

Paste the following content:  
version: '3.5'

services:

zookeeper:

image: quay.io/strimzi/kafka:0.41.0-kafka-3.7.0

command: [

"sh", "-c",

"bin/zookeeper-server-start.sh config/zookeeper.properties"

]

ports:

- "2181:2181"

environment:

LOG\_DIR: /tmp/logs

networks:

- kafkastreams-network

kafka:

image: quay.io/strimzi/kafka:0.41.0-kafka-3.7.0

command: [

"sh", "-c",

"bin/kafka-server-start.sh config/server.properties --override listeners=$${KAFKA\_LISTENERS} --override advertised.listeners=$${KAFKA\_ADVERTISED\_LISTENERS} --override zookeeper.connect=$${KAFKA\_ZOOKEEPER\_CONNECT} --override num.partitions=$${KAFKA\_NUM\_PARTITIONS}"

]

depends\_on:

- zookeeper

ports:

- "9092:9092"

environment:

LOG\_DIR: "/tmp/logs"

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka:9092

KAFKA\_LISTENERS: PLAINTEXT://0.0.0.0:9092

KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

KAFKA\_NUM\_PARTITIONS: 3

networks:

- kafkastreams-network

producer:

image: quarkus-quickstarts/kafka-streams-producer:1.0

build:

context: producer

dockerfile: src/main/docker/Dockerfile.${QUARKUS\_MODE:-jvm}

environment:

KAFKA\_BOOTSTRAP\_SERVERS: kafka:9092

networks:

- kafkastreams-network

aggregator:

image: quarkus-quickstarts/kafka-streams-aggregator:1.0

build:

context: aggregator

dockerfile: src/main/docker/Dockerfile.${QUARKUS\_MODE:-jvm}

environment:

QUARKUS\_KAFKA\_STREAMS\_BOOTSTRAP\_SERVERS: kafka:9092

networks:

- kafkastreams-network

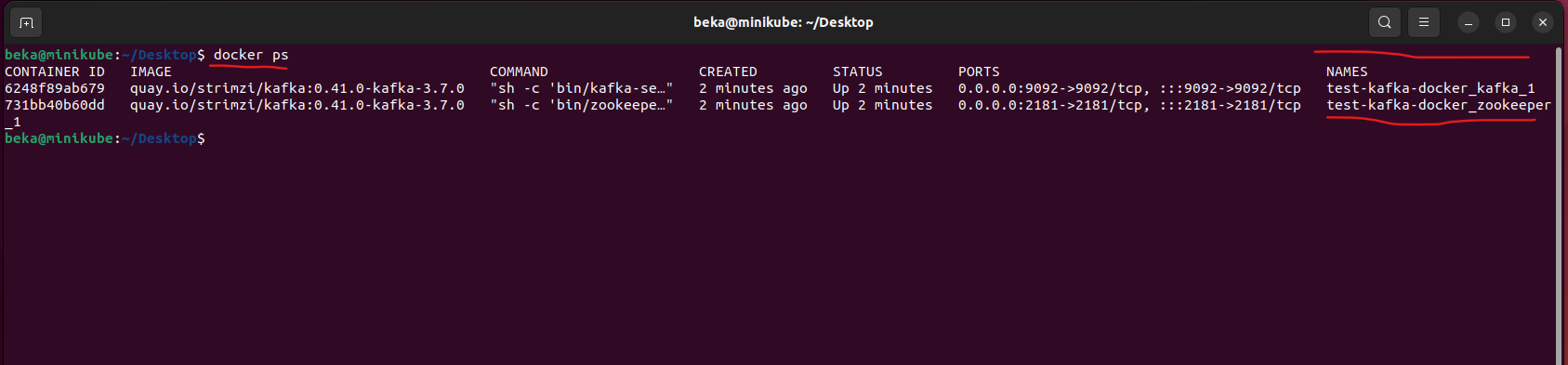
networks:

kafkastreams-network:

name: ks

Create Kafka docker

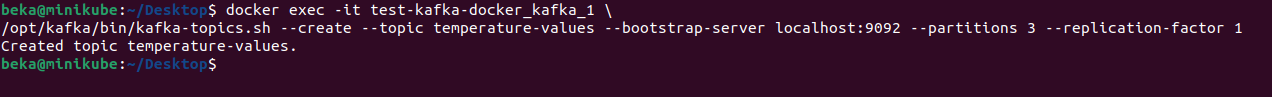




**Create temperature-values topic**:

docker exec -it test-kafka-docker\_kafka\_1 \

/opt/kafka/bin/kafka-topics.sh --create --topic **temperature-values** --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1

