

Kafka Fundamentals

Your Instructors: Petter Graff

- Serial entrepreneur, architect, consultant, teacher, and sometime CXO
- CIO of Northscaler, Praetexo and the CEO of SciSpike
- Architect of [Yaktor](#), a scalable event-driven, agent based rapid development platform
- Teaches classes on:
 - BigData (Hadoop, Spark, HBase, Cassandra, etc.)
 - Computer Science Concepts (Scalability, Architectural Thinking, Design Patterns, OOA&D, etc.)
 - Languages (Java, Scala, C++, JavaScript, ...)
 - And much more...
- Lives in Austin
- O'Reilly author:
[Design Patterns in Java](#)



Northscaler

- Accelerator for scale-ups, including:
 - Definition of architecture
 - Custom software development
 - Strategic advice to the C-suite
- Consultant, and mentor to many large organizations worldwide
- Big Data and Machine Learning
- Custom training to leading firms worldwide



- <http://www.northscaler.com>
- The independent technology advisors

Outline

- Introduction to Kafka
- Kafka core concepts
- Producing data to Kafka using the Producer API
- Consuming data from Kafka using the Consumer API
- Kafka Streaming
- Kafka Administration and Integration
- Exactly once delivery, what does that mean?

What is Kafka?

- "*Kafka is a distributed, partitioned, replicated commit log service*"
 - Fault tolerant
 - Near linearly scalable (horizontally)
 - Durable
- Often used as a publish-subscribe messaging system
- Apache project
- Originally developed by LinkedIn



kafka

Design Goals at LinkedIn

- Decouple the producers and consumers by using a push-pull model
- Persistent storage with possibility of replay
- Optimize for high throughput of information
- Allow for horizontal scaling as the data streams grow

Kafka is Born!

- Publish/subscribe system with an interface where
 - Publishing is similar to a typical messaging system
 - Subscription diverges as it works on batches
 - A storage layer like a log aggregation system
- As of August 2015, LinkedIn produces over one trillion messages and a petabyte of data consumed daily!
- Kafka was released as open source in late 2010 and became an Apache project in 2011

The Name

I thought that since Kafka was a system optimized for writing using a writer's name would make sense. I had taken a lot of lit classes in college and liked Franz Kafka. Plus the name sounded cool for an open source project.

So basically there is not much of a relationship.

Jay Kreps, Lead Developer at LinkedIn

Notable Users

- Cisco Systems
- Netflix
- PayPal
- Spotify
- Uber
- HubSpot
- Betfair
- Shopify

Kafka API's

Sources
ElasticSearch, JDBC,
MongoDB, Cassandra

Kafka Connect

Sink
ElasticSearch, JDBC,
MongoDB, Cassandra

Publisher

Publisher
API

Kafka Core

Consumer
API

Consumer

Streaming API

Use Cases

- Messaging
- Website Activity Tracking
- Metrics
- Log Aggregation
- Stream Processing
- Event Sourcing
- Commit Log

Website Activity Tracking

- The original use case for Kafka was to be able to rebuild user activity as a set of real-time publish-subscribe feeds
- Site activity such as page views, searches, or other user actions are published to central topics with one topic per activity type
- These feeds are then available for subscription for processing, monitoring, and loading into offline processing/reporting
- Activity tracking can be **very high volume** as many messages are generated for each user page view

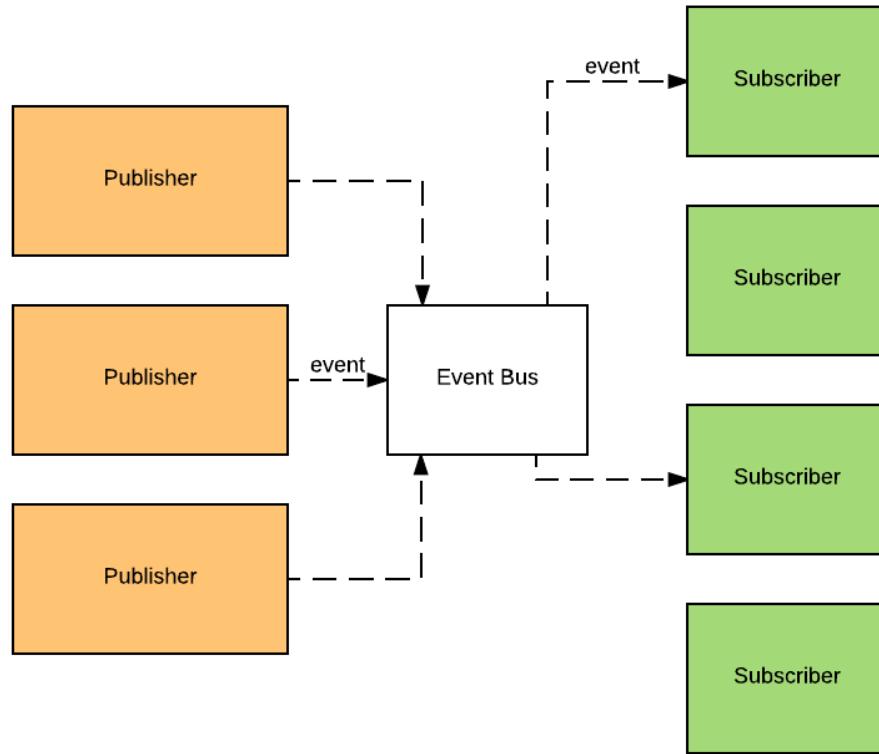
Messaging

- Kafka can be a replacement for a more traditional message broker such as ActiveMQ or RabbitMQ
 - But only for publish-subscribe type use cases
- Message broker scan be used to decouple processing from data producers, buffer unprocessed messages, etc.
- Kafka has replication, built-in partitioning, and fault-tolerance making it a good solution for large scale applications

Publish-Subscribe Pattern

- Sender (publisher) wants to send a piece of data (message)
- The message is not specifically directed to a receiver (subscriber)
- The sender classifies the message and the receiver subscribes to receive certain classes of messages
- Usually there is a central broker where messages are published

Publish-Subscribe Illustrated



Log Aggregation

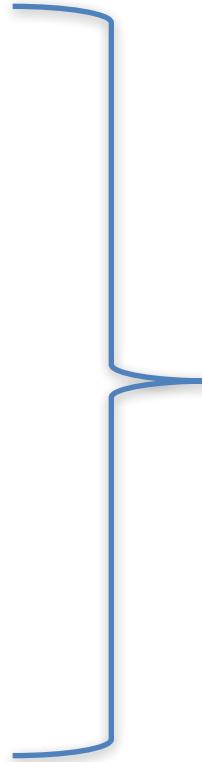
- Log aggregation collects physical log files off servers and put them in a central place such as a file server or HDFS for processing
- Kafka abstracts away the details of files and gives a cleaner abstraction of log or event data as a stream of messages

Why Kafka?

- Able to connect a large number of clients (Producers and Consumers)
- Durable
 - Disk based retention
 - Data is replicated across brokers
- Scalable
 - Expansions can be performed while the cluster is online
- High Performance
 - Producers, consumers, and brokers can all be scaled to handle very large message streams
 - Sub-second message latency to consumers

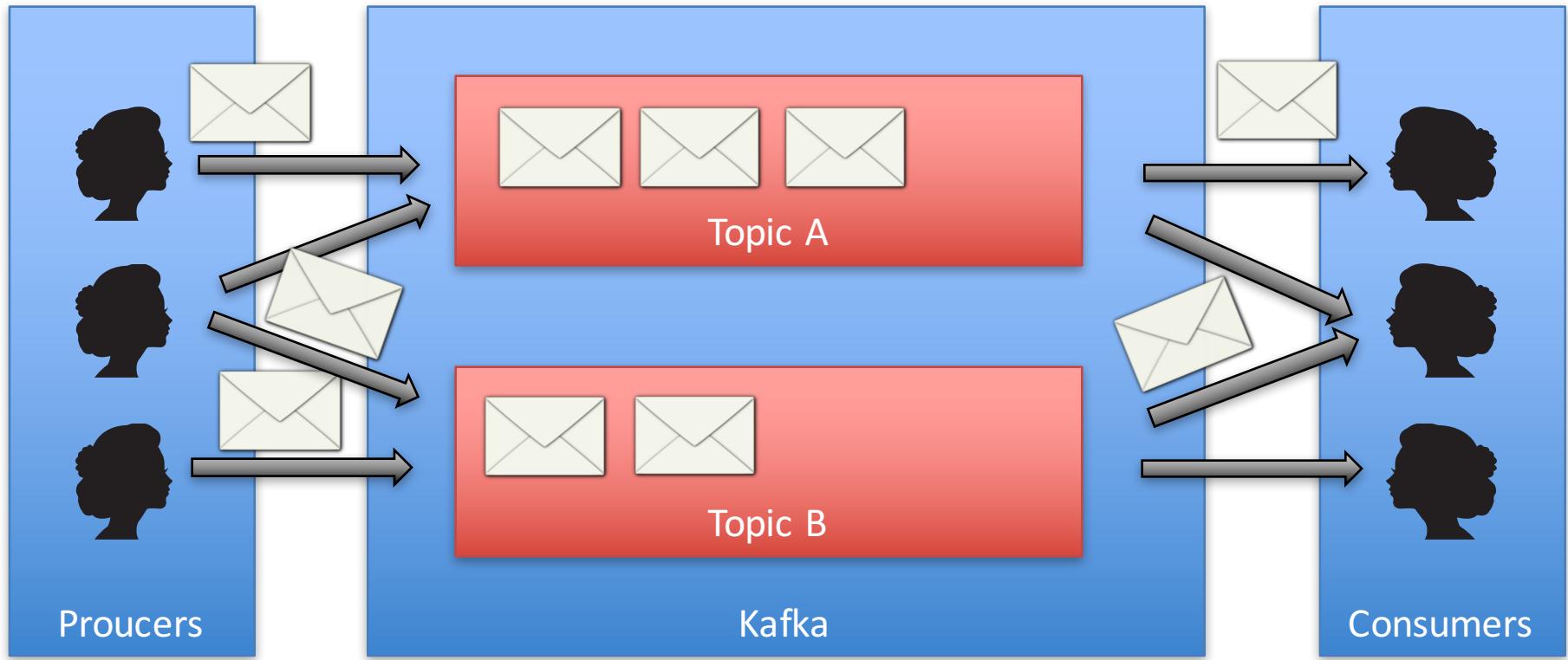
How Big is Big?

- Per day:
 - 800 billion messages
 - 175 TB of data
 - 650 TB of consumed data
- Per second:
 - 13 million messages
 - 2.75 GB of data
- Configuration
 - 1100 Kafka brokers
 - 60 clusters



Numbers from LinkedIn

Kafka Main Concepts: Topics

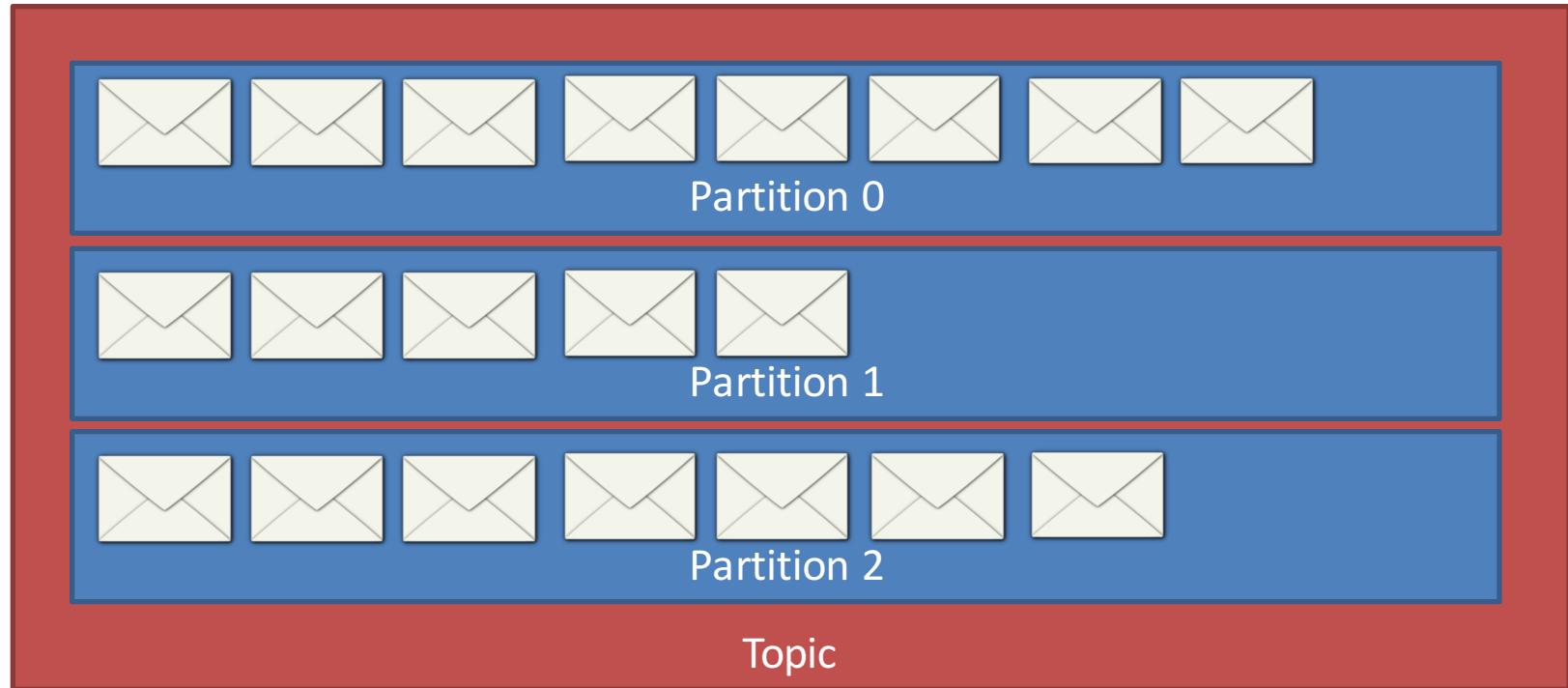


Kafka Main Concepts: Message

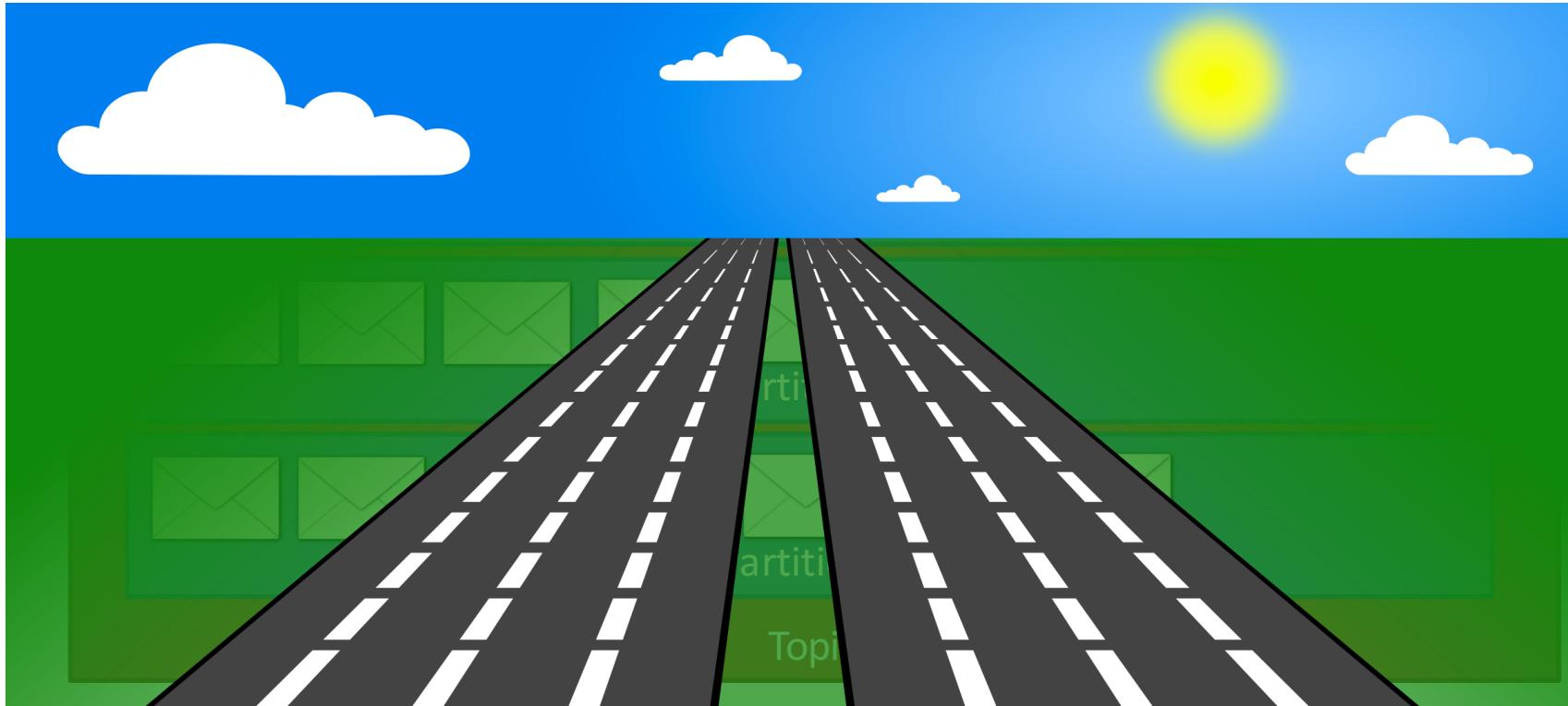


- Kafka looks at every message as a sequence of bytes
- The bytes are separated into:
 - Key
 - Used to determine partition (more about that later)
 - Value
 - The actual payload of the message

