Design of Topics and Partitions

Topic Design

- Name
- Schema
- Payload (Data)
- Key
- Number of partitions
- Number of replicas

The Short Story

- DevOps concerns
  - Bandwidth consumption → Size of messages, `serdes`, etc.
  - Fault tolerance and availability → Size of cluster, replication factor, etc.
  - Performance → Partitions, message size, cost of serialization, etc.
- Producer concerns
  - Ease of production → Clear schema, cost of serialization, delivery guarantees, etc.
- Consumer concerns
  - Can I subscribe to only what I need → Topics and partitions
  - Latency → Cluster size, performance, etc.

How Topics and Partitions Influence Concerns?

- Topic topology
  - Schema (structure, format, etc.)
  - Temporal constraints (frequency, triggers, etc.)
  - Do you use topics to allow for fine grain subscriptions?
- Partitions
  - Determines throughput (but not without cost)
  - Can be used for fine-grained subscription (requires use of low level API)
- Recommendation
  - Use topics to convey semantics
  - Use partition to control throughput

Name

- Descriptive name
- Don't hardcode the name all over your application!
- Rule of thumb:
  - Use a longer name that is easy to understand

Schema

- JSON
  - Common choice
  - Not the most efficient

- Apache Avro
  - Binary format
  - Compression
  - Schema evolution
  - Dynamic typing (no code generation needed for serialization)

Partitions and Throughput

- Unit of parallelism: topic partition
- Writes to different partitions done in parallel
- Consumer: one thread get a single partition's data
- Consumer parallelism: bounded by the number of consumed partitions
- Throughput on a producer is a function of:
  - Batching size
  - Compression codec
  - Acknowledgement type
  - Replication factor
- Consumer throughput is a function of the message processing logic

Overpartitioning

- Problem: Increasing the number of partition and message ordering
  - If messages have keys, increasing the number of partitions may cause problems
  - Kafka maps a message to a partition based on the hash of the key
  - Messages with the same key go to the same partition
  - If we increase the number of partitions, this does not hold
    - Messages with the same key may for the retention period appear in multiple partitions
- Therefore:
  - Overpartition for a situation you expect in future

Too Many Partitions

- Each partition maps to a directory in the broker
  - 2 files: index, actual data
- You may need to configure the open file handle limit
  - Configuration
  - Seen in production > 30K open file handles / broker

Partitions and Availability

- Intra-cluster replication
- A partition can have multiple replicas, each on a different broker
- Clean broker shutdown: the controller moves the leaders off the broker that is shutting down
  - Takes a couple of ms
- Unclean shutdown: the loss of availability is dependent on the number of partitions
  - All replicas become unavailable at the same time
  - A new leader must be elected
  - Potential unavailability in seconds

Partitions and End-to-End Latency

- Consumers can consume a message after it has been committed
  - Requires replication to all in-sync replicas
- Commit time can be significant
  - Too long for some real-time systems

- A rule of thumb:
  - Limit the number of partitions per broker to:
    100 x (number of brokers) x (replication factor)

Based on Confluent Blog

Partitions and Client Memory

- A producer can set the amount of memory for buffering messages
  - Messages are buffered per partition
  - When buffer is full, messages are sent to the broker
- More partitions: more message buffering in the producer
- If out of memory, producer will block or drop new messages
  - Reconfigure
- Allocate at least a few tens of KB per partition

Summary

- Design topics based on your application semantics
  - Message types
  - Consumer concerns
  - Subscription granularity
- Decide on the number of partitions based on throughput
  - The more partitions, the higher theoretical throughput
  - Not without penalty
    - File handlers, consumer memory, latency, etc
  - Evaluate to overpartition to accommodate future growth

Lab: Design

- Let's assume you'll have to collect information from devices installed to keep track of the vitals of a set of patients
  - The patients are being treated from their home
  - You may assume that the patients produce 10 messages per second on average
  - You may have up to 1 million patients being tracked at the same time
- Multiple stakeholders
  - Nurse: Keeps track of a number of patients
  - Service technician: Need to know when a device goes down
  - Billing: Keeps track of events that are billable
  - ...

