Messaging with Kafka

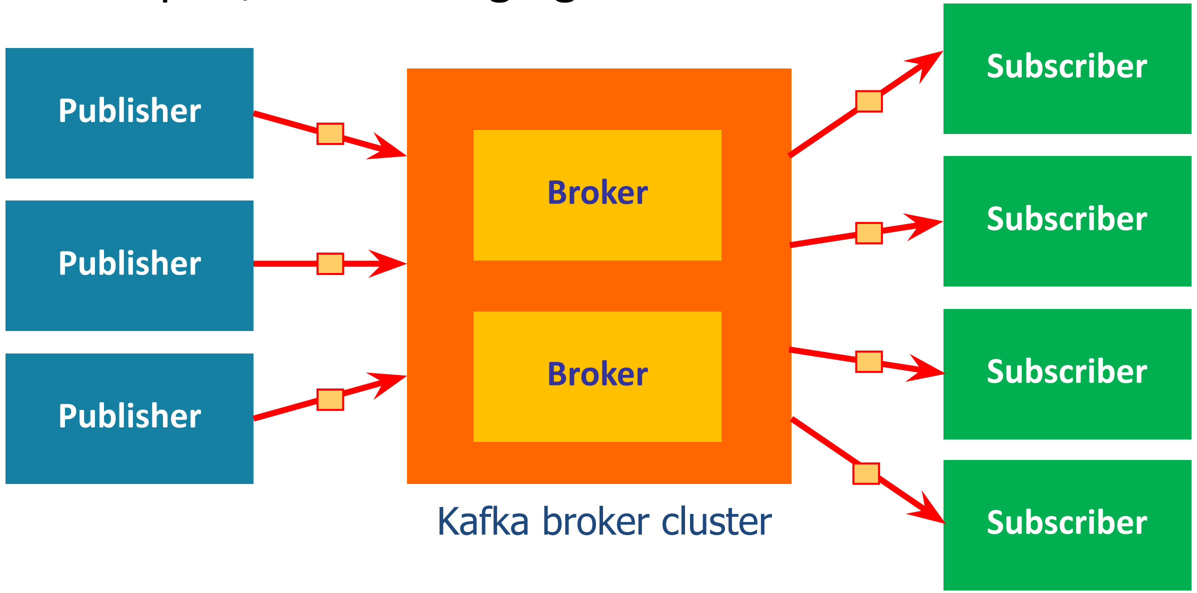
1. Overview of messaging using Kafka
2. Using Kafka in a Spring Boot application

1. Overview of Messaging using Kafka

- What is Kafka?
- Kafka in industry
- Installing Kafka
- A few words about Zookeeper
- Tweak the Kafka and Zookeeper config files
- Starting Zookeeper and Kafka

What is Kafka?

- Apache Kafka is a distributed message broker
  - Kafka runs as a cluster of broker nodes
  - Primary goal is high throughput, idea for cloud-scale architectures
  - Uses a pub/sub messaging architecture



Kafka in Industry

- Kafka is widely used by major players in industry
  - LinkedIn
  - Twitter
  - Netflix
  - Airbnb
  - Goldman Sachs
  - PayPal
  - Coursera
  - Hotels.com
  - Etc.

Installing Kafka

- In Windows…
  - Go to https://kafka.apache.org/downloads.html
  - Download and unzip the latest binary distribution, e.g. `kafka_2.12-2.8.1.tgz`

- In macOS:
  - Install Homebrew (if not already installed)
  - Then run the following command:

```
brew install kafka
```

A Few Words about Zookeeper

- Apache Kafka uses Apache Zookeeper to coordinate cluster info
  - Zookeeper is a distributed hierarchical key-value store
  - Provides a naming service for large distributed systems

- The Kafka download already includes Zookeeper
  - So you don't need to download Zookeeper separately
  - You just need to run Zookeeper first, then you can run Kafka…

Tweak the Kafka and Zookeeper Config Files

- You must edit the Kafka and Zookeeper configuration files
  - Set the data log directories appropriately for your environment

- Edit the Kafka configuration file, located here:
  - `<Kafka_Home>/config/server.properties`

```
log.dirs=C:/temp/kafka-logs                         server.properties
```

- Edit the Zookeeper configuration file, located here:
  - `<Kafka_Home>/config/zookeeper.properties`

```
dataDir=C:/temp/zookeeper                        zookeeper.properties
```

Pearson

Starting Zookeeper and Kafka on Windows

- To start Zookeeper on Windows:
  - Open a Command Prompt window
  - In the Kafka installation directory, run the following command:

```
bin\windows\zookeeper-server-start.bat config\zookeeper.properties
```

- To start Kafka on Windows:
  - Open another Command Prompt window
  - In the Kafka installation directory, run the following command:

```
bin\windows\kafka-server-start.bat config\server.properties
```

Starting Zookeeper and Kafka on macOS

- To start Zookeeper and Kafka on macOS:
  - Open a new Terminal window
  - In the Kafka installation directory, run the following commands:

```
zookeeper-server-start /usr/local/etc/kafka/zookeeper.properties &
kafka-server-start /usr/local/etc/kafka/server.properties
```

Pearson

2. Using Kafka in a Spring Boot Application

- Spring Boot dependency for Kafka
- Configuring application properties
- Sending messages to a topic
- Consuming messages from a topic
- Example REST API to publish messages
- Pinging the REST API

Demo package: `demo.kafka.simple`

Spring Boot Dependency for Kafka

- To use Kafka in a Spring Boot application, add the following dependency in your pom file:

```xml
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
```

pom.xml

Configuring Application Properties

- You can tell Kafka to create topics dynamically
  - This enables your application to send/receive from a topic, without you having to create the topic first