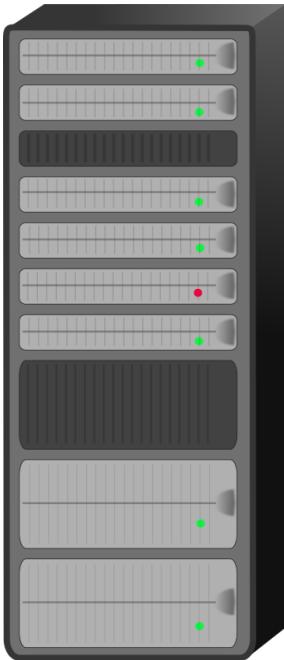
Kafka Key Concepts: Partitions



Kafka Key Concepts: Partitions



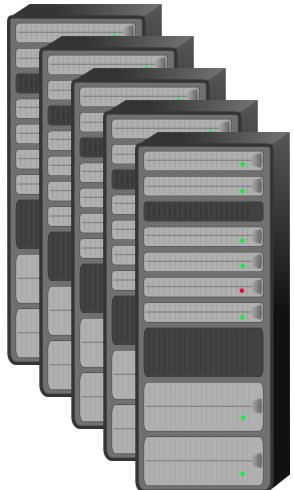
Kafka Main Concepts: Broker



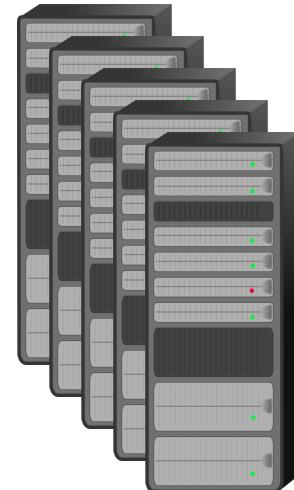
- A server or process running Kafka
- Holds multiple partitions across multiple topics
- The physical manifestation of Kafka

Kafka Main Concepts: Cluster

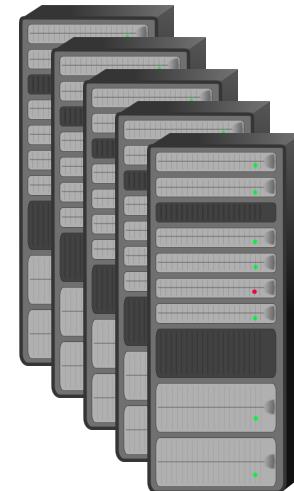
Availability Zone



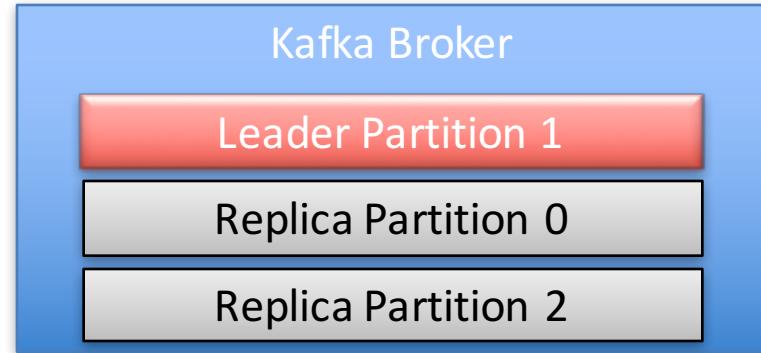
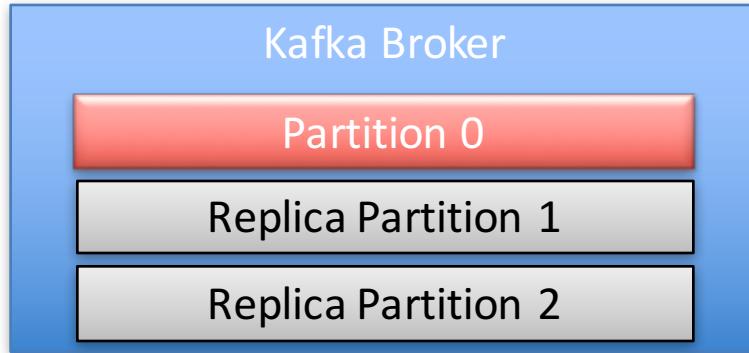
Availability Zone



Availability Zone



Kafka Main Concepts: Partitions and Clusters



Summary

- Apache Kafka is an open source, distributed, partitioned, and replicated commit-log based publish-subscribe messaging system
 - Scalable
 - High Performance
 - Multiple Consumers
 - Multiple Producers
 - Disk-based Retention

Start of Exercise

- Do you have Docker installed?
- We will be using Docker to run Kafka
- If you have not done so already, please install Docker
 - <https://docs.docker.com/engine/installation/>
- Check if Docker works by running

Check installation of Docker

```
$ docker -v  
Docker version 1.13.0, build 49bf474  
$ docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
78445dd45222: Already exists  
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.
```

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

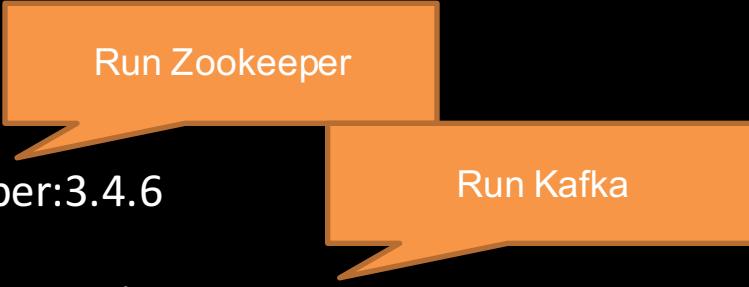
```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:
<https://cloud.docker.com/>

For more examples and ideas, visit:
<https://docs.docker.com/engine/userguide/>

docker-compose.yml

```
version: '2'
services:
  zookeeper:
    image: zookeeper:3.4.6
  kafka:
    image: wurstmeister/kafka:1.1.0
    environment:
      HOSTNAME_COMMAND: "echo $HOSTNAME"
      KAFKA_ADVERTISED_PORT: 9092
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    depends_on:
      - zookeeper
```



Essential Kafka CLI Commands (used in exercise)

- Kafka can be configured via the command
- Key commands
 - kafka-topics.sh
 - Allows us to manipulate and view topics
 - kafka-console-producer.sh
 - Allows us to produce data from using stdin
 - kafka-console-consumer.sh
 - Allows us to consume messages from the console
- We're running all tools in docker, so we have to 'reach into' the docker instance, hence our commands look like this:
 - docker-compose exec kafka /opt/kafka/bin/CMD
 - You may want to alias these commands to reduce typing or simply run a bash shell inside the docker image
 - docker-compose exec kafka /bin/bash

Lab

- In this lab we'll simply ensure that we can run Kafka
- We'll run Kafka using Docker
 - Easy configuration
 - Runs Kafka and Zookeeper
 - Flexible setup that allows us to setup a cluster of machines for later exercises
- The lab simply:
 1. Starts the docker images using docker-compose
 2. Runs a simple producer
 3. Runs a simple consumer
 4. Allows you to type messages in the producer and see them consumed by the consumer

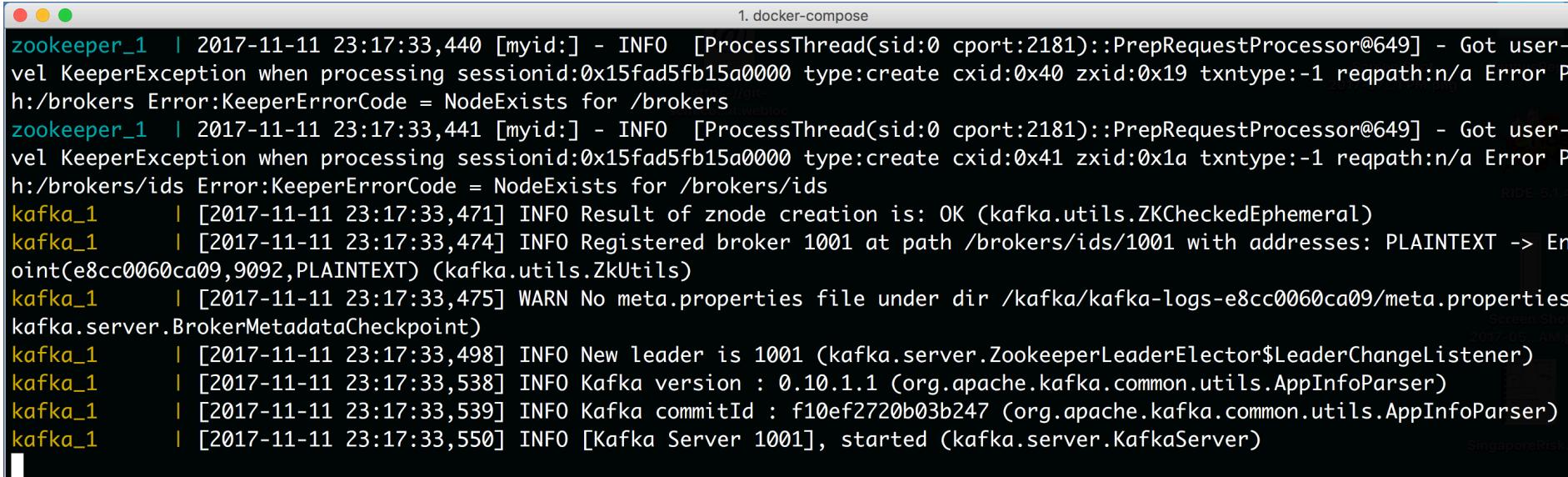
Step 1: Start Kafka and ZooKeeper

- a) In a terminal, go to:

kafka-labs/labs/01-Verify-Installation

- b) Run:

docker-compose up



The screenshot shows a terminal window with the title "1. docker-compose". The output of the command "docker-compose up" is displayed, showing logs from three containers: "zookeeper_1", "kafka_1", and "kafka_2". The logs indicate the startup of ZooKeeper and Kafka servers, including the creation of brokers and the election of a new leader broker (1001). The terminal window has a dark background with light-colored text.

```
zookeeper_1 | 2017-11-11 23:17:33,440 [myid:] - INFO  [ProcessThread(sid:0 cport:2181)::PrepRequestProcessor@649] - Got user-level KeeperException when processing sessionid:0x15fad5fb15a0000 type:create cxid:0x40 zxid:0x19 txntype:-1 reqpath:n/a Error Path /brokers Error:KeeperErrorCode = NodeExists for /brokers
zookeeper_1 | 2017-11-11 23:17:33,441 [myid:] - INFO  [ProcessThread(sid:0 cport:2181)::PrepRequestProcessor@649] - Got user-level KeeperException when processing sessionid:0x15fad5fb15a0000 type:create cxid:0x41 zxid:0x1a txntype:-1 reqpath:n/a Error Path /brokers/ids Error:KeeperErrorCode = NodeExists for /brokers/ids
kafka_1    | [2017-11-11 23:17:33,471] INFO Result of znode creation is: OK (kafka.utils.ZKCheckedEphemeral)
kafka_1    | [2017-11-11 23:17:33,474] INFO Registered broker 1001 at path /brokers/ids/1001 with addresses: PLAINTEXT -> Endoint(e8cc0060ca09,9092,PLAINTEXT) (kafka.utils.ZkUtils)
kafka_1    | [2017-11-11 23:17:33,475] WARN No meta.properties file under dir /kafka/kafka-logs-e8cc0060ca09/meta.properties
kafka.server.BrokerMetadataCheckpoint)
kafka_1    | [2017-11-11 23:17:33,498] INFO New leader is 1001 (kafka.server.ZookeeperLeaderElector$LeaderChangeListener)
kafka_1    | [2017-11-11 23:17:33,538] INFO Kafka version : 0.10.1.1 (org.apache.kafka.common.utils.AppInfoParser)
kafka_1    | [2017-11-11 23:17:33,539] INFO Kafka commitId : f10ef2720b03b247 (org.apache.kafka.common.utils.AppInfoParser)
kafka_1    | [2017-11-11 23:17:33,550] INFO [Kafka Server 1001], started (kafka.server.KafkaServer)
```

Step 2: Create a Topic

- a) Run a second terminal and go to the directory
kafka-labs/labs/01-Verify-Installation

- b) In this directory run:

```
docker-compose exec kafka
/opt/kafka/bin/kafka-topics.sh
--create
--zookeeper zookeeper:2181
--replication-factor 1
--partitions 1
--topic helloworld
```

```
LM-SJN-21001415:gte pgraff$ cd ~/kafka/kafka-lab/labs/01-Verify-Installation/
LM-SJN-21001415:01-Verify-Installation pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-topics.sh --create --
ookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic helloworld
WARNING: The HOSTNAME variable is not set. Defaulting to a blank string.
Created topic "helloworld".
LM-SJN-21001415:01-Verify-Installation pgraff$ █
```

Step 3: Run the Kafka Producer

- a) In the same terminal as you created the topic, in the directory:

kafka-labs/labs/01-Verify-Installation

- b) In this directory run:

docker-compose exec kafka

/opt/kafka/bin/kafka-console-producer.sh --bootstrap-server kafka:9092 --topic helloworld

```
× docker-compose
LM-SJN-21001415:01-Verify-Installation pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-console-producer.sh --broker-list kafka:9092 --topic helloworld
WARNING: The HOSTNAME variable is not set. Defaulting to a blank string.
Hello World!
```

Step 3: Run the Kafka Producer

- a) Run a third terminal and go to the directory
kafka-labs/labs/01-Verify-Installation
- b) In this directory run:

```
docker-compose exec kafka  
/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-  
server kafka:9092  
--topic helloworld  
--from-beginning
```

```
LM-SJN-21001415:01-Verify-Installation pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic helloworld --from-beginning  
WARNING: The HOSTNAME variable is not set. Defaulting to a blank string.  
Hello World!
```

```
LM-SJN-21001415:01-Verify-Installation pgraaff$ docker-compose exec kafka /opt/kafka_2.11  
-0.10.1.1/bin/kafka-console-producer.sh --broker-list kafka:9092 --topic helloworld  
WARNING: The HOSTNAME variable is not set. Defaulting to a blank string.  
Hello World!  
This is a test message  
This is another message that will show up at the consumer
```

```
1001]: Finished loading offsets from [__  
consumer_offsets,39] in 1 milliseconds.  
(kafka.coordinator.GroupMetadataManager)  
kafka_1 | [2017-11-11 23:33:51,461]  
INFO [Group Metadata Manager on Broker  
1001]: Loading offsets and group metadata  
from [__consumer_offsets,42] (kafka.coordinator.GroupMetadataManager)  
kafka_1 | [2017-11-11 23:33:51,462]  
INFO [Group Metadata Manager on Broker  
1001]: Finished loading offsets from [__  
consumer_offsets,42] in 1 milliseconds.  
(kafka.coordinator.GroupMetadataManager)  
kafka_1 | [2017-11-11 23:33:51,463]  
INFO [Group Metadata Manager on Broker  
1001]: Loading offsets and group metadata  
from [__consumer_offsets,45] (kafka.coordinator.GroupMetadataManager)  
kafka_1 | [2017-11-11 23:33:51,464]  
INFO [Group Metadata Manager on Broker  
1001]: Finished loading offsets from [__  
consumer_offsets,45] in 1 milliseconds.  
(kafka.coordinator.GroupMetadataManager)  
kafka_1 | [2017-11-11 23:33:51,464]  
INFO [Group Metadata Manager on Broker  
1001]: Loading offsets and group metadata  
from [__consumer_offsets,48] (kafka.coordinator.GroupMetadataManager)  
kafka_1 | [2017-11-11 23:33:51,466]  
INFO [Group Metadata Manager on Broker  
1001]: Finished loading offsets from [__  
consumer_offsets,48] in 2 milliseconds.  
(kafka.coordinator.GroupMetadataManager)
```

9. Finally, stop the Kafka and Zookeeper servers with Docker Compose:

```
...  
$ docker-compose down  
...
```

Congratulations, this lab is complete!