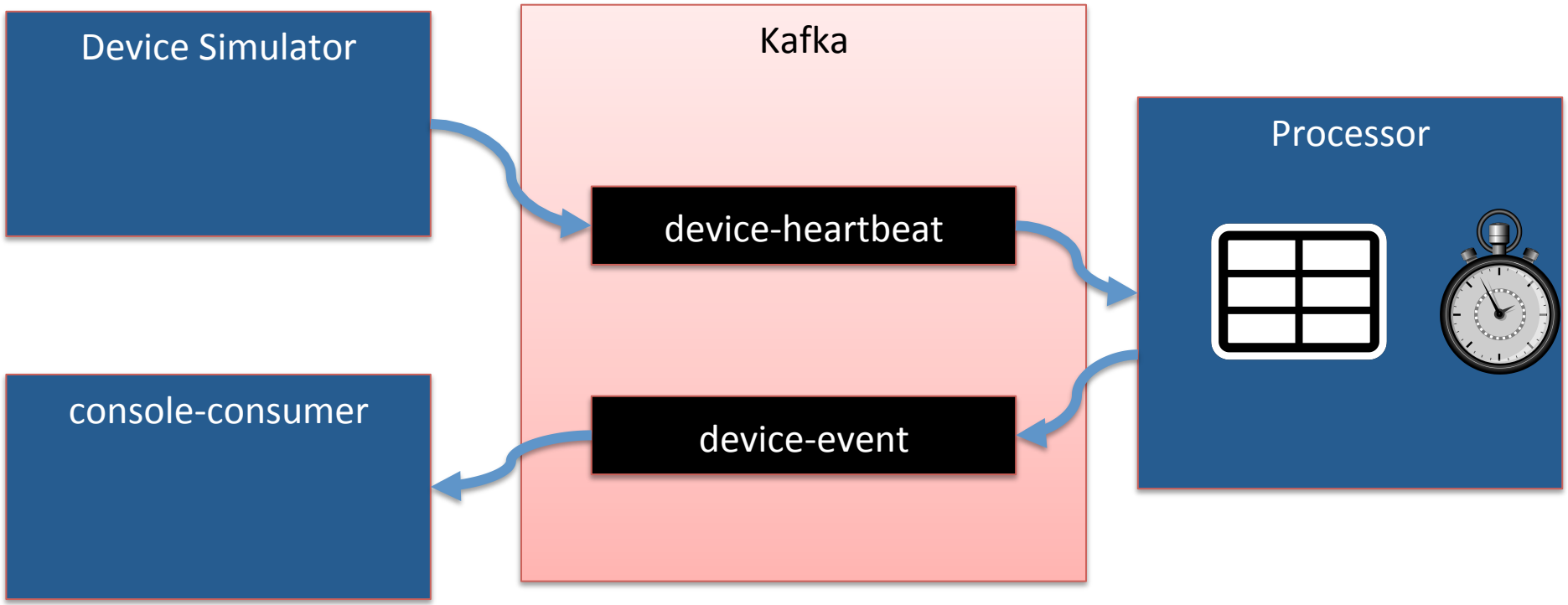
Lab: Design... The Question

- What would be the topics for such a system?
- How do we decide how many partitions per topic?

Lab: Implementation

- We'll take a look at a small sleeve of the problem
- Service technician's view:
  - Assuming all devices send a heartbeat
  - We want to know if the devices go offline and when they come back online
- We'll implement a processor that listens to the incoming heartbeats and decides if the device is online or offline

What We'll Run



Step 1: The Docker Setup

- I'm not going to repeat it here, but you'll have to get the docker-compose file to run
  - Find your IP
    - `ipconfig | grep inet`
  - Edit the `docker-compose.yml` file to update the IP
  - `docker-compose up`

Step 2: Create the Topics

- This step we're also been through before. We'll create two topics:
  - **device-heartbeat**
  - **device-event**
- Make sure you're in the docker directory

```
sh
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-topics.sh --create --zookeeper zookeep
er:2181 --replication-factor 1 --partitions 1 --topic device-heartbeat
Created topic "device-heartbeat".
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-topics.sh --create --zookeeper zookeep
er:2181 --replication-factor 1 --partitions 1 --topic device-event
Created topic "device-event".
```

Step 3: Build and run the Device Simulator

- Provided is a device simulator
  - Produces the heartbeat
  - Randomly will miss the 1 minute deadline to make the processor decide that the device is offline

```
sh
LM-SJN-21001415:04-Implement-Topics-And-Partitions pgraff$ cd heartbeat-simulator/
LM-SJN-21001415:heartbeat-simulator pgraff$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building device-monitor 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ device-monito
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.5.1:compile (default-compile) @ device-monitor -
```

The Critical Code of DeviceSim

```
14  public DeviceSim(final String deviceId, final KafkaProducer<String, String> producer) {
15    timer.scheduleAtFixedRate(new TimerTask() {
16
17      @Override
18      public void run() {
19        if (r.nextBoolean()) {
20          System.out.println("Produced heartbeat for device " + deviceId);
21          producer.send(
22              new ProducerRecord<String, String> (
23                  "device-heartbeat",
24                  deviceId,
25                  deviceId + " sent heartbeat at " + new Date().toString()
26                  ));
27        }
28      }
29    }, r.nextInt(10000)+10000, 15*1000);
30  }
```