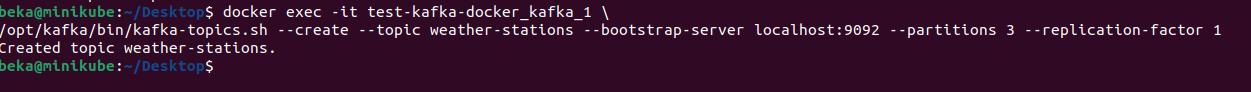


Create Three Kafka toppics:

**Create** **weather-stations** **topic**:

docker exec -it test-kafka-docker\_kafka\_1 \

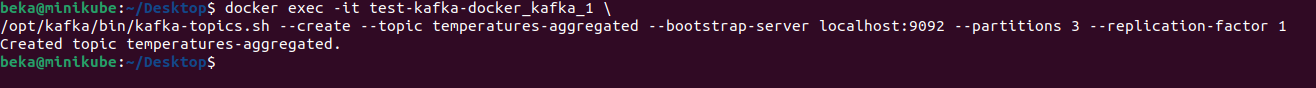
/opt/kafka/bin/kafka-topics.sh --create --topic **weather-stations** --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1



**Create temperatures-aggregated topic**:

docker exec -it test-kafka-docker\_kafka\_1 \

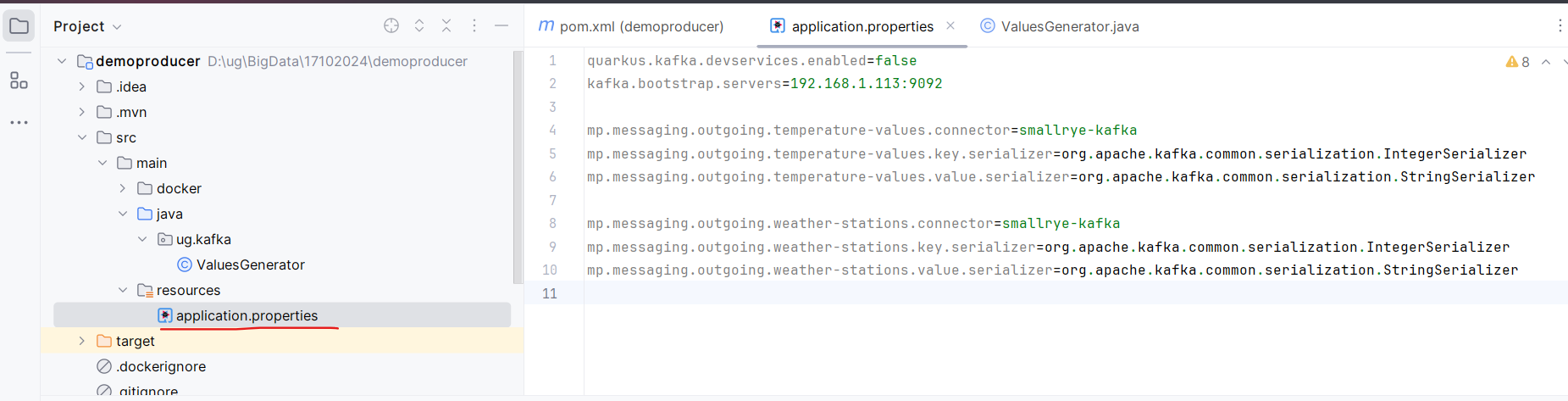
/opt/kafka/bin/kafka-topics.sh --create --topic **temperatures-aggregated** --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1



Create Quarkus maven project, and add dependencies:

<dependencies>  
 <dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-kafka-streams</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-messaging-kafka</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-arc</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>io.quarkus</groupId>  
 <artifactId>quarkus-junit5</artifactId>  
 <scope>test</scope>  
 </dependency>  
</dependencies>