- To enable Kafka to create topics dynamically, define the following application property:

```
spring.kafka.listener.missing-topics-fatal=false
```

`application.properties`

Pearson

Sending Messages to a Topic

- To send messages to a topic:
  - Autowire a `KafkaTemplate<K,V>` bean
  - Call `send(topicName,key,value)`
  - Note that Kafka messages are key-value pairs ☺️

```java
@Service
public class MyPublisher {

    private static final String TOPIC_NAME = "mytopic";

    @Autowired
    private KafkaTemplate<String,String> kafkaTemplate;

    public void sendMessage(String key, String value){
        this.kafkaTemplate.send(TOPIC_NAME, key, value);
    }
}
```

MyPublisher.java

Pearson

Consuming Messages from a Topic (1 of 2)

- To consume messages from a topic, define a method and annotate it with:
  - `@KafkaListener(topics,groupId)`

- You can define multiple listeners for the same topic
  - One listener in each group will receive the message

- Listener methods receive the message value
  - Can also receive message header metadata, e.g. key, timestamp

Consuming Messages from a Topic (2 of 2)

- Example consumer in our demo app:

```
@Service
public class MyConsumer {

    @KafkaListener(topics = "mytopic", groupId="group1")
    public void group1ConsumerA(
            @Header(KafkaHeaders.RECEIVED_MESSAGE_KEY) String key,
            @Header(KafkaHeaders.RECEIVED_TOPIC) String topic,
            @Header(KafkaHeaders.RECEIVED_TIMESTAMP) String timestamp,
            String value) {

        // Process the message here…
    }
}
```

MyConsumer.java

Example REST API to Publish Messages

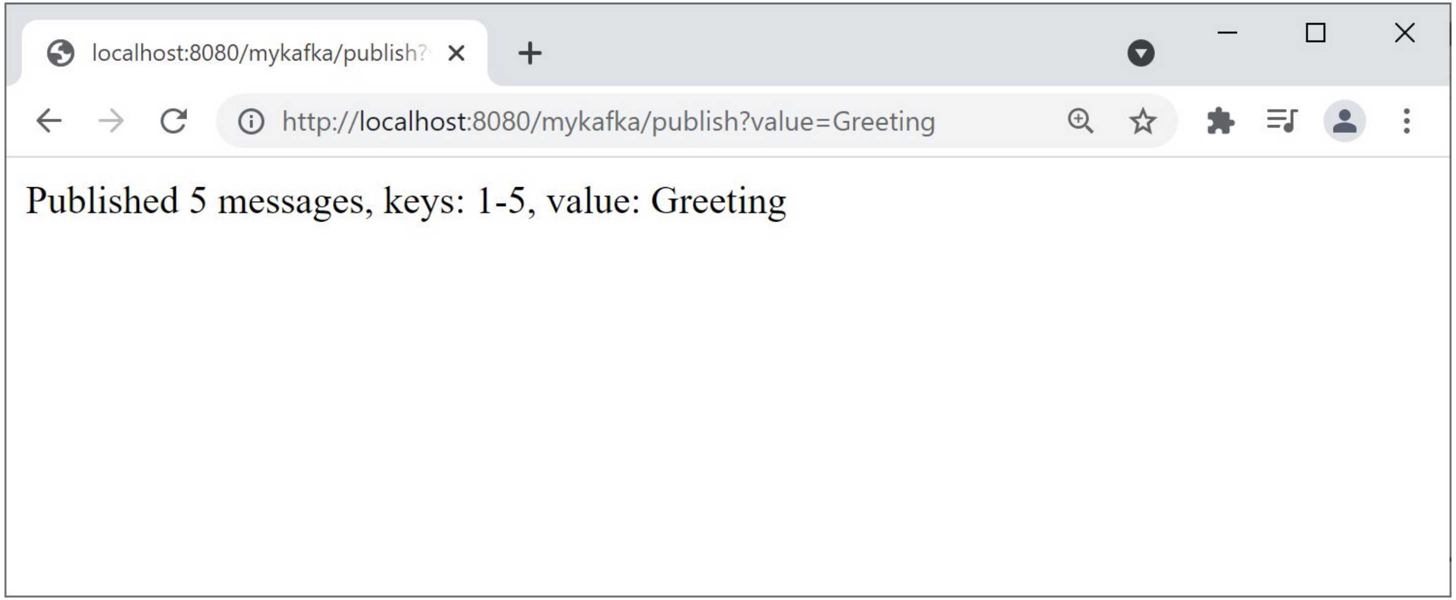
- The demo app has a REST controller
  - Allows users to publish messages to a topic

```java
@RestController
@RequestMapping("/mykafka")
@CrossOrigin
public class MyRestController {

    @Autowired
    private MyPublisher publisher;

    @GetMapping("/publish")
    public String publish(@RequestParam("value") String value) {
        for (int i = 1; i <= 5; i++) {
            this.publisher.sendMessage("key" + i, value);
        }
        return "Published 5 messages, keys: 1-5, value: " + value;
    }
}
```

MyRestController.java

Pinging the REST API

- You can ping the REST API as follows:
  - http://localhost:8080/mykafka/publish?value=Greeting


localhost:8080/mykafka/publish?
http://localhost:8080/mykafka/publish?value=Greeting

```
Published 5 messages, keys: 1-5, value: Greeting
```

Summary

- Overview of Kafka
- Using Kafka in a Spring Boot application

Exercise

- The simple example we've just shown sends messages with the following data types:
  - Key: `String`
  - Value: `String`

- Kafka lets you send any data types as keys/values
  - You must configure `KafkaTemplate`
  - Specify how to serialize/deserialize objects
  - See: `demo.kafka.customobjects`

Demo package: demo.kafka.customobjects