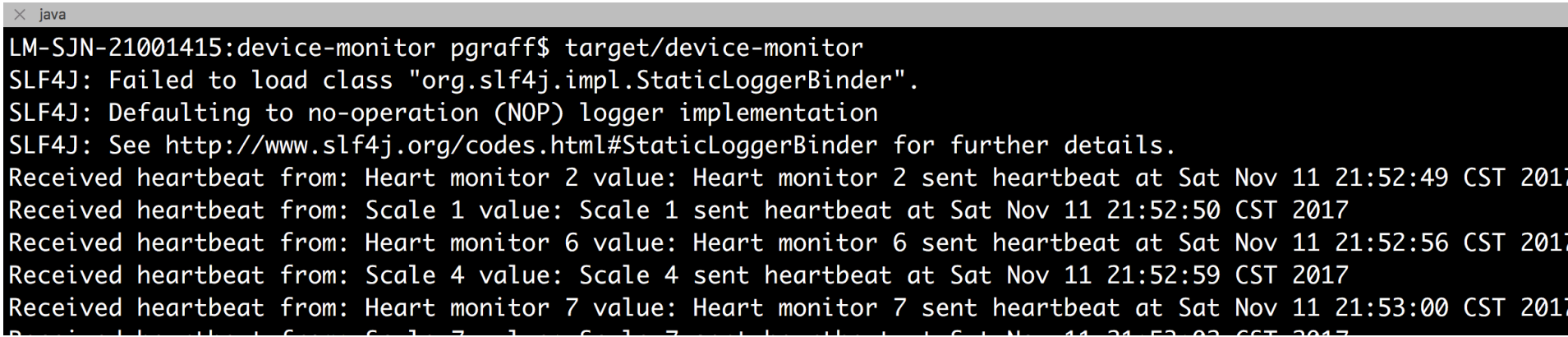
Step 5: Run the DeviceSim

- `cd kafka-lab/labs/04-Implement-Topics-And-Partitions/device-sim`
- `mvn package`
- `target/simulator`

```
java
LM-SJN-21001415:heartbeat-simulator pgraff$ target/simulator
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.
Press enter to quit
Produced heartbeat for device Scale 4
Produced heartbeat for device Heart monitor 1
Produced heartbeat for device Scale 7
Produced heartbeat for device Heart monitor 3
Produced heartbeat for device Heart monitor 4
Produced heartbeat for device Scale 4
Produced heartbeat for device Heart monitor 7
```

Step 6: Run the DeviceSim

- cd kafka-lab/labs/04-Implement-Topics-And-Partitions/device-monitor
- mvn package
- target/simulator