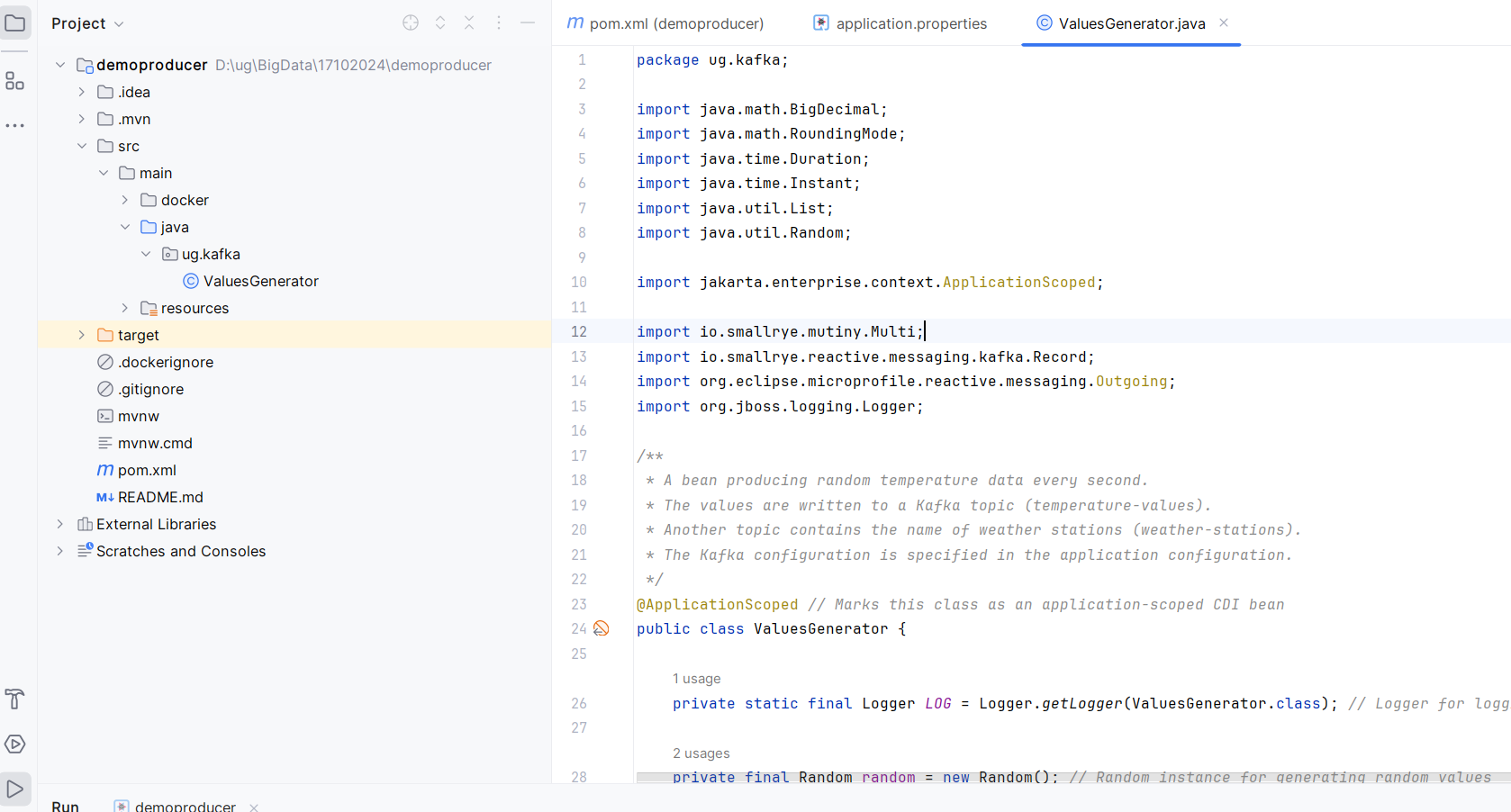
Add following content to application properties file:



quarkus.kafka.devservices.enabled=false  
kafka.bootstrap.servers=192.168.1.113:9092  
  
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

and create ValuesGenerator class, copy code from url:

<https://quarkus.io/guides/kafka-streams>



package ug.kafka;  
  
import java.math.BigDecimal;  
import java.math.RoundingMode;  
import java.time.Duration;  
import java.time.Instant;  
import java.util.List;  
import java.util.Random;  
  
import jakarta.enterprise.context.ApplicationScoped;  
  
import io.smallrye.mutiny.Multi;  
import io.smallrye.reactive.messaging.kafka.Record;  
import org.eclipse.microprofile.reactive.messaging.Outgoing;  
import org.jboss.logging.Logger;  
  