```
LM-SJN-21001415:01-Verify-Installation pgraaff$ docker-compose exec kafka /opt/kafka_2.11  
-0.10.1.1/bin/kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic helloworld  
--from-beginning  
WARNING: The HOSTNAME variable is not set. Defaulting to a blank string.
```

```
Hello World!  
This is a test message  
This is another message that will show up at the consumer
```

Final Step: Shutdown Kafka and Zookeeper

- Final step is to shutdown the experiment
 1. Control-C out of the Producer and the Consumer
 2. In one of the terminals (e.g., Producer) run:
docker-compose down

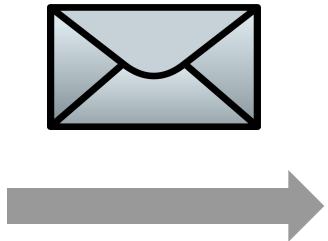
```
^CLM-SJN-21001415:01-Verify-Installation pgraff$ docker-compose down
WARNING: The HOSTNAME variable is not set. Defaulting to a blank string.
Stopping 01verifyinstallation_kafka_1    ... done
Stopping 01verifyinstallation_zookeeper_1 ... done
Removing 01verifyinstallation_kafka_1    ... done
Removing 01verifyinstallation_zookeeper_1 ... done
Removing network 01verifyinstallation_default
LM-SJN-21001415:01-Verify-Installation pgraff$ █
```

Producer and Consumer

Outline

- Producers
 - What is a producer?
 - The producer API
- What is a Consumer?
 - What is a consumer and a consumer group?
 - The consumer API
- Anatomy of messages

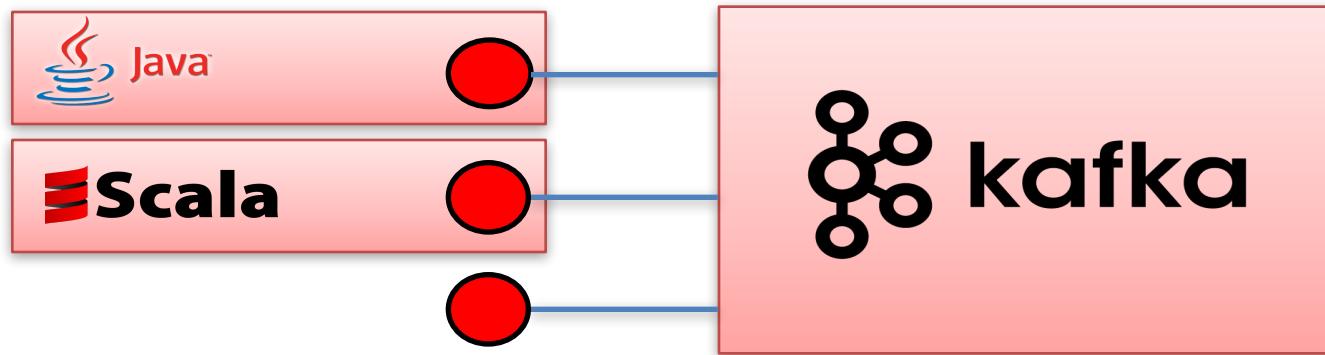
What is a Producer?



➤ Producer

- Producers produces the data sent to the Kafka clusters
 - Sent via topics
 - Directly involved in load-balancing
 - Controls the resiliency of messages

Kafka APIs



- Kafka ships with built in client APIs for developers to use with applications
 - Kafka ships with a Java client that is recommended
 - Legacy Scala clients are still included
 - Kafka also includes a **binary wire protocol**
 - Many tools in other languages that implement this wire protocol

The Java API

Generic sender where:
K = Type of key
V = Type of message

Constructor takes a configuration
(mostly a hashmap of options)

Send a messages (with or without
callbacks)

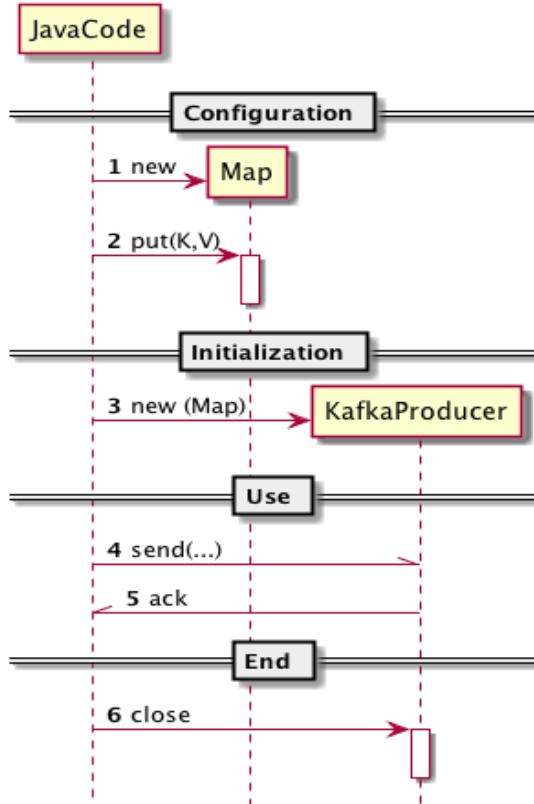
Get metrics for this producer

org.apache.kafka.clients.producer

KafkaProducer<K,V>

```
KafkaProducer(config: Properties) send(ProducerRecord<K,V>):  
Future<RecordMetaData>  
send(ProducerRecord<K,V>, Callback): Future<...>  
flush()  
metrics(): Map<MetricName, ? extends Metric>  
close()
```

Java API Behavior



// Configuration

```
Properties kp = new Properties();
kp.put("bootstrap.servers",
       "mybroker1:9092,mybroker2:9092");
kp.put("key.serializer", "...");
```

// Initialization

```
KafkaProducer<String, String> producer =
    new KafkaProducer<String, String>(kp);
```

// Use

```
Future<...> f = producer.send(...);
... f.get(); // when acked
```

// End

```
producer.close();
```

Creating a Kafka Producer

- Constructing a Kafka producer requires 3 mandatory properties
 - `bootstrap.servers` – list of host:port pairs of Kafka brokers. This doesn't have to include all brokers in the cluster as the producer will query about additional brokers. It is recommended to include at least 2 in case one broker goes down
 - `key.serializer` – should be set to a class that implements the `Serializer` interface that will be used to serialize **keys**
 - `value.serializer` - should be set to a class that implements the `Serializer` interface that will be used to serialize **values**

The ProducerRecord

Generic record where:
K = message key
V = Type of message

org.apache.kafka.clients.producer

ProducerRecord<K,V>

Message Key is optional

Partition Key is optional

key: K

value: V

topic: String

partition: Integer

ProducerRecord(topic: String, partition: Integer, key: K, value:V)

ProducerRecord(topic: String, key: K, value: V)

ProducerRecord(topic: String, value: V)

Sending a Message

```
ProducerRecord<String, String> record =  
    new ProducerRecord<String, String>(  
        "someTopic", "someKey", "someValue");  
  
producer.send(record, new Callback() {  
    public void onCompletion(  
        RecordMetadata metadata, Exception e) {  
            if(e != null) e.printStackTrace();  
            System.out.println(  
                "Offset: " + metadata.offset());  
    }  
});
```

Producer Controls Message Guarantees

- As a producer you can determine what guarantees you want Kafka to give you when sending a message
 - Controlled by acknowledgement
- Different use cases require different guarantees
 - Web page clicks log → Don't care if I loose a few messages
 - Credit card payment → I want best possible guarantee
- Kafka provides options
 - The better guarantee, the lower latency

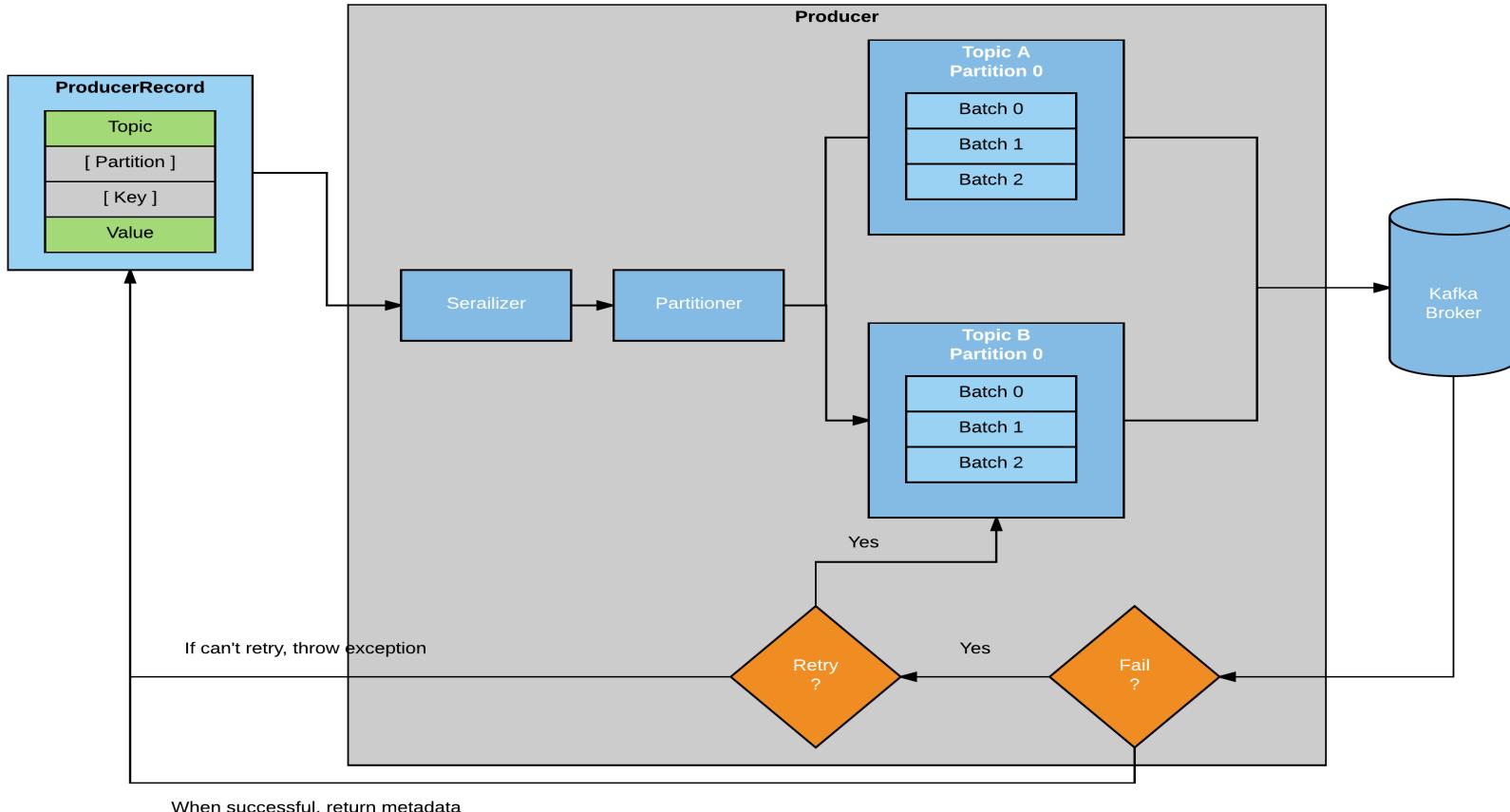
Producer API

- Different use case requirements will influence the way the producer API is used to write messages to Kafka and its configuration
- Three primary ways of sending messages
 - Fire-and-forget – send a message and don't really care if it arrived successfully or not. Most of the time it will arrive successfully but it's possible that some messages will get lost
 - Synchronous send – message is sent and a Future object is returned which can be used to see if the send() was successful
 - Asynchronous send – the send() method has a callback function which is triggered when a response is received from the Kafka broker

Acknowledgement of Messages

- No ack (0)
 - Kafka will most likely receive the message
 - Producer will not wait for any reply from the broker before assuming the message was sent successfully
 - If something goes wrong, producer will not know and message is lost
 - Because producer is not waiting for a response, high throughput can be achieved
- Ack from N replicas (1..N)
 - A message is not considered consumed by the Kafka cluster unless N replicas holding the message has acknowledged
- Ack from all replicas (-1)
 - Every replica must acknowledge the message

Producer Overview



When successful, return metadata

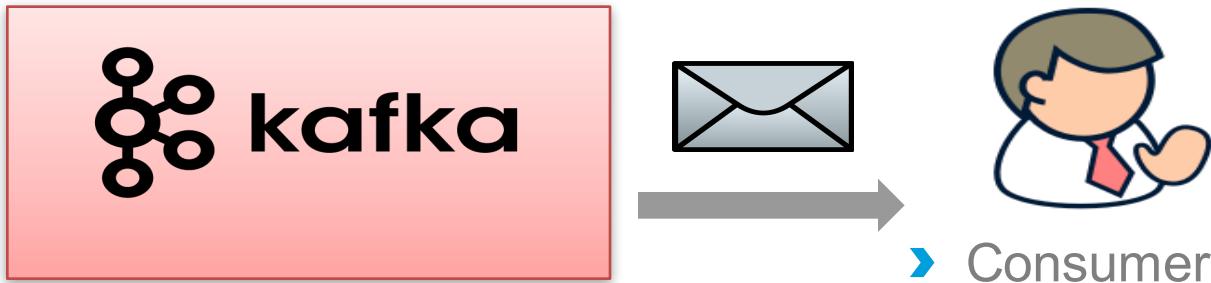
Serialization

- Kafka messages are byte arrays (to Kafka)
 - Key → Array of bytes
 - Value → Array of bytes
- The Java API allows you to pass any object as key or value
 - Makes the code readable, but...
 - ... requires serializes and deserializes
- Kafka includes an interface for this
`org.apache.kafka.common.serialization.Serializer`

Built-in Serializers

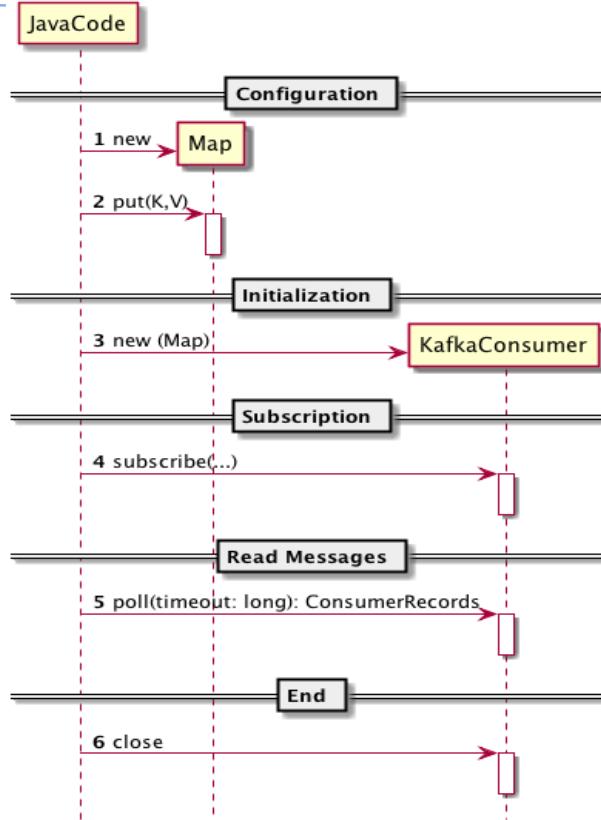
- Kafka includes serializers for common types:
 - ByteArraySerializer
 - StringSerializer
 - IntegerSerializer
 - ...
- Most organizations settle on some standard serialization strategy
 - JSON, XML, Apache Avro, Protobuf

Consumers and Consumer Groups



- Applications that read data from Kafka are consumers
 - Subscribes to topics
 - Use KafkaConsumer to read messages from these topics
 - Kafka consumers are usually part of a *consumer group*
 - **The main way consumption of data from a Kafka topic is scaled is by adding more consumers to a consumer group**

Java API Behavior



```
// Configuration
Properties kp = new Properties();
kp.put("bootstrap.servers",
       "mybroker1:9092,mybroker2:9092");
kp.put("key.deserializer", "...");

// Initialization
KafkaConsumer<...> consumer=
    new KafkaConsumer<...>(kp);

// Subscription
consumer.subscribe("interesting.*");

// Read messages
ConsumerRecords<...> records = consumer.poll(100);
for (ConsumerRecord<...> cr : records) {
    // cr.value(); cr.key(); cr.offset();
}