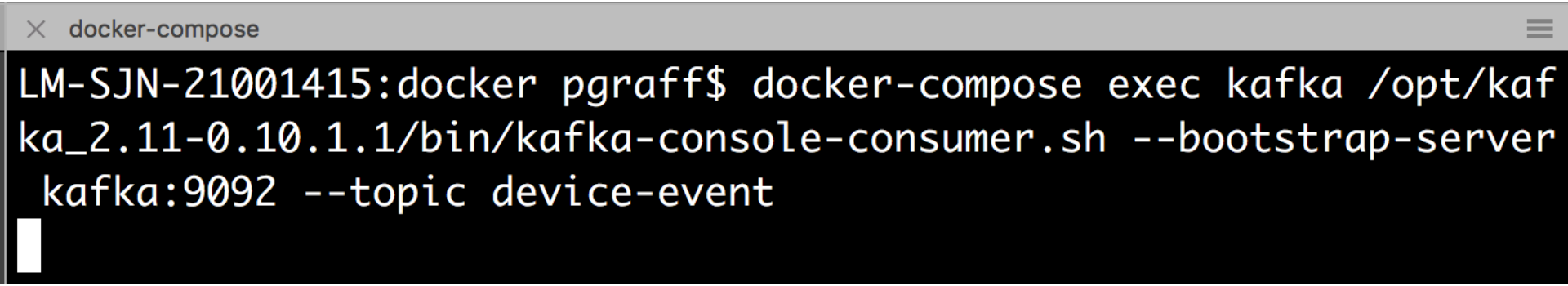


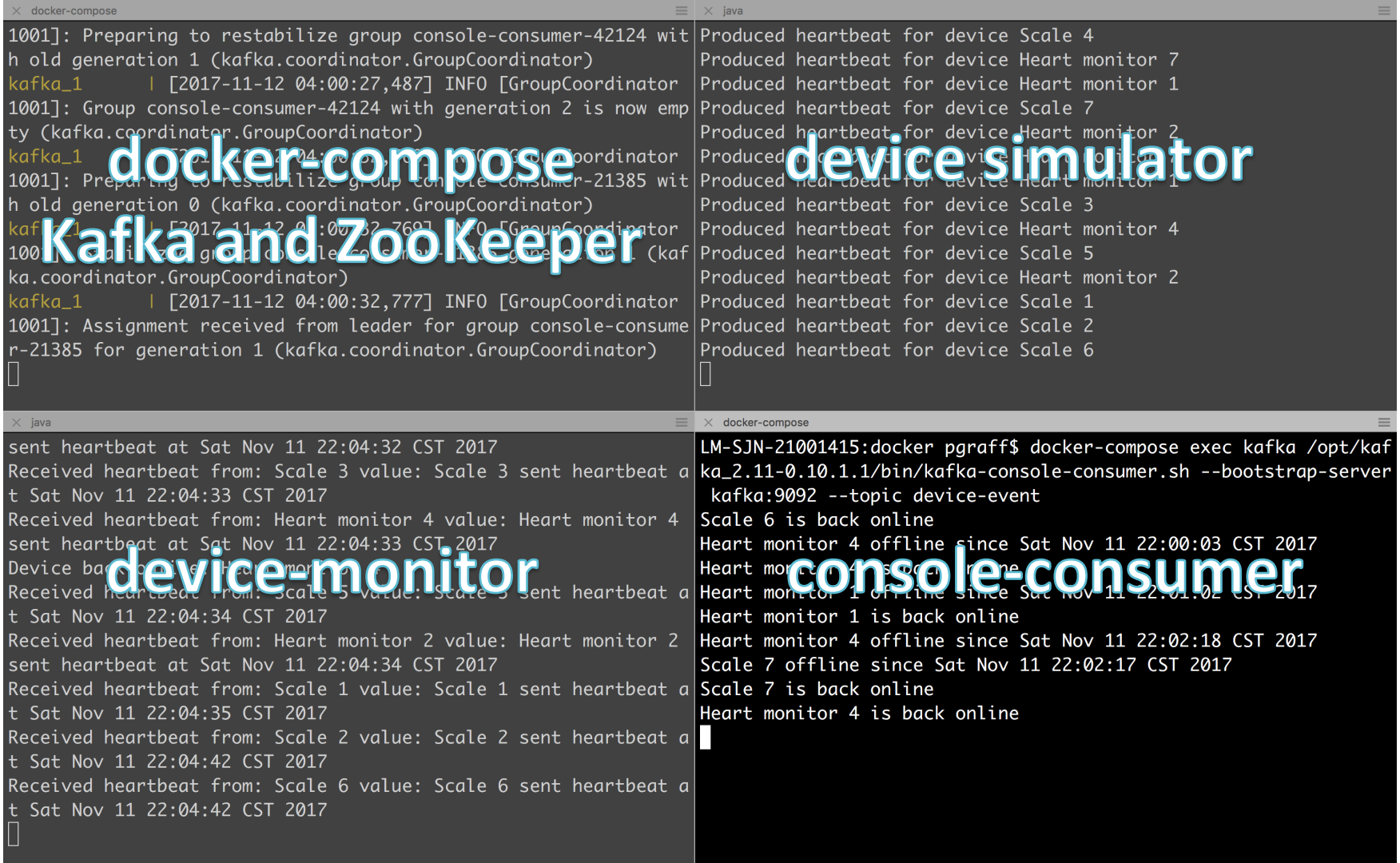
Interesting Code inside the Device Monitor

```
32      while (true) {
33        final ConsumerRecords<String, String> records = consumer.poll(1000);
34        for (ConsumerRecord<String, String> record : records) {
35          final String key = record.key();
36          lastSeenMap.put(record.key(), new Date());
37          System.out.println("Received heartbeat from: " + key + " value: " + record.value());
38          if (offlineDevices.contains(key)) {
39            offlineDevices.remove(key);
40            System.out.println("Device back online: " + key);
41            producer.send(createOnlineMessage(key));
42          }
43
44        }
45      }
```

Step 7: Run a Console Consumer

- Let's make sure we can see the output from the device monitor
- Make sure you're in the docker directory
- Run the console-consumer
  - **docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic device-event**




docker-compose
Kafka and ZooKeeper
device simulator
device-monitor
console-consumer

Step 8: Close it all down

- `docker-compose down`