***/\*\*  
 \* A bean producing random temperature data every second.  
 \* The values are written to a Kafka topic (temperature-values).  
 \* Another topic contains the name of weather stations (weather-stations).  
 \* The Kafka configuration is specified in the application configuration.  
 \*/***@ApplicationScoped *// Marks this class as an application-scoped CDI bean*public class ValuesGenerator {  
  
 private static final Logger *LOG* = Logger.*getLogger*(ValuesGenerator.class); *// Logger for logging temperature data* private final Random random = new Random(); *// Random instance for generating random values* ***// List of predefined weather stations, each with an ID, name, and average temperature***private final List<WeatherStation> stations = List.*of*(  
 new WeatherStation(1, "Hamburg", 13),  
 new WeatherStation(2, "Snowdonia", 5),  
 new WeatherStation(3, "Boston", 11),  
 new WeatherStation(4, "Tokio", 16),  
 new WeatherStation(5, "Cusco", 12),  
 new WeatherStation(6, "Svalbard", -7),  
 new WeatherStation(7, "Porthsmouth", 11),  
 new WeatherStation(8, "Oslo", 7),  
 new WeatherStation(9, "Marrakesh", 20));  
  
 @Outgoing("temperature-values") *// Specifies the Kafka topic to which this method sends data* public Multi<Record<Integer, String>> generate() {  
 *// Creates a stream that emits a tick every 500 milliseconds* return Multi.*createFrom*().ticks().every(Duration.*ofMillis*(500))  
 .onOverflow().drop() *// Drops any overflow events if the consumer is too slow* .map(tick -> {  
 *// Selects a random weather station from the list  
 //შემთხვევითად არჩევს stations ლისტიდან ერთს და ინახავს station ცვლადში* WeatherStation station = stations.get(random.nextInt(stations.size()));  
  
 *// Generates a temperature based on the station's average temperature,  
 // adds some random noise, and rounds it to 1 decimal place* double temperature = BigDecimal.*valueOf*(random.nextGaussian() \* 15 + station.averageTemperature)  
 .setScale(1, RoundingMode.*HALF\_UP*)  
 .doubleValue();  
  
 *// Logs the station name and generated temperature  
 LOG*.infov("station: {0}, temperature: {1}", station.name, temperature);  
  
 *// Creates a Kafka record containing the station ID and a string with  
 // the current timestamp and temperature, then returns it* return Record.*of*(station.id, Instant.*now*() + ";" + temperature);  
 });  
 }  
  
 @Outgoing("weather-stations") *// Specifies the Kafka topic for sending station information* public Multi<Record<Integer, String>> weatherStations() {  
 *// Creates a stream of station records that contain their ID and name in JSON format* return Multi.*createFrom*().items(stations.stream()  
 .map(s -> Record.*of*(s.id, "{ \"id\" : " + s.id + ", \"name\" : \"" + s.name + "\" }"))  
 );  
 }  
  
 *// Inner class representing a weather station with an ID, name, and average temperature* private static class WeatherStation {  
  
 int id; *// Unique ID of the weather station* String name; *// Name of the weather station* int averageTemperature; *// Average temperature of the weather station  
  
 // Constructor to initialize the weather station* public WeatherStation(int id, String name, int averageTemperature) {  
 this.id = id;  
 this.name = name;  
 this.averageTemperature = averageTemperature;  
 }  
 }  
}

**Summary of How This Application is Launched and the Methods Executed**

1. **Launching the Application**:
   * This application is typically launched as a Quarkus or MicroProfile application. It requires the Quarkus runtime or a compatible Java application server that supports CDI (Contexts and Dependency Injection) and the MicroProfile Reactive Messaging specification.
   * When the application starts, the Quarkus runtime sets up all the necessary configurations, including connections to Kafka based on the settings defined in the application.properties or similar configuration files.
   * Kafka topics like temperature-values and weather-stations are set up for publishing messages, as specified in the @Outgoing annotations.
2. **Methods Executed**:
   * **generate()**:
     + This method is annotated with @Outgoing("temperature-values"), which means it produces messages for the temperature-values Kafka topic.
     + When the application starts, this method is triggered, creating a stream that emits data every 500 milliseconds.
     + For each tick, it randomly selects a weather station, generates a temperature value, logs the information, and sends the temperature data as a Kafka record.
   * **weatherStations()**:
     + This method is annotated with @Outgoing("weather-stations"), meaning it produces messages for the weather-stations Kafka topic.
     + It executes once when the application starts, creating a stream of station data that converts each weather station into a JSON-formatted record.
     + The station records are then published to the weather-stations topic.