// End
consumer.close()
```

The Java API

Generic sender where:
K = Type of key
V = Type of message

Constructor takes a configuration
(mostly a hashmap of options)

Multiple ways to subscribe.
Subscribe by list, wildcard, etc.

Read messages from Kafka

Multiple (sync/async) ways to
confirm reception to consumer
groups

org.apache.kafka.clients.consumer

KafkaConsumer<K,V>

KafkaConsumer(config: Properties)
subscribe(...)
poll(timeout: long): ConsumerRecords<K,V>
commit(...)
...

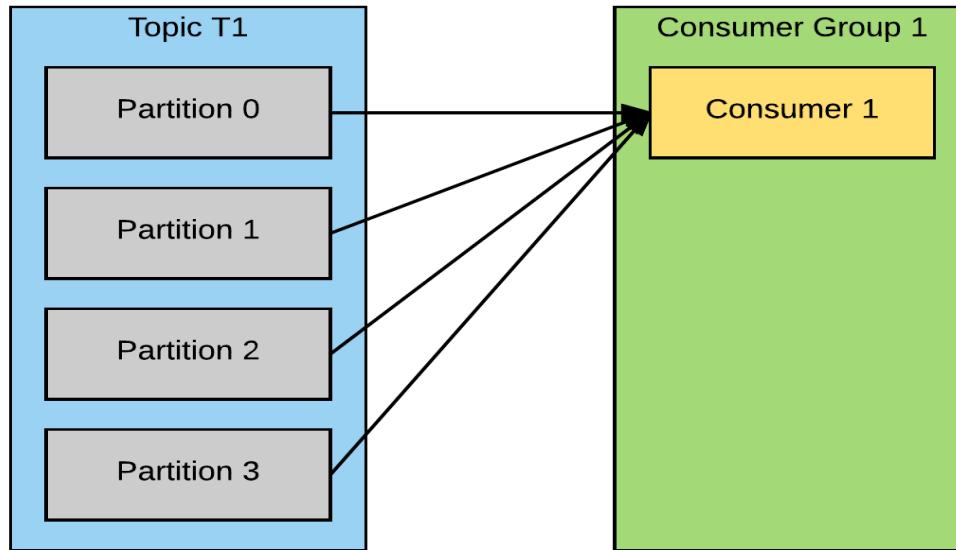
A set of other methods such as: metrics(), pause(...), assign(...), close(), etc

How to Use the API?

- The *org.apache.kafka.clients.consumer.KafkaConsumer* acts as a proxy for the consumer
- Some key issues to resolve
 - Setup of consumer groups
 - Which topics to subscribe to
 - Which partitions to subscribe to (optional)
 - Manual or automatic offset management
 - Multi-threaded or single-threaded consumption

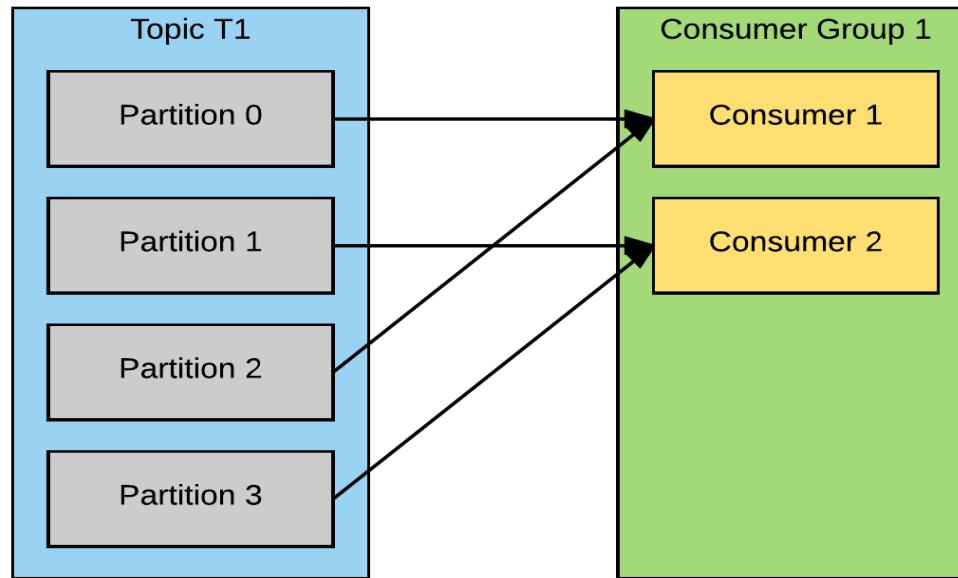
Consumer Group

- The consumer will receive all messages from all four partitions



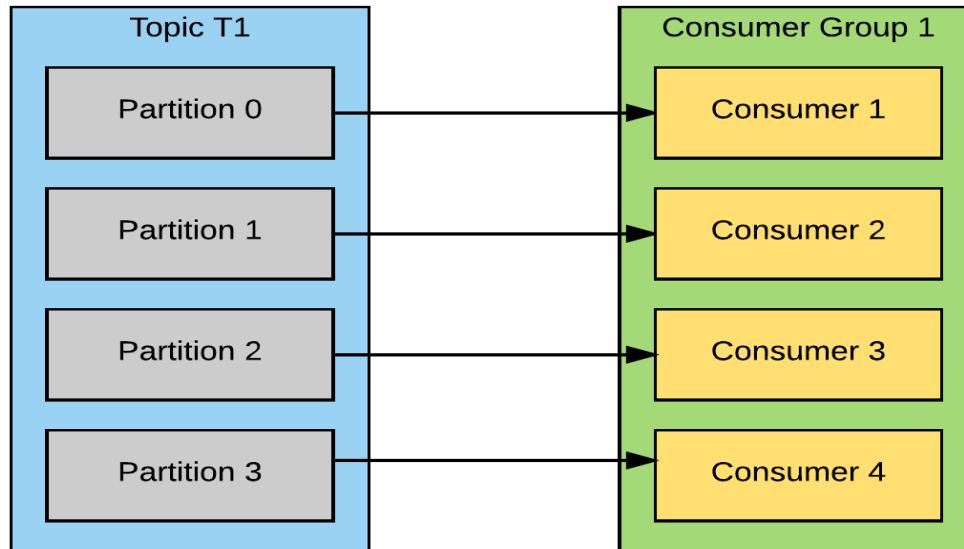
Consumer Group

- Each consumer will only get messages from two partitions



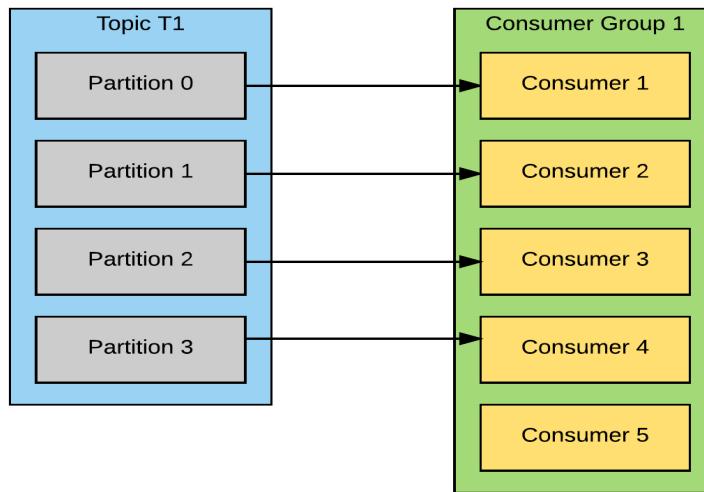
Consumer Group

- If the consumer group has the same number of consumers as partitions, each will read messages from a single partition



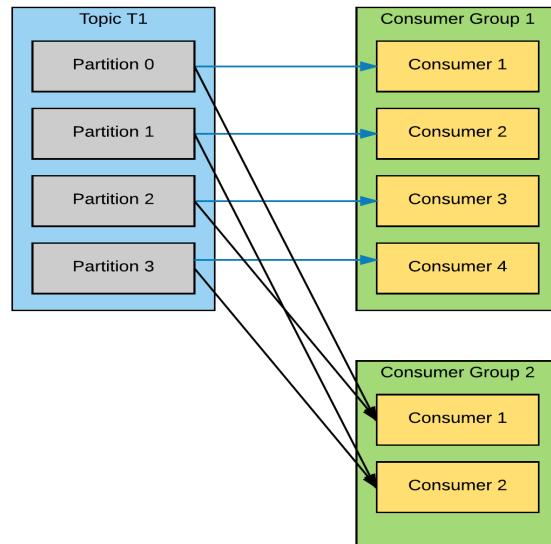
Consumer Group

- If there are more consumers than partitions for a topic, some consumers will be idle and receive no messages
- Create topics with a large number of partitions to allow adding more consumers when load increases



Multiple Consumer Groups

- One of the main design goals of Kafka was to allow multiple applications the ability to read data from the same topic
- Make sure each application has its own consumer group for this purpose



Partition Rebalance

- Consumers in a consumer group share ownership of the partitions in the topics they subscribe to
- When a new consumer is **added** to the group, it consumes messages from partitions which were previously consumed by another consumer
- When a consumer **leaves** the group (shuts down, crashes, etc), the partitions it used to consume will be consumed by one of the remaining consumers
- Reassignment of partitions to consumers can also happen when topics are modified – an administrator adds new partitions, for example

Partition Rebalance

- The event in which partition ownership is moved from one consumer to another is called a *rebalance*
- During a rebalance, consumers can't consume messages!
- Steps can be taken to safely handle rebalances and avoid unnecessary rebalances

Creating a Kafka Consumer

- Creating a `KafkaConsumer` is similar to creating a `KafkaProducer`
- Like the producer, you must specify `bootstrap.servers`,
`key.deserializer`, and `value.deserializer` in a
Properties object
- You must also specify a `group.id` which specifies the consumer group
for which the `KafkaConsumer` instance belongs to

Setup of KafkaConsumer Example

```
private Properties kafkaProps = new Properties();
kafkaProps.put("bootstrap.servers", "mybroker1:9092,mybroker2:9092");

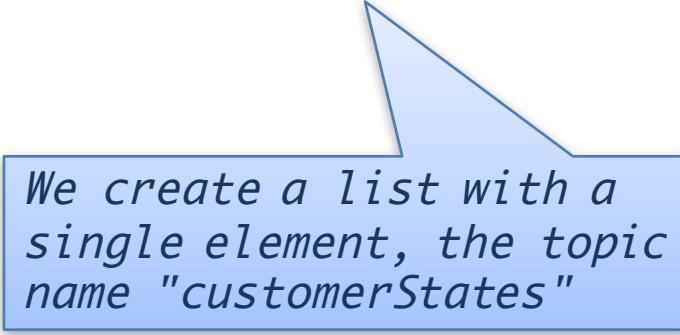
kafkaProps.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
kafkaProps.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
kafkaProps.put("group.id", "StateCounter");

KafkaConsumer<String, String> consumer = new KafkaConsumer<String,
String>(kafkaProps);
```

Subscribing to Topics

- Once a consumer is created, you can subscribe to one or more topics

```
consumer.subscribe(Collections.singletonList("customerStates"));
```



We create a list with a single element, the topic name "customerStates"

Subscribing with Regular Expressions

- You can also subscribe to topics using regular expressions
- The expression can match multiple topic names
- If a new topic is created with a name that matches, a rebalance will happen and consumers will start consuming from the new topic
- Useful for applications that need to consume from multiple topics
- Example: subscribe to all test topics
 - `consumer.subscribe("test.*");`

Consumer Poll Loop

- Once a consumer subscribes to topics, a loop polls the server for more data

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("foo", "bar"));
while (true) {
    ConsumerRecords<String, String> records = consumer.poll(100);

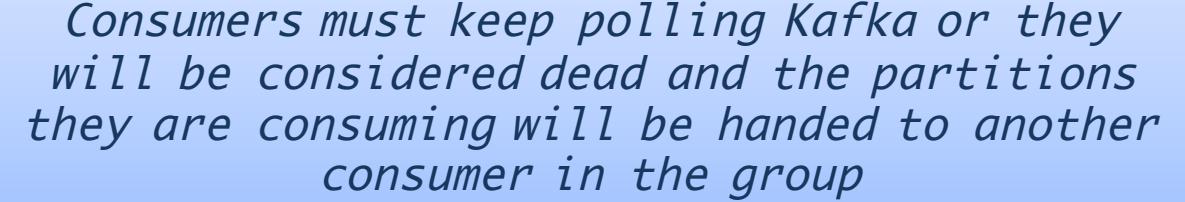
    for (ConsumerRecord<String, String> record : records)
        System.out.printf("offset = %d, key = %s, value = %s",
                          record.offset(),
                          record.key(),
                          record.value());
} finally {
    consumer.close();
}
```

Consumer Poll Loop

- Once a consumer subscribes to topics, a loop polls the server for more data

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("foo", "bar"));
while (true) {
    ConsumerRecords<String, String> records = consumer.poll(100);

    for (ConsumerRecord<String, String> record : records)
        System.out.printf("offset = %d, key = %s, value = %s", record.offset(),
record.key(), record.value());
} finally {
    consumer.close();
}
```



Consumers must keep polling Kafka or they will be considered dead and the partitions they are consuming will be handed to another consumer in the group

Poll Method

- `poll()` returns a list of records that contain:
 - Topic and partition the record came from
 - Offset of the record within the partition
 - Key and value of the record
- Takes a timeout parameter that specifies how long it will take to return, with or without data
 - *How fast do you want to return control to the thread that does the polling?*

Multithreading Considerations

- **One consumer per thread**
- To run multiple consumers in the same group in one application, each must run in its own thread
- You can wrap the consumer logic in its own object and then use Java's `ExecutorService` to start multiple threads each with its own consumer

Commits

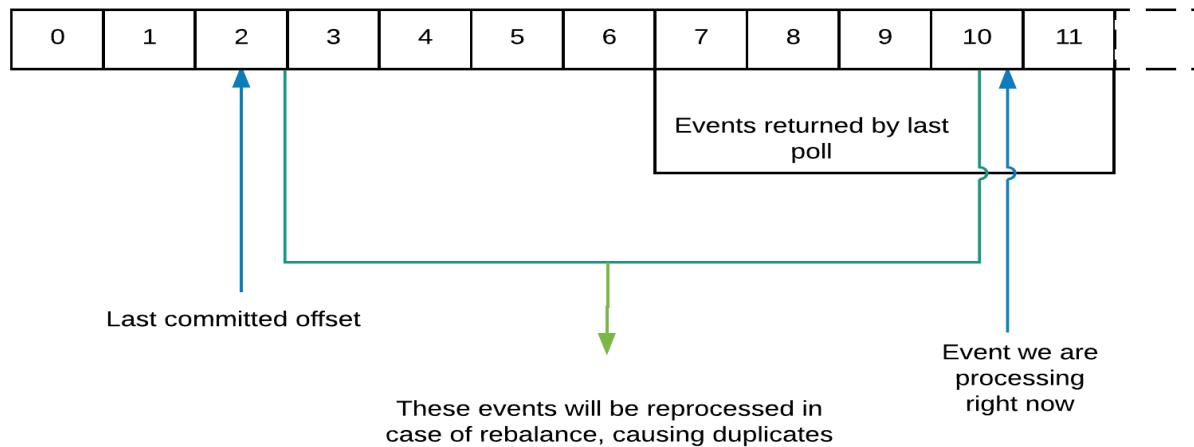
- Unlike other JMS queues, Kafka does not track acknowledgements from consumers
- When `poll()` is called, it returns records that consumers in the group have not yet read
- The records that have been read by a consumer of the group are tracked by their position (offset) in each partition
- When the current position in the partition is updated, it is known as a *commit*

Consumers Commit Offsets

- Consumers send a message to Kafka to a reserved topic with the committed offset for each partition
- If a consumer crashes or a new consumer joins the consumer group, a rebalance is triggered
- After a rebalance, a consumer may be assigned a new set of partitions and must read the latest committed offset of each partition and continue from there

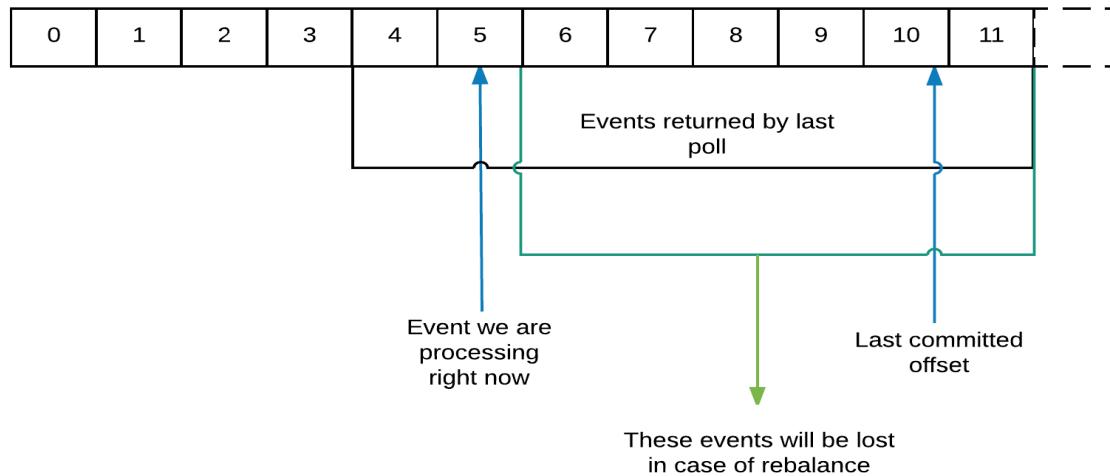
Messages Processed Twice

- If the committed offset is smaller than the offset of the last message the client processed, the messages between the last processed offset and the committed offset will be processed twice



Messages Missed

- If the committed offset is larger than the offset of the last message the client processed, messages between the last processed and the committed offset will be missed by the consumer group



Automatic Commit

- If you configure `enable.auto.commit=true`, the consumer will commit the largest offset your client received from `poll()` every 5 seconds by default
- Whenever there is a `poll`, the consumer checks if its time to commit and if so, will commit the offsets it returned in the last poll
- Convenient but don't give developers enough control to avoid duplicate messages

Commit Current Offset

- Developers usually want to exercise control over the time offsets are committed to eliminate possibility of missing messages **and** reduce the number of duplicate messages during rebalancing
- Setting `auto.commit.offset=false` means that offsets will only be committed explicitly
- Consumer has a `commitSync()` API to commit the latest offset returned by `poll()`

commitSync Example

