-producer

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quarkus-producer
  namespace: kafka
spec:
  selector:
    matchLabels:
      app: quarkus-producer
  template:
    metadata:
      labels:
        app: quarkus-producer
        version: v1
    spec:
      containers:
      - name: quarkus-producer
        image: ktimoney/quarkus-producer-native
        imagePullPolicy: IfNotPresent
        env:
        - name: KEYSTORE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: quarkus-jks
              key: keystore_password
        - name: TRUSTSTORE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: quarkus-jks
              key: truststore_password
        - name: KEY_PASSWORD
          valueFrom:
            secretKeyRef:
              name: quarkus-jks
              key: key_password
        volumeMounts:
        - name: jks
          mountPath: /etc/ssl/
          readOnly: true
      volumes:
      - name: jks
        secret:
          secretName: quarkus-jks
---
apiVersion: v1
kind: Service
metadata:
  name: quarkus-producer
  namespace: kafka
  labels:
    app: quarkus-producer
    service: quarkus-producer
spec:
  type: ClusterIP
  selector:
    app: quarkus-producer
  ports:
  - port: 8080
    name: http-80
```

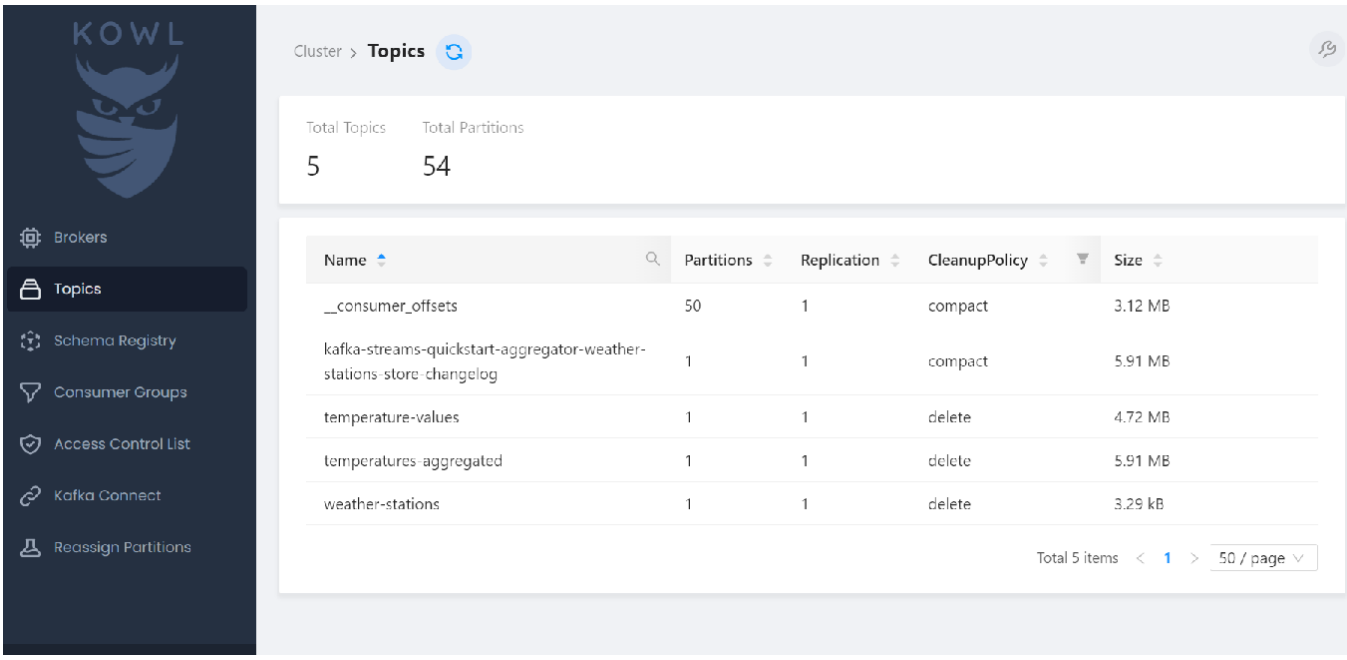
quarkus-aggregator

quarkus-aggregator

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quarkus-aggregator
  namespace: kafka
spec:
  selector:
    matchLabels:
      app: quarkus-aggregator
  template:
    metadata:
      labels:
        app: quarkus-aggregator
        version: v1
    spec:
      containers:
      - name: quarkus-aggregator
        image: ktimoney/quarkus-aggregator-native
        imagePullPolicy: IfNotPresent
        env:
        - name: KEYSTORE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: quarkus-jks
              key: keystore_password
        - name: TRUSTSTORE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: quarkus-jks
              key: truststore_password
        - name: KEY_PASSWORD
          valueFrom:
            secretKeyRef:
              name: quarkus-jks
              key: key_password
        volumeMounts:
        - name: jks
          mountPath: /etc/ssl/
          readOnly: true
      volumes:
      - name: jks
        secret:
          secretName: quarkus-jks
---
apiVersion: v1
kind: Service
metadata:
  name: quarkus-aggregator
  namespace: kafka
  labels:
    app: quarkus-aggregator
    service: quarkus-aggregator
spec:
  type: ClusterIP
  selector:
    app: quarkus-aggregator
  ports:
  - port: 8080
    name: http-80
```

Once the pods are up and running you'll see the quarkus-producer writing records to the "temperature-values" and "weather stations" topics. quarkus-aggregator will write records to the "temperature-aggregated".

The "kafka-streams-quickstart-aggregator-weather-stations-store-changelog" is an internal topic used by the Kafka streams API.



Links

Create a native executable