Day 2 – Lab2:

Implement Topics and Partitions

1. Patient monitoring

Introduction

In the design exercise we asked you to define topics and partitions for a remote patient monitoring system. Every one of you may have created different topics and partitions for this exercise.

We'll just suggest one solution and have you implement a small portion of the problem.

The Concerns

This system has been built by developers for a big US hospital some years back. Although we will not have access to the code, below are highlights of some of the concerns that needed to be designed for.

Here are some of these concerns:

- The site we designed was multi-tenant (several hospitals using the same system). We had to make sure that the data from one patient is sent to the correct hospital/doctor.
- The data had very different shape and frequency based on the kind of device (e.g., a weight scale is quite different from an EKG)
- Nurses want to see data from a subset of patients

Making it simple

We have limited time for this course, but we really want to give you a feeling for how you would take advantage of a tool like Kafka to implement a solution for this.

Let's make it simple and assume that we work on behalf of a single hospital. Let's also further simplify by assuming that we only keep track of a couple of types of devices.

Let's say we track:

- Weight scales

- Blood pressure monitor

What is interesting here is perhaps that while we may step on the weight scale once per day, we may have a blood pressure monitor that produce events once every 10 seconds.

Let's also give you some other metrics:

- We expect to monitor up to 100,000 patients
- We intend to build a UI for nurses where each nurse monitors about 1,000 patients each
    - From this follows that at any time we would have about 100 nurses monitoring the patents

Let's also assume some very simple rules that decide upon actions:

- When the blood pressure moves from normal to high, we want the nurse assigned to be alerted. We assume that a move to values above 150/100 should trigger the event for the nurse.

Let's also assume that each of the devices sends out a ping once per hour to say that they are healthy. We would also want to ensure that we monitor the devices and alert the service technicians if one of the devices does not call home.

Suggested topics

Here are some suggested topics:

- Device installation
    - We will create one topic to manage device installations
    - Messages sent on this topic:
        - Device installed
        - Device uninstalled
- Patient observation
    - We will use this topic to send messages from the scales and the blood pressure monitors. We can expect that we will see many more kinds of messages, but we want to use a single topic with polymorphic messages. The main reason for this is to ensure that we get the messages for a single patient no matter which device sent the message.
    - Message sent on this topic

        - Device event
            - Multiple shapes, but a common header
            - Header
                - Device id
                - Observation time
                - Message type
            - Body
                - Weight measurement
                    - Weight in kilograms
                - Blood pressure
                    - Pulse
                    - Systolic pressure
                    - Diastolic pressure
- Device heartbeat
    - This message would be a simple message that simply say "I'm online and working".

What about partitions?

Since we're running a single node cluster in this exercise, we will not bother to implement the partitions. BTW, if you do want to play with a larger cluster, the docker image for Kafka supports that. You can simply tell `docker-compose` to scale out the Kafka cluster to more nodes (for this, run `docker-compose scale kafka=X`, where `X` is the number of nodes you want to run). Partitions would be used primarily for controlling throughput.

We can imagine that the patient observations may require significant throughput. Let's say we start introducing devices that produce hundreds of messages per second.

Ideally, we should try to process all events from a patient in order, however, that can turn out to be tricky in this case as each device may be streaming out events independently. Perhaps the best we can do is to ensure that the events from a single device are processed in order.

A typical design then would be:

- Measure the number of messages that each Kafka node can process per second (e.g., say we use 10,000)
- Measure the number of messages we expect on the topic per second (e.g., say 100,000)
- Divide the numbers for expected messages and processing ability and we get at least 10 partitions. We probably want to have more than that, so perhaps we settle on 50 partitions.
- Since we want to make sure we process a single device stream in order, let's use the device id as the source for the partition key.

Serialization / Deserialization

The serialization and de-serialization would depend somewhat on what we can control (the devices may have their own protocol and in the world of IoT, we often must support many protocols. Common protocols are ASN.1, MQTT, JSON, XML, etc.).

Let's assume for our simple example that you can use JSON.

Processor

We probably want at least two processors for this scenario

Device monitor

This process is simple. It will simply read the heartbeats for each device.

The behavior would be something like this:

- If it has not heard from the device in 2 hours, it would produce a device-warning message.
- If it has not heard from the device in 3 hours, it would produce a device-failure message.
- If it hears from a device that is in any state, it produces a device-back-online message.

Patient monitor

This processor listens to the device events and builds up a current picture of the patient vitals.

The processor would have to be supported by a database where we map the devices to individual patients.

Whenever the patient vitals change, it pushes an event to a new patient topic.

Create the topics

Create the following topics:

- device-heartbeat
- device-events
- patient-vitals
- patient-alarm
- device-status

Create the processors

Heartbeat monitor

Create a processor that listens to the heartbeats and if you don't hear from a device within a specified period, then push an event to the device-status topic.

Patient monitor

Create a processor that listens to the device events and if you see a message where the blood pressure is above a given limit, you push an event to the patient-alarm topic.

2. Topics and partitions

Introduction

This is a dramatically simplified version of this exercise.

We have left an exercise description for a more ambitious exercise in this directory if you want a harder challenge after this course is finished.

We'll simply implement a heartbeat example.

Imagine that the devices used by the remote patients sends a heartbeat at least every minute to say they are alive and well.

We want to implement a processor that listens to these heartbeats and when we have not heard from the device for one minute, we want to sound an alarm.