```
while (true) {  
    ConsumerRecords<String, String> records = consumer.poll(100);  
  
    for (ConsumerRecord<String, String> record : records)  
        System.out.printf("offset = %d, key = %s, value = %s", record.offset(),  
    record.key(), record.value());  
  
    try {  
        consumer.commitSync();  
    } catch (CommitFailedException e) {  
        log.error("commit failed", e);  
    }  
  
} finally {  
    consumer.close();  
}
```

*Once we are done
processing all records in
the current batch,
commitSync is called
before polling for more*

Asynchronous Commit

- The application is blocked until the broker responds to the commit request with commitSync() limiting throughput
- An alternative is to use commitAsync() which commits the last offsets and continues

```
while (true) {  
    ConsumerRecords<String, String> records = consumer.poll(100);  
  
    for (ConsumerRecord<String, String> record : records)  
        System.out.printf("offset = %d  
record.key(), record.value());  
  
    consumer.commitAsync();  
}
```

You can also pass a callback which is invoked by the consumer when the commit finishes (either successfully or not)

Summary

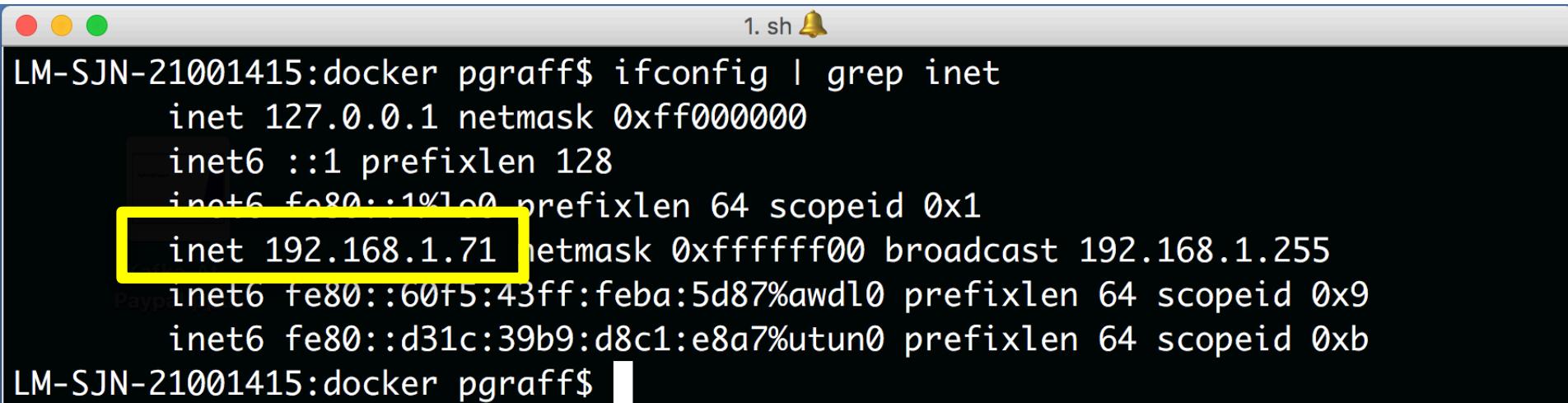
- Producers
 - API
 - Sending messages
 - Serialization
- Consumers
 - API
 - Subscribing
 - Consumer groups
 - Rebalance

Lab

- In this lab we'll use the Java API to consume and produce messages
- Two projects
 - Producer
 - Consumer
- Required tools:
 - Java
 - Maven
 - Docker (as before)
- Maven can be run stand alone or using a docker-container

Step 1: Configure Your docker-compose.yml file

- We need to configure the docker-compose.yml with the IP of your machine
- To find the IP (**ifconfig** on the Mac, **ipconfig** on a Windows machine)



```
1. sh 🎙
LM-SJN-21001415:docker pgraff$ ifconfig | grep inet
    inet 127.0.0.1 netmask 0xff000000
    inet6 ::1 prefixlen 128
    inet6 fe80::1%1c0 prefixlen 64 scopeid 0x1
    inet 192.168.1.71 netmask 0xfffffff00 broadcast 192.168.1.255
    inet6 fe80::60f5:43ff:feba:5d87%awdl0 prefixlen 64 scopeid 0x9
    inet6 fe80::d31c:39b9:d8c1:e8a7%utun0 prefixlen 64 scopeid 0xb
LM-SJN-21001415:docker pgraff$ █
```

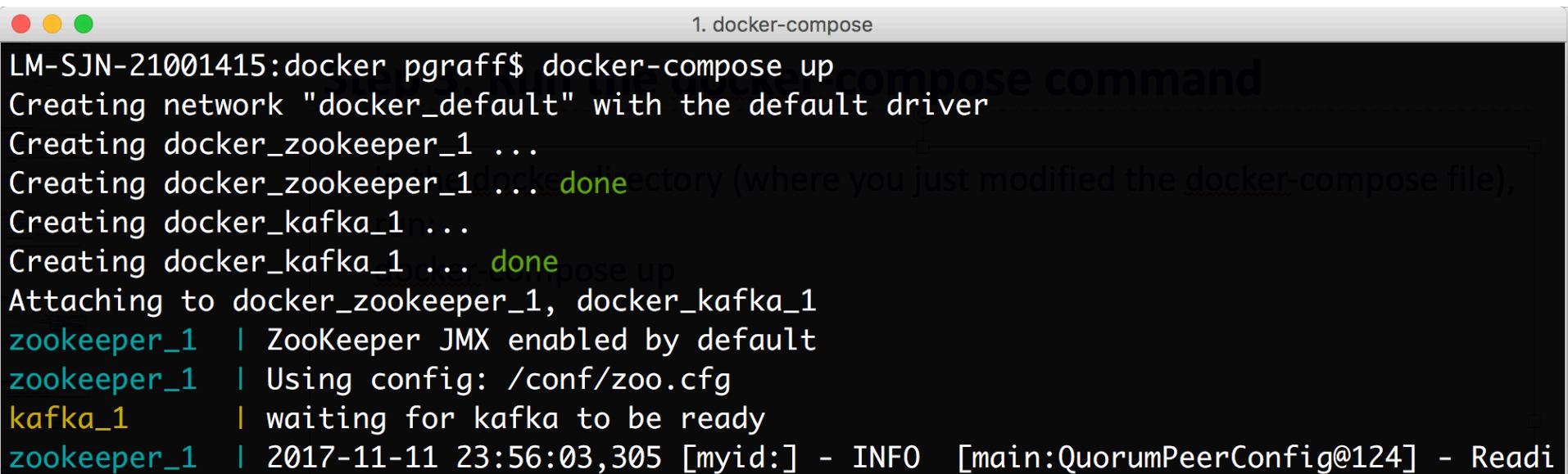
Step 2: Edit your docker-compose.yml file

- Using your favorite editor, edit the docker-compose.yml file in the directory:
kafka-labs/labs/02-Publish-And-Subscribe/docker
- Look for the [insert] statement and replace it with your IP... E.g.:

```
- 9092:9092
- 7203:7203
environment:
  KAFKA_ADVERTISED_HOST_NAME: 192.168.1.70
  KAFKA_ADVERTISED_PORT: 9092
  KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
depends_on:
```

Step 3: Run the docker-compose command

- In the docker directory (where you just modified the docker-compose file), run:
docker-compose up



The screenshot shows a terminal window on a Mac OS X system (indicated by the red, yellow, and green window control buttons). The title bar of the terminal says "1. docker-compose". The command entered is "docker pgraff\$ docker-compose up". The terminal output shows the creation of a network and two services: "docker_zookeeper_1" and "docker_kafka_1". The Kafka service is shown as "done". Finally, it attaches to both services. The ZooKeeper logs mention JMX enabled by default and config files. The Kafka log shows it's waiting for kafka to be ready.

```
LM-SJN-21001415:docker pgraff$ docker-compose up
Creating network "docker_default" with the default driver
Creating docker_zookeeper_1 ...
Creating docker_zookeeper_1 ... done
Creating docker_kafka_1 ...
Creating docker_kafka_1 ... done
Attaching to docker_zookeeper_1, docker_kafka_1
zookeeper_1  | ZooKeeper JMX enabled by default
zookeeper_1  | Using config: /conf/zoo.cfg
kafka_1      | waiting for kafka to be ready
zookeeper_1  | 2017-11-11 23:56:03,305 [myid:] - INFO  [main:QuorumPeerConfig@124] - Readi
```

Step 4: Create the Topics

- In a second terminal, cd into the directory
kafka-lab/labs/02-Publish-And-Subscribe/docker
- Create the two topics required for this exercise (user-event, global-events)
 - user-events
 - global-events

```
x sh
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-topics.sh
--create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic user-events
Created topic "user-events".
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-topics.sh
--create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic global-events
Created topic "global-events".
LM-SJN-21001415:docker pgraff$
```

Step 5: Build the Producer

- Use maven (either natively or using the docker image as described in the exercise description on GitHub)
- Build using mvn package from the directory
kafka-lab/labs/02-Publish-And-Subscribe/producer

```
sh
LM-SJN-21001415:producer pgraff$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building pubsub-producer 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ pubsub-producer ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.5.1:compile (default-compile) @ pubsub-producer ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ pubsub-producer ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
```

Producer code

```
1  for (int i = 0; i < 100000; i++) {  
2      producer.send(new ProducerRecord< String, String > (  
3          "user-events", // topic  
4          "user_id_" + i, // key  
5          "some_value_" + System.nanoTime())); // value  
6  
7      if (i % 100 == 0) {  
8          String event = global_events[(int)(Math.random() * global_events.  
9  
10         producer.send(new ProducerRecord< String, String > (  
11             "global-events", // topic  
12             event)); // value  
13  
14         producer.flush();  
15         System.out.println("Sent message number " + i);  
16     }  
}
```

Step 6: Run the Publisher

- The maven job creates a 'real executable' (target/producer)
- We'll run the executable

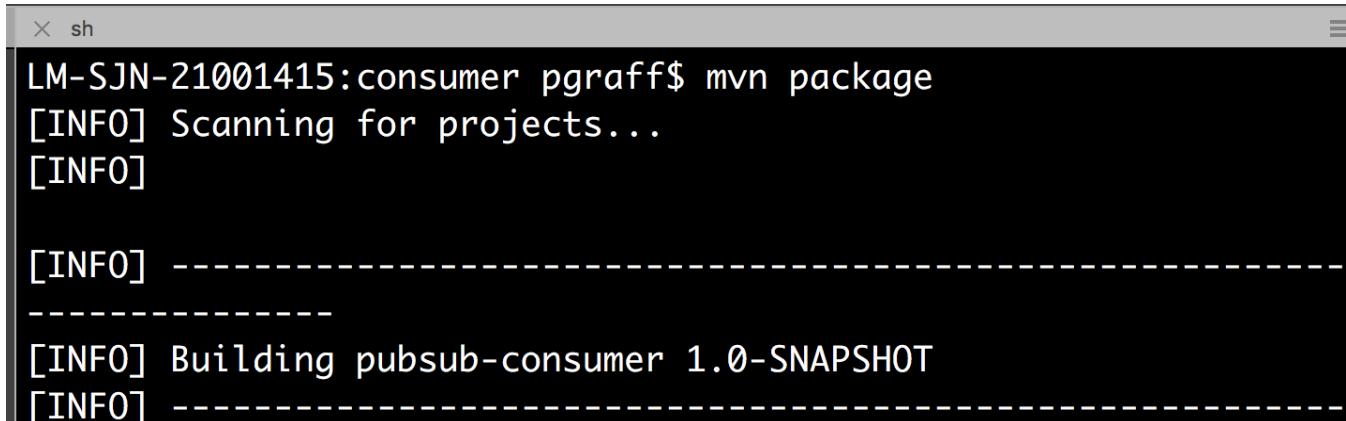


A terminal window titled "sh" showing the output of a command. The command is "producer pgraff\$ target/producer". The output shows the producer sending messages numbered from 0 to 900. It also includes SLF4J logging information about failed class loading and defaulting to a no-operation logger.

```
LM-SJN-21001415:producer pgraff$ target/producer
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.
Sent message number 0
Sent message number 100
Sent message number 200
Sent message number 300
Sent message number 400
Sent message number 500
Sent message number 600
Sent message number 700
Sent message number 800
Sent message number 900
```

Step 7: Build the Consumer

- Open another terminal (just so that we can run the producer again if we want to)
- Change into the directory
kafka-lab/labs/02-Publish-And-Subscribe/consumer
- Run the maven command to build:



A screenshot of a terminal window titled "sh". The window shows the command "mvn package" being run in the directory "consumer". The output includes Maven's [INFO] messages indicating the scanning of projects and the building of a snapshot version (1.0-SNAPSHOT) of the pubsub-consumer module.

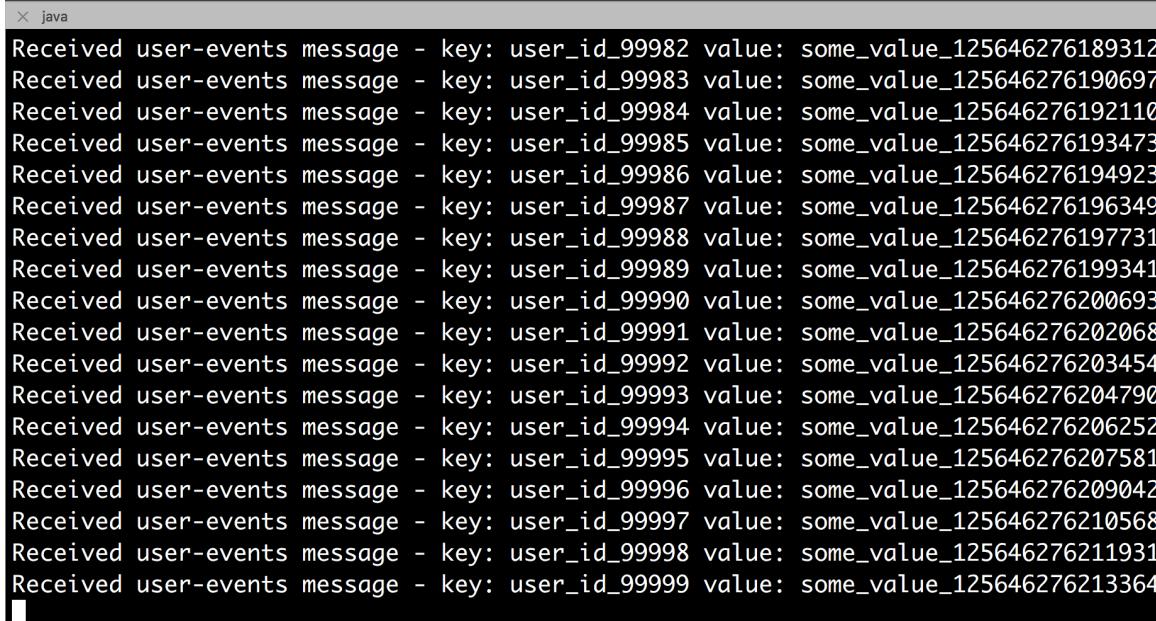
```
LM-SJN-21001415:consumer pgraff$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building pubsub-consumer 1.0-SNAPSHOT
[INFO] -----
```

The Consumer...

```
13 class Consumer {
14     public static void main(String[] args) throws IOException {
15         KafkaConsumer<String, String> consumer;
16         try (InputStream props = Resources.getResource("consumer.properties").openStream()) {
17             Properties properties = new Properties();
18             properties.load(props);
19             consumer = new KafkaConsumer<>(properties);
20         }
21         consumer.subscribe(Arrays.asList("user-events", "global-events"));
22
23     while (true) {
24         ConsumerRecords<String, String> records = consumer.poll(100);
25
26         for (ConsumerRecord<String, String> record : records) {
27             switch (record.topic()) {
28                 case "user-events":
29                     System.out.println("Received user-events message - key: " + record.key() + " value
30                     break;
31                 case "global-events":
32                     System.out.println("Received global-events message - value: " + record.value());
33                     break;
34                 default:
35                     throw new IllegalStateException("Shouldn't be possible to get message on topic " +
36                 }
37             }
38         }
```

Step 8: Run the consumer

- Run the consumer (it will be able to consume the already produced messages as it is 'playing from the beginning')
- The consumer is in **target/consumer**



A screenshot of a terminal window with the title bar 'java'. The window displays a series of log entries from a Java application. Each entry consists of the prefix 'Received user-events message' followed by a list of key-value pairs. The keys are all 'user_id_99982' and the values are all 'some_value_125646276189312'. There are 20 such entries, indicating 20 messages were received.