In short, when the application is launched, it starts producing temperature values continuously to the temperature-values Kafka topic and publishes a one-time list of weather station metadata to the weather-stations Kafka topic.

Important:

return Record.of(station.id, Instant.now() + ";" + temperature);

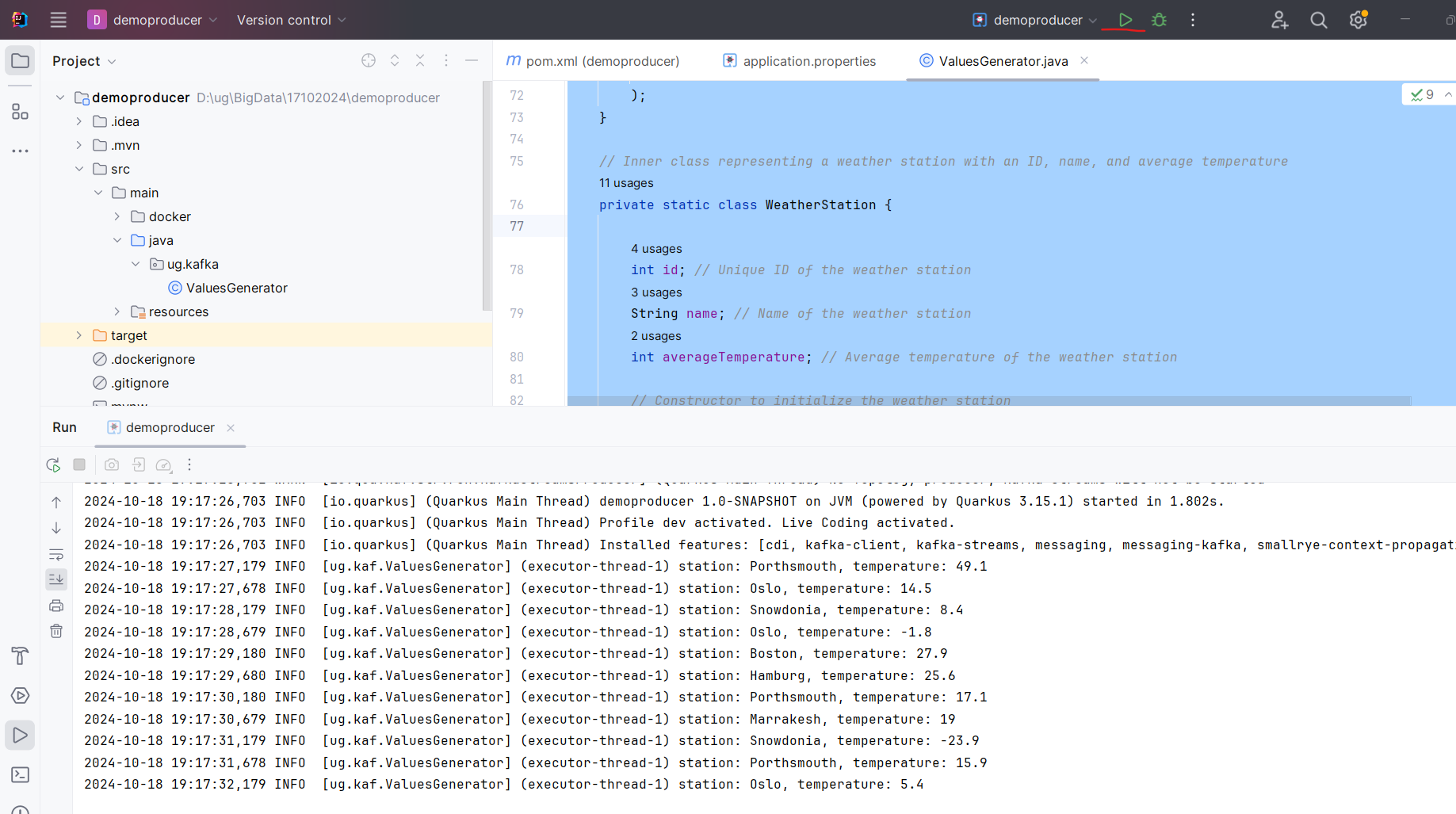
**Explanation of the Record.of Method**

* **Record.of(key, value)**: This method is used to create a Kafka record with a specified key and value.
  + The **key** in this case is station.id, which represents the stationId (an Integer).
  + The **value** is a String that combines the current timestamp and the generated temperature (e.g., "2024-10-17T02:40:35.156891Z;19.4").

**Explanation of the Record.of Method**

* **Record.of(key, value)**: This method is used to create a Kafka record with a specified key and value.
  + The **key** in this case is station.id, which represents the stationId (an Integer).