Start docker containers

Make sure you shut down the docker images from the last exercises, then change into this lab's directory and start docker:

```
$ docker-compose up
```

Create the topics

In a new terminal, change directory to where we have our docker-compose file (`04-Implement-Topics-And-Partitions`) and run the following commands:

```
$ docker-compose exec kafka kafka-topics.sh --bootstrap-server :9092 --create --replication-factor 1 --partitions 1 --topic device-event
$ docker-compose exec kafka kafka-topics.sh --bootstrap-server :9092 --create --replication-factor 1 --partitions 1 --topic device-heartbeat
```

Build and run the device simulator

We've already created a device simulator app, but we must build it first.

In a terminal, change to the lab's `device-simulator` directory and run the following command:

```
$ docker run -it --rm -v "$(cd "$PWD/../.."; pwd)":/course-root -w "/course-
root/$(basename $(cd "$PWD/.."; pwd))/$(basename "$PWD")" -v
"$HOME/.m2/repository":/root/.m2/repository maven:3-jdk-11 ./mvnw clean package
```

On a windows machine, you must replace the `$PWD` with the current directory and the `$HOME` with a directory where you have the `.m2` folder.

Start the device-simulator

Next, let's start the simulator of device messages.

```
$ docker run --network "$(cd .. && basename "$(pwd)" | tr '[:upper:]'
'[:lower:]')_default" --rm -it -v "$PWD:/pwd" -w /pwd openjdk:11 java -jar
target/device-simulator-app-*.jar
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
...
```

On a windows machine, you must replace the `$PWD` with the current directory and the `$HOME` with a directory where you have the `.m2` folder.
We are now producing simulated device events.

Create a simple console consumer

To see the messages, let's run our usual console consumer. In a new terminal `cd` into the `04-Implement-Topics-And-Partitions` directory and then run the Kafka consumer.

```
$ docker-compose exec kafka kafka-console-consumer.sh --bootstrap-server kafka:9092 -
-topic device-heartbeat
```

After a few seconds you should start to see heartbeat messages being produced:

```
...
Scale 1 sent heartbeat at Sat Jun 17 15:43:06 CDT 2017
Scale 2 sent heartbeat at Sat Jun 17 15:43:10 CDT 2017
Heart monitor 2 sent heartbeat at Sat Jun 17 15:43:11 CDT 2017
Heart monitor 5 sent heartbeat at Sat Jun 17 15:43:13 CDT 2017
Scale 5 sent heartbeat at Sat Jun 17 15:43:13 CDT 2017
Scale 6 sent heartbeat at Sat Jun 17 15:43:19 CDT 2017
Scale 1 sent heartbeat at Sat Jun 17 15:43:21 CDT 2017
Heart monitor 6 sent heartbeat at Sat Jun 17 15:43:22 CDT 2017
Heart monitor 1 sent heartbeat at Sat Jun 17 15:43:24 CDT 2017
Scale 4 sent heartbeat at Sat Jun 17 15:43:24 CDT 2017
Heart monitor 2 sent heartbeat at Sat Jun 17 15:43:26 CDT 2017
Heart monitor 7 sent heartbeat at Sat Jun 17 15:43:27 CDT 2017
Heart monitor 5 sent heartbeat at Sat Jun 17 15:43:28 CDT 2017
Scale 5 sent heartbeat at Sat Jun 17 15:43:28 CDT 2017
Scale 6 sent heartbeat at Sat Jun 17 15:43:34 CDT 2017
...
```

Create the device monitor

We've also written a monitor for you. You can find this monitor in the directory `device-monitor/`. Take some time to study the code and see how it works.
First, build the application:

```
$ docker run -it --rm -v "$(cd "$PWD/../.."; pwd)":/course-root -w "/course-
root/$(basename $(cd "$PWD/.."; pwd))/$(basename "$PWD")" -v
"$HOME/.m2/repository":/root/.m2/repository maven:3-jdk-11 ./mvnw clean package
```

On a windows machine, you must replace the `$PWD` with the current directory and the `$HOME` with a directory where you have the `.m2` folder. Next, run it:

```
$ docker run --network "$(cd .. && basename "$(pwd)" | tr '[:upper:]'
'[:lower:]')_default" --rm -it -v "$PWD:/pwd" -w /pwd openjdk:11 java -jar
target/device-monitor-*.jar
```

On a windows machine, you have to replace the `$PWD` with the current directory and the `$HOME` with a directory where you have the `.m2` folder.
You should now see a set of output like this:

```
...
2022-02-07T15:47:53.049305Z
2022-02-07T15:47:56.444218Z
2022-02-07T15:47:58.672334Z
2022-02-07T15:48:00.970748Z
2022-02-07T15:48:01.37685Z
2022-02-07T15:48:01.400834Z
...
```

Create a console consumer online/offline messages

```
$ docker-compose exec kafka kafka-console-consumer.sh --bootstrap-server kafka:9092 -
-topic device-event
```

It may take some time before you see online or offline messages (see the device simulator and you'll see the randomness of the heartbeat production).

You should eventually start to see devices go offline and online.

For example:

```
...
Scale 7 offline since Sat Jun 17 16:58:10 CDT 2017
Scale 7 is back online
Heart monitor 2 offline since Sat Jun 17 16:59:41 CDT 2017
Heart monitor 2 is back online
...
```

Congratulations. You finished the lab!