```
Received user-events message - key: user_id_99982 value: some_value_125646276189312
```

< docker-compose

```
consumer_offsets,45] (kafka.coordinator.GroupMetadataManager)
kafka_1    | [2017-11-12 02:07:47,257] INFO [Group Metadata Manager on Broker 1001]: Finished loading offsets from [__consumer_
ffsets,45] in 1 milliseconds. (kafka.coordinator.GroupMetadataManager)
kafka_1    | [2017-11-12 02:07:47,158] INFO [Group Metadata Manager on Broker 1001]: Loading offsets and group metadata from [__c
consumer_offsets,48] (kafka.coordinator.GroupMetadataManager)
kafka_1    | [2017-11-12 02:07:47,259] INFO [Group Metadata Manager on Broker 1001]: Finished loading offsets from [__consumer_
ffsets,48] in 1 milliseconds. (kafka.coordinator.GroupMetadataManager)
kafka_1    | [2017-11-12 02:07:47,305] INFO [GroupCoordinator 1001]: Preparing to restabilize group test with old generation 0
kafka.coordinator.GroupCoordinator
kafka_1    | [2017-11-12 02:07:47,314] INFO [GroupCoordinator 1001]: Restabilizing group test with generation 1 (kafka.coordinator.Grou
Coordinator)
kafka_1    | [2017-11-12 02:07:47,321] INFO [GroupCoordinator 1001]: Assignment received from leader for group test for generat
on 1 (kafka.coordinator.GroupCoordinator)
```

docker-compose

Kafka and ZooKeeper

Sent message number 98200
Sent message number 98300
Sent message number 98400
Sent message number 98500
Sent message number 98600
Sent message number 98700
Sent message number 98800
Sent message number 98900
Sent message number 99000
Sent message number 99100
Sent message number 99200
Sent message number 99300
Sent message number 99400
Sent message number 99500
Sent message number 99600
Sent message number 99700
Sent message number 99800
Sent message number 99900

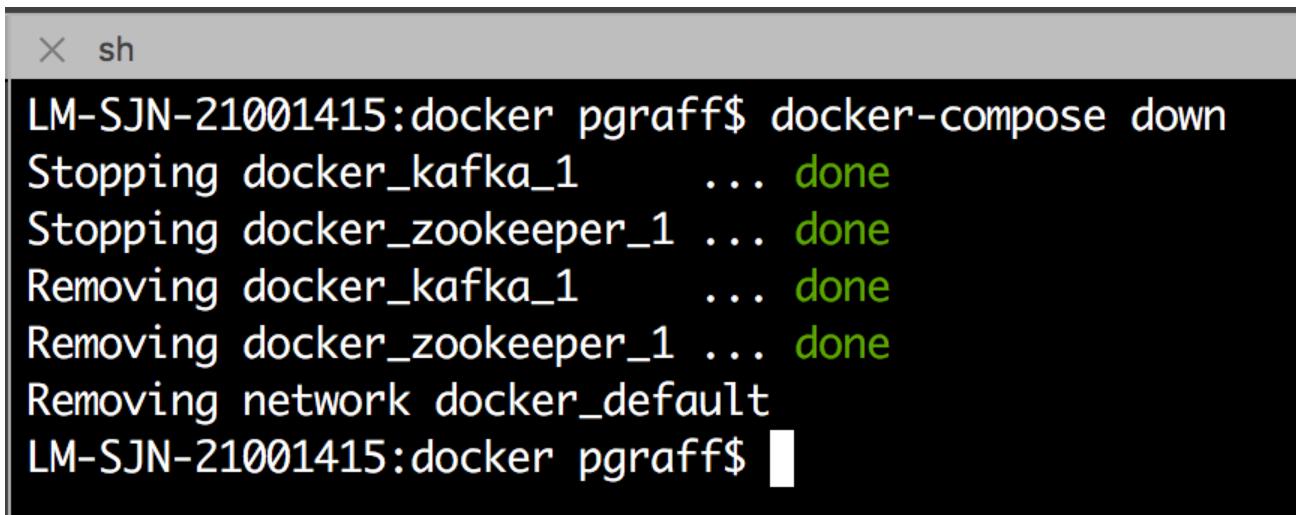
Producer

7833210
Received user-events message - key: user_id_99989 value: some_value_1261761
7834601
Received user-events message - key: user_id_99990 value: some_value_1261761
7835831
Received user-events message - key: user_id_99991 value: some_value_1261761
7837097
Received user-events message - key: user_id_99992 value: some_value_1261761
7838306
Received user-events message - key: user_id_99993 value: some_value_1261761
7839577
Received user-events message - key: user_id_99994 value: some_value_1261761
7843167
Received user-events message - key: user_id_99995 value: some_value_1261761
7844334
Received user-events message - key: user_id_99996 value: some_value_1261761
7845470
Received user-events message - key: user_id_99997 value: some_value_1261761
7846637
Received user-events message - key: user_id_99998 value: some_value_1261761
7847738
Received user-events message - key: user_id_99999 value: some_value_1261761
7851492
Received global-events message - value: plan_removed_126176143550296
Received global-events message - value: plan_added_126176145745964

Consumer

When You're Done...

- Remember to run the docker-compose down



```
sh
LM-SJN-21001415:docker pgraff$ docker-compose down
Stopping docker_kafka_1    ... done
Stopping docker_zookeeper_1 ... done
Removing docker_kafka_1    ... done
Removing docker_zookeeper_1 ... done
Removing network docker_default
LM-SJN-21001415:docker pgraff$
```

A screenshot of a terminal window titled "sh". The window shows the command "docker-compose down" being run and its execution. The output indicates that the "docker_kafka_1" and "docker_zookeeper_1" services are stopping and removing, and the "network docker_default" is also being removed. The command has completed successfully, as indicated by the green "done" status.

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
- Evaluate use of hierarchy in naming
- 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 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
 - ...

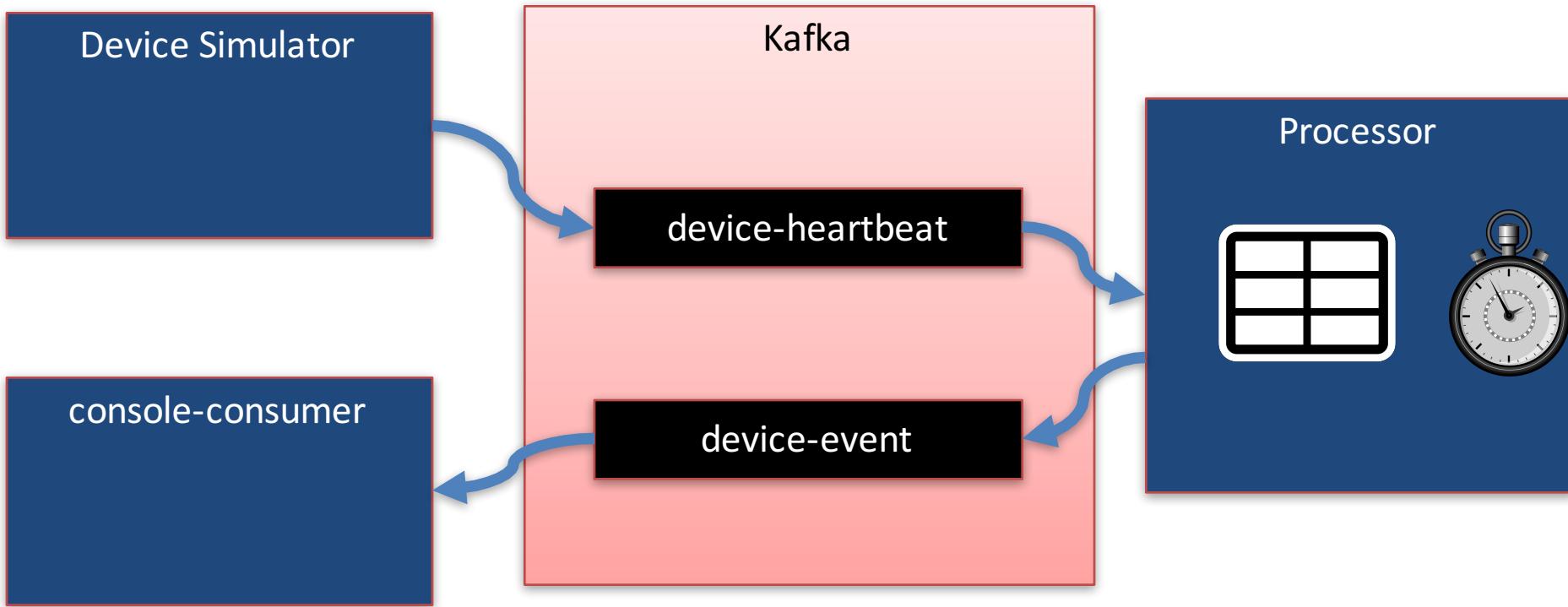
Lab: Design... The Question

- 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

```
x sh
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-topics.sh --create --zookeeper zookeeper: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 zookeeper: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

```
× sn
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 -
[INFO] Nothing to compile - all classes are up-to-date
```

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**

```
× java
LM-SJN-21001415:device-monitor pgraff$ target/device-monitor
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.
Received heartbeat from: Heart monitor 2 value: Heart monitor 2 sent heartbeat at Sat Nov 11 21:52:49 CST 2017
Received heartbeat from: Scale 1 value: Scale 1 sent heartbeat at Sat Nov 11 21:52:50 CST 2017
Received heartbeat from: Heart monitor 6 value: Heart monitor 6 sent heartbeat at Sat Nov 11 21:52:56 CST 2017
Received heartbeat from: Scale 4 value: Scale 4 sent heartbeat at Sat Nov 11 21:52:59 CST 2017
Received heartbeat from: Heart monitor 7 value: Heart monitor 7 sent heartbeat at Sat Nov 11 21:53:00 CST 2017
Received heartbeat from: Scale 1 value: Scale 1 sent heartbeat at Sat Nov 11 21:53:00 CST 2017
Received heartbeat from: Heart monitor 6 value: Heart monitor 6 sent heartbeat at Sat Nov 11 21:53:00 CST 2017
Received heartbeat from: Scale 4 value: Scale 4 sent heartbeat at Sat Nov 11 21:53:00 CST 2017
Received heartbeat from: Heart monitor 7 value: Heart monitor 7 sent heartbeat at Sat Nov 11 21:53:00 CST 2017
```

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`

```
X docker-compose
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic device-event
```

x docker-compose

```
1001]: Preparing to restabilize group console-consumer-42124 with old generation 1 (kafka.coordinator.GroupCoordinator)
kafka_1 | [2017-11-12 04:00:27,487] INFO [GroupCoordinator 1001]: Group console-consumer-42124 with generation 2 is now empty (kafka.coordinator.GroupCoordinator)
kafka_1 | [2017-11-12 04:00:27,487] INFO [GroupCoordinator 1001]: Preparing to restabilize group console-consumer-21385 with old generation 0 (kafka.coordinator.GroupCoordinator)
kafka_1 | [2017-11-12 04:00:32,767] INFO [GroupCoordinator 1001]: Assignment received from leader for group console-consumer-21385 for generation 1 (kafka.coordinator.GroupCoordinator)
```

]

x java

```
sent heartbeat at Sat Nov 11 22:04:32 CST 2017
Received heartbeat from: Scale 3 value: Scale 3 sent heartbeat at Sat Nov 11 22:04:33 CST 2017
Received heartbeat from: Heart monitor 4 value: Heart monitor 4 sent heartbeat at Sat Nov 11 22:04:33 CST 2017
Device back online: Heart monitor 4
Received heartbeat from: Scale 5 value: Scale 5 sent heartbeat at Sat Nov 11 22:04:34 CST 2017
Received heartbeat from: Heart monitor 2 value: Heart monitor 2 sent heartbeat at Sat Nov 11 22:04:34 CST 2017
Received heartbeat from: Scale 1 value: Scale 1 sent heartbeat at Sat Nov 11 22:04:35 CST 2017
Received heartbeat from: Scale 2 value: Scale 2 sent heartbeat at Sat Nov 11 22:04:42 CST 2017
Received heartbeat from: Scale 6 value: Scale 6 sent heartbeat at Sat Nov 11 22:04:42 CST 2017
```

device-monitor

NOR

x java

```
Produced heartbeat for device Scale 4
Produced heartbeat for device Heart monitor 7
Produced heartbeat for device Heart monitor 1
Produced heartbeat for device Scale 7
Produced heartbeat for device Heart monitor 2
Produced heartbeat for device Heart monitor 7
Produced heartbeat for device Heart monitor 1
Produced heartbeat for device Scale 3
Produced heartbeat for device Heart monitor 4
Produced heartbeat for device Scale 5
Produced heartbeat for device Heart monitor 2
Produced heartbeat for device Scale 1
Produced heartbeat for device Scale 2
Produced heartbeat for device Scale 6
```

]

x docker-compose

```
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic device-event
Scale 6 is back online
Heart monitor 4 offline since Sat Nov 11 22:00:03 CST 2017
Heart monitor 4 is back online
Heart monitor 1 offline since Sat Nov 11 22:01:02 CST 2017
Heart monitor 1 is back online
Heart monitor 4 offline since Sat Nov 11 22:02:18 CST 2017
Scale 7 offline since Sat Nov 11 22:02:17 CST 2017
Scale 7 is back online
Heart monitor 4 is back online
```

console-consumer

Step 8: Close it all down

- **docker-compose down**

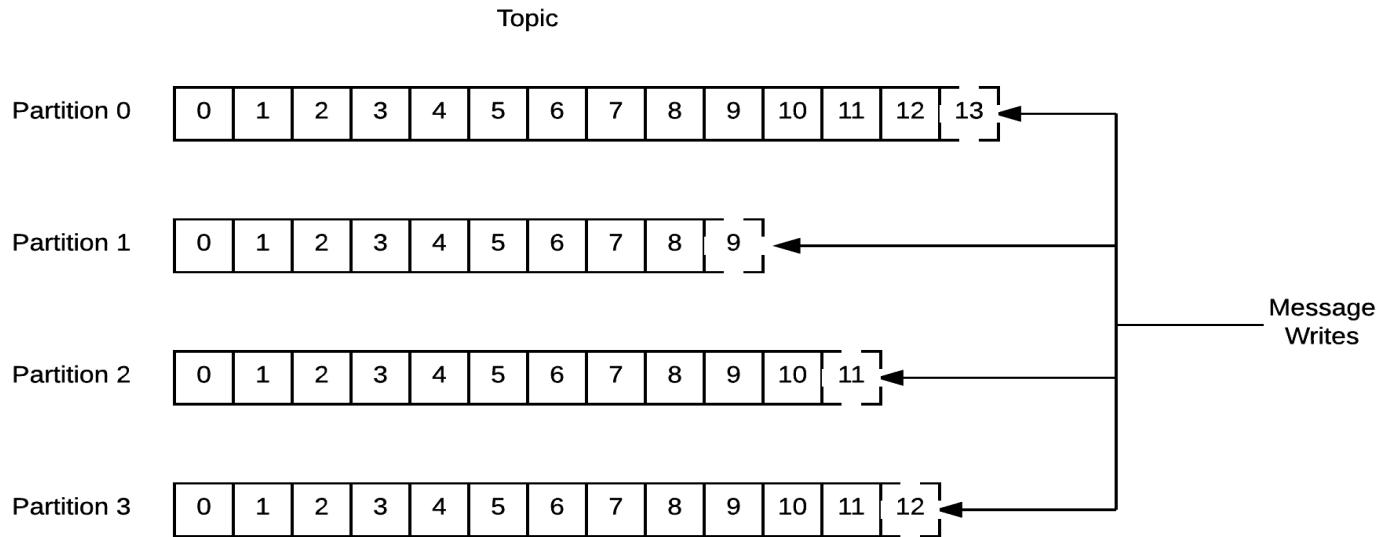
```
× sh
LM-SJN-21001415:docker pgraff$ docker-compose down
Stopping docker_kafka_1    ... done
Stopping docker_zookeeper_1 ... done
Removing docker_kafka_1    ... done
Removing docker_zookeeper_1 ... done
Removing network docker_default
LM-SJN-21001415:docker pgraff$
```

Scaling Kafka

Topics and Partitions

- Messages in Kafka are categorized into *topics*
- Think of a topic as a database table or folder in a filesystem
- Topics are broken down into a number of *partitions*
- A topic generally has multiple partitions