  + The **value** is a String that combines the current timestamp and the generated temperature (e.g., "2024-10-17T02:40:35.156891Z;19.4").

**After launching the application:**

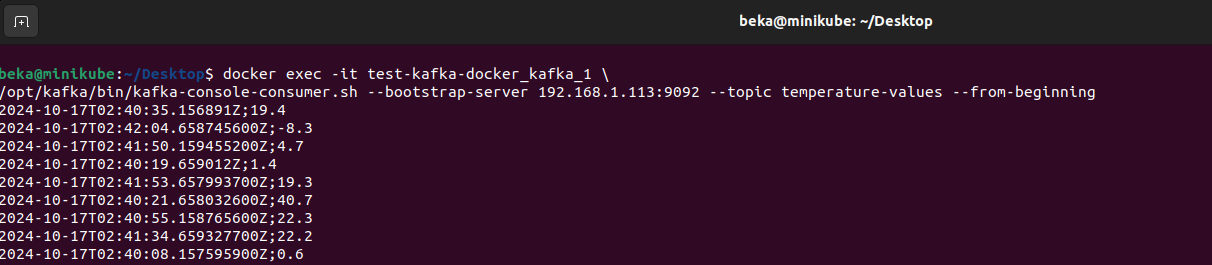
****

**Retrieve the Kafka Topic messaes**

To see the messages in the Kafka topic temperature-values, you can use the kafka-console-consumer.sh script, which allows you to consume messages from a topic. Since you are running Kafka in Docker and Kafka is on a remote machine, you should run a command like this from the Docker container where Kafka is running:

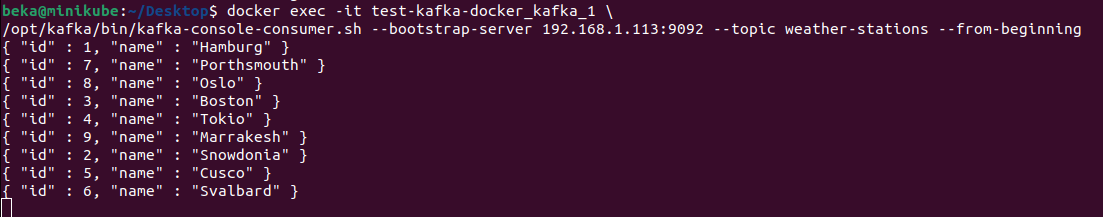
docker exec -it test-kafka-docker\_kafka\_1 \

/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server 192.168.1.113:9092 --topic temperature-values --from-beginning



docker exec -it test-kafka-docker\_kafka\_1 \

/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server 192.168.1.113:9092 --topic **weather-stations** --from-beginning