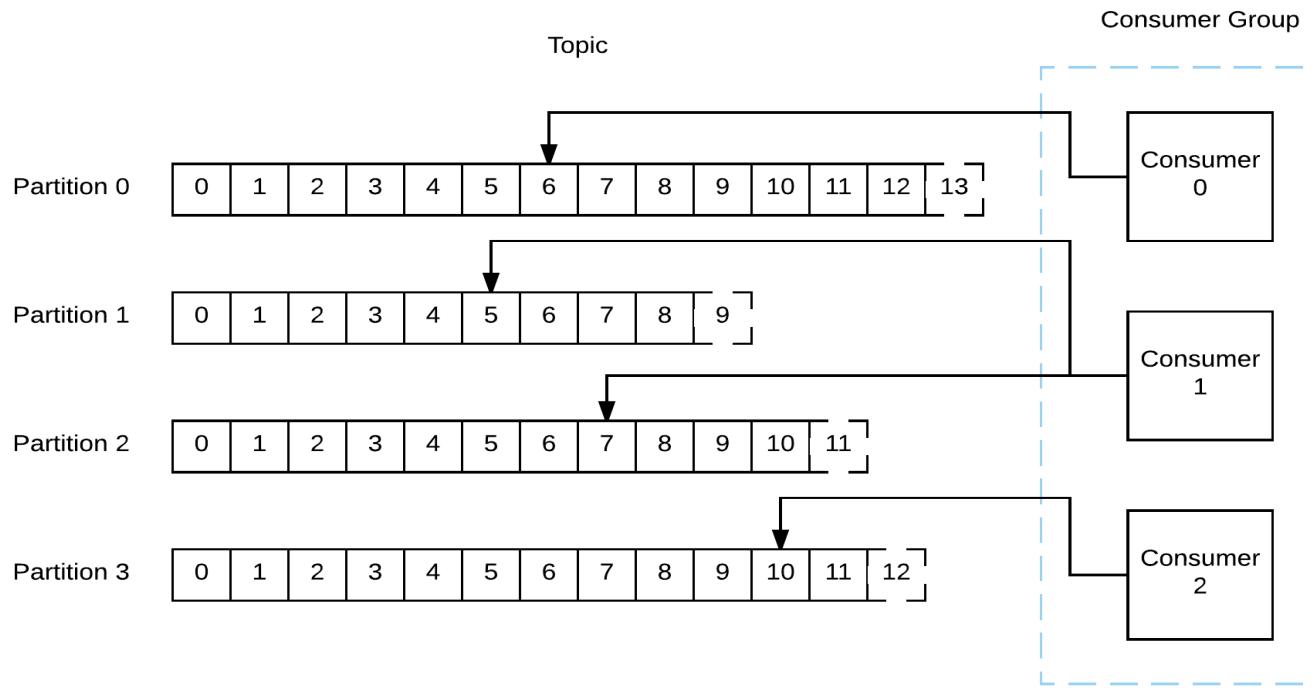
Topics and Partitions



Consumer Groups

- Consumers work as part of a *consumer group*
- One or more consumers that work together to consume a topic
- Group assures that each partition is only consumed by one member
- Mapping of a consumer to a partition is called *ownership* of the partition by the consumer

Consumer Group Illustrated



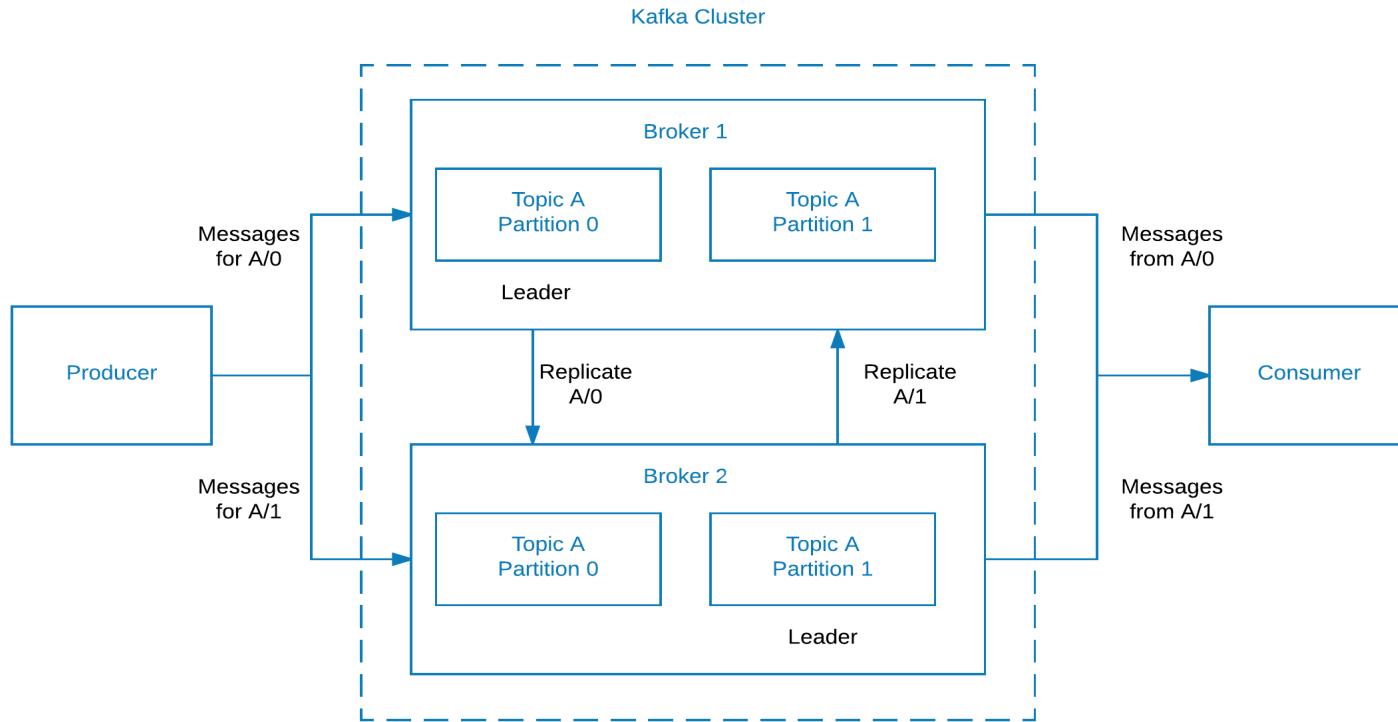
Brokers

- A single Kafka server is called a *broker*
- The broker receives messages from producers, assigns offsets to them, and commits the messages to storage on disk
- Responds to consumer fetch requests with the messages in the requested partition
- **A single broker can handle thousands of partitions and millions of messages per second**

Clusters

- Kafka brokers operate as part of a cluster
- Within a cluster of brokers, one broker acts as the ***cluster controller***
- The controller handles administrative operations such as assigning partitions to brokers and monitoring for broker failures
- A partition is owned by a single broker in the cluster known as the ***leader*** for the partition
- Another broker can take over leadership if there is a broker failure
- All consumers and producers on that partition must connect to the leader

Replication of Partitions in a Cluster



Retention

- Kafka brokers are configured with a default retention setting for topics either retaining messages for some period of time or until the topic reaches a certain size in bytes
- When limits are reached, messages are expired and deleted
- Topics can be configured with their own retention settings
 - Application metrics may only be useful for a few hours
 - Tracking topics may be useful for several days
- Topics can be *log compacted* – Kafka only retains the last message produced with a specific key
 - For example, changelog data where only the last update is useful

Compression

- In some of the largest Kafka deployments, hundreds of billions of messages per day are processed amounting to **moving hundreds of terabytes of data**
- Enabling compression allows you to reduce network utilization which can be a bottleneck when sending messages to Kafka
- Compression is an optional configuration setting in the broker and producer

Supported Compression Algorithms

- **snappy**
 - Created by Google
 - Provides good compression ratio with low CPU overhead
 - Recommended when both performance and bandwidth are a concern
- **gzip**
 - Uses more CPU and time but results in better compression ratios
 - Recommended where network bandwidth is more restricted
- **lz4**
 - Newest algorithm to Kafka
 - Faster than gzip and smaller than snappy

Compression Overview

- Multiple messages are bundled and compressed
- The compressed messages are then appended to Kafka's log file
- Compression works on a batch of messages rather than individual to take advantage of the fact that compressors work more efficiently with bigger data

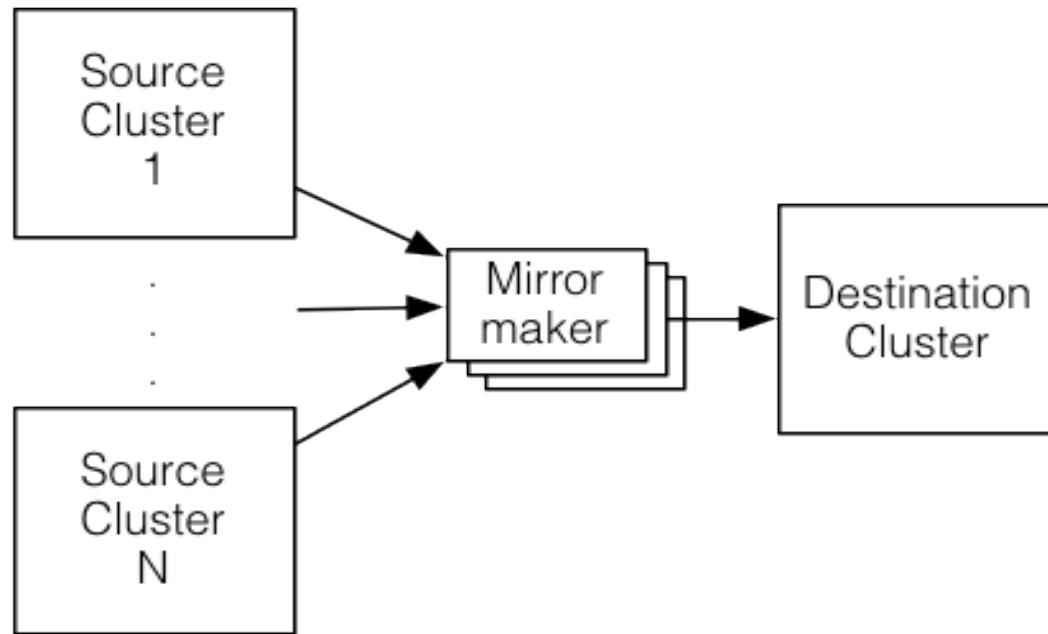
Multiple Clusters

- As your Kafka deployment grows, it can be advantageous to have multiple clusters:
 - Segregation of types of data
 - Isolation for security requirements
 - Multiple datacenters (disaster recovery)
- With multiple datacenters, messages must be copied between them
- Replication within a Kafka cluster only works within a single cluster, not between multiple clusters

Cluster Mirroring

- Kafka includes a tool called *Mirror Maker* to mirror a source Kafka cluster into a target (mirror) cluster
- Uses a Kafka consumer to consume messages from the source cluster and re-publishes the messages to the local (target) cluster using an embedded Kafka producer

Mirror Maker Illustrated



Scaling with MirrorMaker

- Designate clusters into two categories
- **Mission critical**, production
 - E.g. supporting microservices
- **Non-mission critical**
 - Consumers which are not mission critical read from another cluster
 - E.g. run analytics, feed real-time analytics and dashboards
- Mission critical cluster is not affected by readers

Multiple MirrorMakers

- Production cluster
- Analytics cluster
- Backup cluster

- Mirrormakers:
 - Production → MirrorMaker1 → Analytics
 - Production → MirrorMaker2 → Backup

Summary

- Topics and Partitions
- Producers and Consumers
- Consumer Groups
- Brokers
- Clusters
- Retention
- Compression
- Mirror Maker

Kafka and Streaming

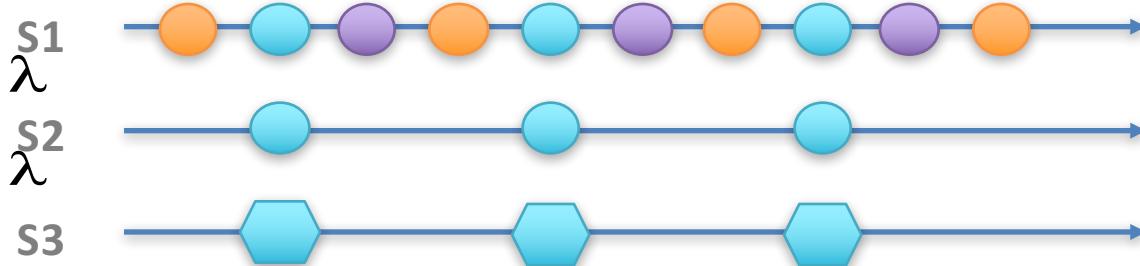
Introduction

- Kafka Streaming: An alternative for streaming to and from Kafka
 - Part of the Apache Kafka
 - Powerful
 - Highly scalable, fault-tolerant
 - Rich feature set with support for both stateless and stateful processing
 - Support for fault-tolerant local state
 - Lightweight
 - Just a library integrated in Kafka (no external dependencies)
 - No need for dedicated clusters
 - Near real-time
 - Millisecond processing latency
- Kafka and Spark / Flink

Two API's

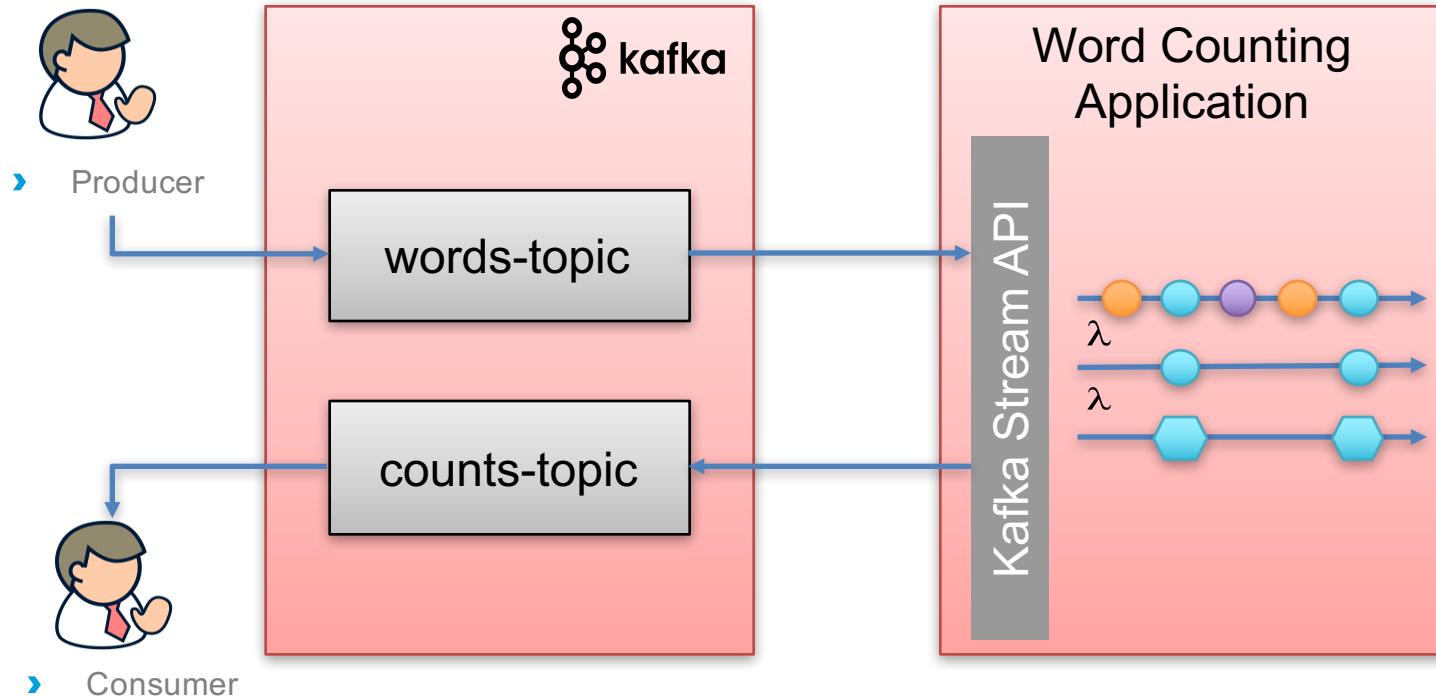
- Kafka supports a low level API that can be used to manipulate streams
- We'll focus on the Kafka Stream `DSL`
 - Higher level
 - Closer to other streaming ideas
 - Brining in some of the functional programming paradigms

What is Stream Processing?



- Programming with streams of data (even unbounded streams)
- Typical traits
 - Functional
 - Apply a dataflow or a sequence of functions to the streams
 - Combining functions provides a powerful expression language (lambda expression)
 - Reactive
 - Process incoming stream data immediately upon receipt

The Obligatory Word Count Example



Kafka Stream Classes

- StreamsConfig
 - A wrapper around a map used for configuration
- KStreamBuilder
 - The key class for setting up the stream procoessing
 - Based on the GoF Builder Pattern
 - Fluent API
 - Constructs the processing dataflow
- KafkaStreams
 - A wrapper of the actual stream

Enable Kafka Stream (Using Maven)

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
    <version>1.1.0</version>
</dependency>
```

- Kafka Streams are distributed as a simple jar file
- Simply include the jar file in your program and you'll be able to use the streaming library

Programming to Stream

```
// Configure
Properties props = new Properties();
props.put(StreamsConfig.CLIENT_ID_CONFIG, "...")
...
StreamsConfig config = new StreamsConfig(props);

// Serialization/deserialization
Serde<String> serDeser = Serdes.String();

// Create the data processing pipeline
KStreamBuilder bld = new KStreamBuilder();
bld.stream(serDeser, serDeser, "topic")
.map(...)