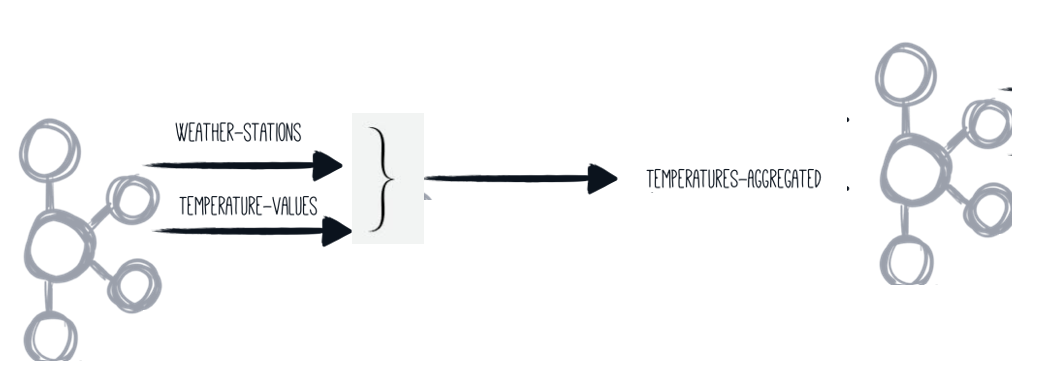
შევამოწმოთ რომ შემდეგ ტოპიკში ჯერ არაფერი არ მოდის:

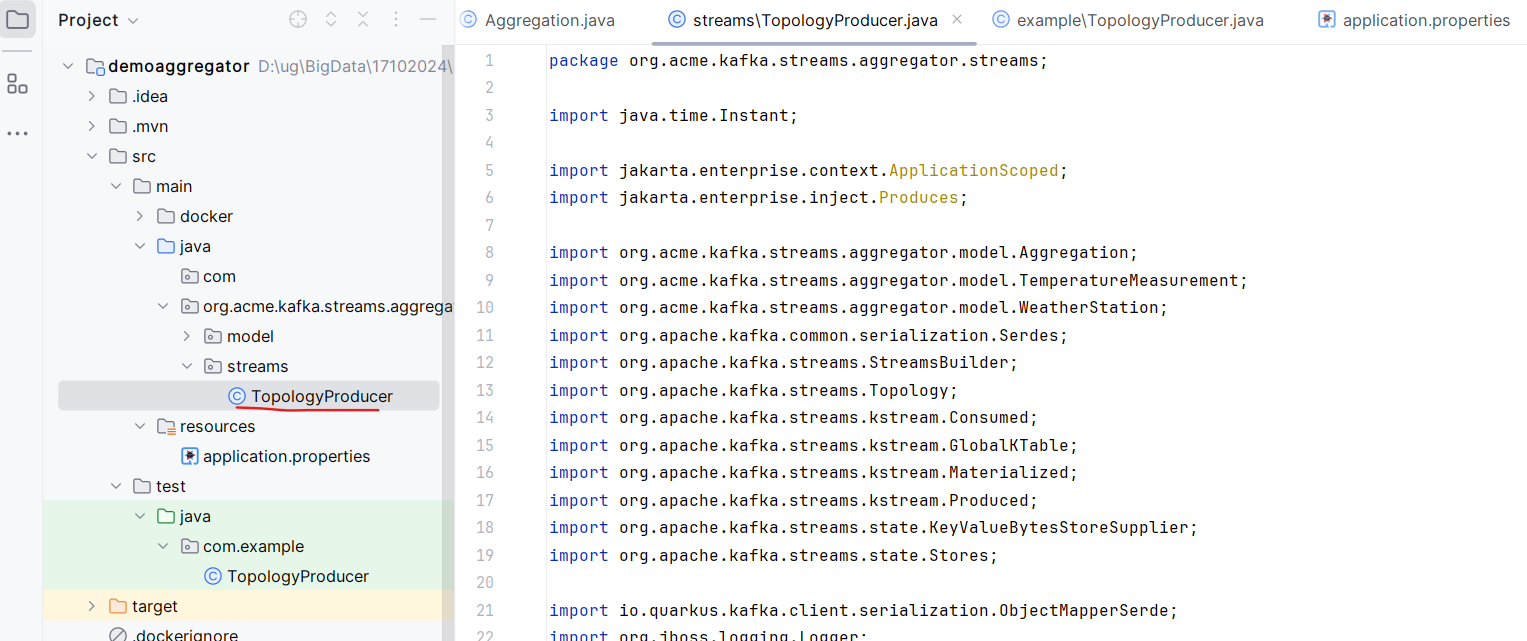
docker exec -it test-kafka-docker\_kafka\_1 \

/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server 192.168.1.113:9092 --topic **temperatures-aggregated** --from-beginning



**Now lets create the aggregator Project:**





TopologyProducer class, copy code from url:

<https://quarkus.io/guides/kafka-streams>