// Create stream
KafkaStreams sp = new KafkaStreams(bld, config);

// Start stream
kafkaStreams.start();
```

Word Count Streaming

```
StreamsConfig config = new StreamsConfig(props);
Serde<String> serde = Serdes.String();

KStreamBuilder bld = new KStreamBuilder();
bld.stream(serde, serde, "words-topic")
    .flatMapValues(text -> asList(text.split(" ")))
    .map((key, word) -> new KeyValue<>(word, word))
    .countByKey(serde, "Counts")
    .toStream()
    .map((word, count) ->
        new KeyValue<>(word, word + ":" + count))
    .to(serde, serde, "counts-topic");

KafkaStreams s= new KafkaStreams(bld, config);
s.start();
```

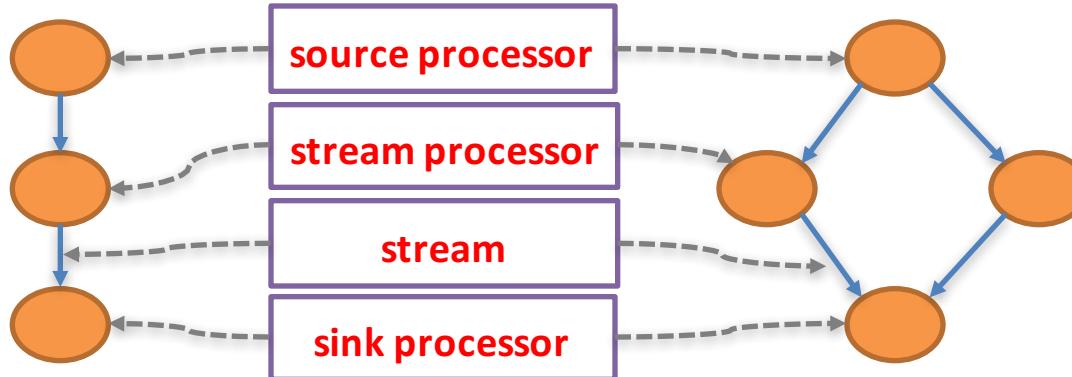
kafkacat – A Useful Tool

- kafkacat is a tool created by Magnus Edenhill
 - <https://github.com/edenhill/kafkacat>
 - Doesn't require JVM
 - Similar to 'netcat'
 - Can act as a Producer and consumer
 - Useful also to query metadata from Kafka
- To provide the word list to the application, we could simply use kafkacat instead of writing new producers and consumers
- Use as a producer
 - kafkacat –P –b myBroker –t topic1
- Use as a consumer
 - kafkacat –b myBroker –G mygroup topic1 topic 2

Installing kafkacat

- The installation of kafkacat is trivial on Mac most Linux distributions
 - Mac
 - brew install kafkacat
 - Ubuntu or Debian
 - [sudo] apt-get install kafkacat
- Our Kafka docker image is using Alpine distribution so if you want to run it in the docker image you'll have to:
 - apk add --update alpine-sdk bash python cmake
 - curl https://codeload.github.com/edenhill/kafkacat/tar.gz/master | tar xzf - && cd kafkacat-* && bash ./bootstrap.sh

Processor Topology



- The topology defines the computational logic of the data processing pipeline
- The topology typically forms a chain, but more advanced computation may be defined as a graph

Processing and Time

- In streaming time is an important design consideration
 - When working with concepts such as windowing, it is essential to establish which time to use
- Event-time
 - The point in time when an event or data record was created by the source
- Processing-time
 - The point in time when an event was processed by the stream processor
- Ingestion-time
 - The point in time when an event was stored in a topic partition by one of the Kafka brokers

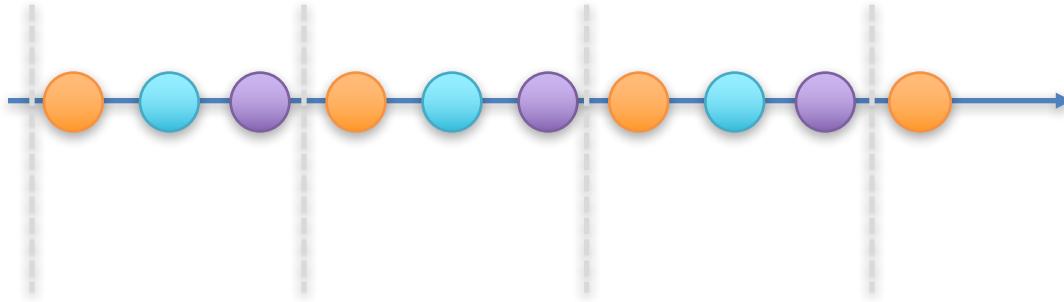
Stateful Stream Processing

- Typically we try to make our stream processing stateless
- However, for some use cases, stateful streaming may be essential
- The Kafka Stream DSL provides support for stateful processing

Stream vs. Table

- Kafka streams introduces the concept of tables, where
 - A stream can be viewed as a table: KStream → KTable
 - A table can be viewed as a stream: KTable → KStream
 - (called stream-table duality)
- Streams as tables
 - A stream can be considered a changelog of a table
 - One can convert the stream into a table by replaying the event stream
- Table as streams
 - A table can be converted into a stream by iterating over each key-value entry

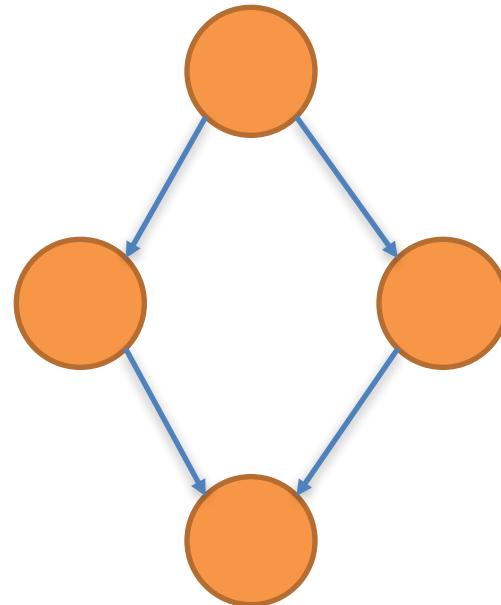
Windowing



- We may have to process data in time buckets, called windows
- Typically as part of some form of aggregation
- Windowing operations are available in the Kafka Stream DSL
- A window has a retention period to handle possible late arriving events

Parallel Stream Processing Pipelines

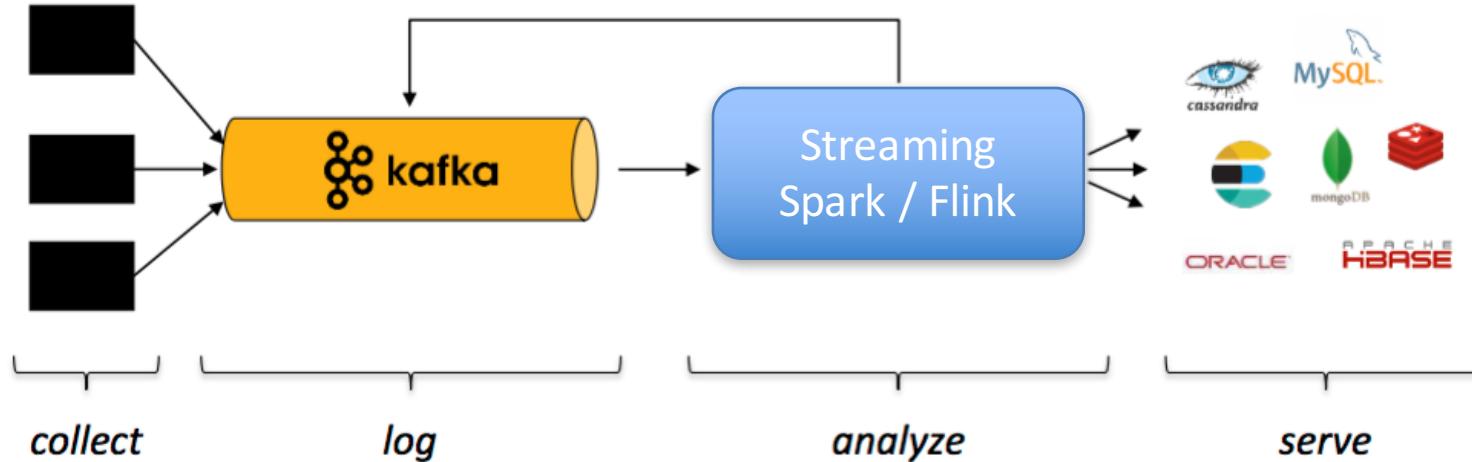
- The Kafka Stream DSL supports ways to parallelize and rendezvous processing steps
- Join
 - This operation merges two streams into a single stream
- Aggregation
 - Takes one input stream and yields a new stream by combining multiple input events into a single output event



Backpressure

- Kafka Streams don't need a backpressure mechanism
- Kafka uses a depth-first processing strategy
 - Each event consumed from Kafka goes through the complete processor topology before the next message is processed

Kafka with Spark or Flink



- Gather and backup streams
- Offer streams for consumption
- Provide stream recovery
- Analyze and correlate streams
- Create derived streams and state
- Provide these to downstream systems

Spark Streaming and Kafka

- We can set up Spark to use Kafka stream as the source of events
- Code structure:
 - A bit of configuration for Spark Streaming
 - A bit of configuration for Kafka parameters
 - Create stream from Kafka
- At this point, we have a DStream and we can do the normal Spark Streaming processing on it

Spark Streaming with Kafka – Spark Configuration

```
// Consume command line parameters
val Array(brokers, topics, interval) = args

// Create Spark configuration
val sparkConf = new SparkConf().setAppName("SparkKafka")

// Create streaming context, with batch duration in ms
val ssc = new StreamingContext(sparkConf, Duration(interval.toLong))
ssc.checkpoint("./output")
```

Spark Streaming with Kafka – Kafka Configuration

```
// Create a set of topics from a string
val topicsSet = topics.split(",").toSet

// Define Kafka parameters
val kafkaParams = Map[String, Object](
  "bootstrap.servers" -> brokers,
  "key.deserializer" -> classOf[StringDeserializer],
  "value.deserializer" -> classOf[StringDeserializer],
  "group.id" -> "use_a_separate_group_id_for_each_stream",
  "auto.offset.reset" -> "latest",
  "enable.auto.commit" -> (false: java.lang.Boolean))
```

Create a Kafka Stream in Spark and Use It

```
// Create a Kafka stream
val stream = KafkaUtils.createDirectStream[String, String](
  ssc, PreferConsistent, Subscribe[String, String](topicsSet,kafkaParams))
// Get messages - lines of text from Kafka
val lines = stream.map(consumerRecord => consumerRecord.value)
// Split lines into words
val words = lines.flatMap(_.split(" "))
// Map every word to a tuple
val wordMap = words.map(word => (word, 1))
// Count occurrences of each word
val wordCount = wordMap.reduceByKey(_ + _)
//Print the word count
wordCount.print()
```

Summary

- Kafka provides a stream processing library
- Two APIs
 - Low level API
 - Kafka Stream DSL
- The Kafka Stream DSL enables the definition of processing topologies
- Kafka streams support stateful stream processing
- Kafka has a reach support of functions and features
 - Windowing
 - Aggregation
 - Join
 - Functions (map, flatMap, etc.)
- Integration with other streaming systems: Spark, Flink, ...

Lab

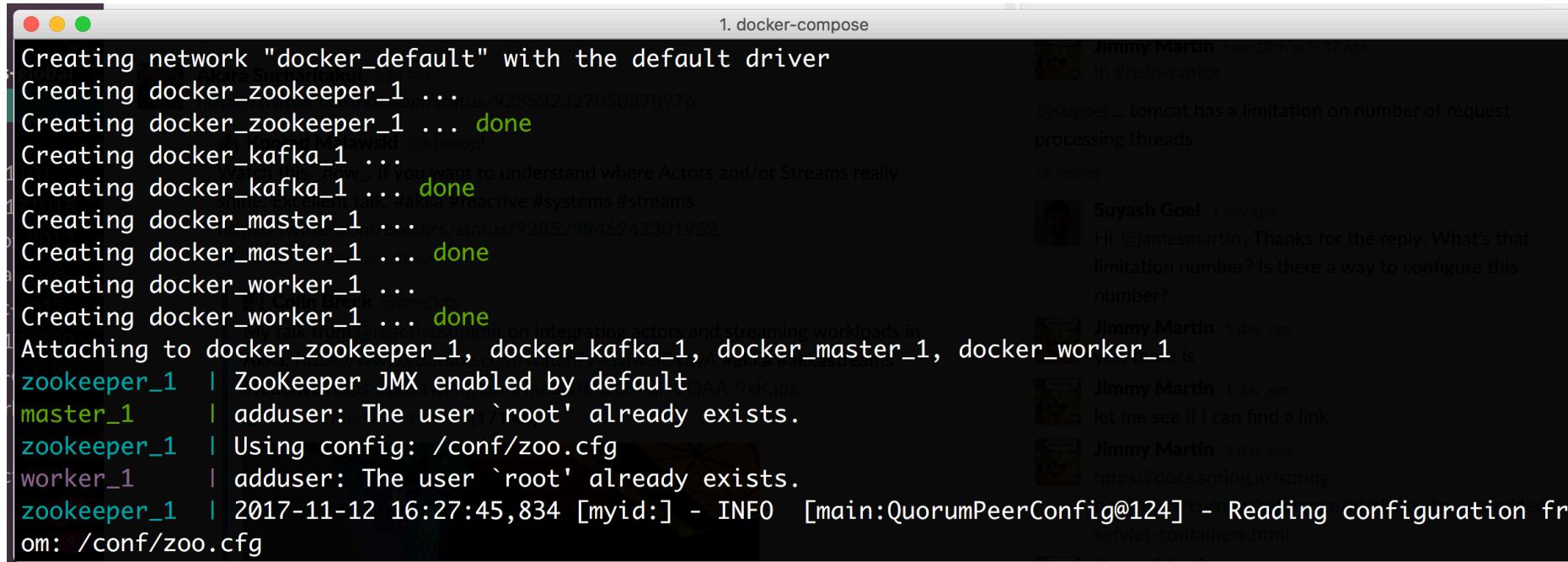
- We have 3 labs for streaming
- Lab 1: Word count using Kafka Streams
 - I will show this in slides
- Lab 2: Word count in Spark
 - I'll leave that for you to do after the class
- Lab 3: IoT example in Spark
 - I'll walk you through the code
- Example 4: IoT example implemented in Kafka Streams
 - I'll walk you through the code

Step 1: Setup Your docker.yml File

- We've done this before...
- The **docker-compose.yml** file is in the directory:
 - **kafka-lab/labs/06-Streaming/docker**
- Modify the **KAFKA_ADVERTISED_HOST_NAME** to refer to the IP of your machine
- Notice that in this docker-compose file we have more processes:
 - ZooKeeper
 - Kafka
 - Spark Master
 - Spark Worker

Step 2: Start Your Docker Processes

- Start your processes with:
docker-compose up



```
Creating network "docker_default" with the default driver
Creating docker_zookeeper_1 ... done
Creating docker_zookeeper_1 ... done
Creating docker_kafka_1 ...
Creating docker_kafka_1 ... done
Creating docker_master_1 ...
Creating docker_master_1 ... done
Creating docker_worker_1 ...
Creating docker_worker_1 ... done
Attaching to docker_zookeeper_1, docker_kafka_1, docker_master_1, docker_worker_1
zookeeper_1  | ZooKeeper JMX enabled by default
master_1     | adduser: The user `root' already exists.
zookeeper_1  | Using config: /conf/zoo.cfg
worker_1     | adduser: The user `root' already exists.
zookeeper_1  | 2017-11-12 16:27:45,834 [myid:] - INFO  [main:QuorumPeerConfig@124] - Reading configuration from: /conf/zoo.cfg
```

Jimmy Martin Nov 10 11:23 AM
in #help-raptor
@sugoi... tomcat has a limitation on number of request processing threads
16 replies

Suyash Goel 1 day ago
Hi @jamesmartin , Thanks for the reply. What's that limitation number? Is there a way to configure this number?

Jimmy Martin 1 day ago
let me see if I can find a link
<https://docs.spring.io/spring-servlet-containers.html>

Step 3: Create Some Topics

- Start a new terminal (you may want to keep this terminal to eventually shutdown the docker containers)
- For the word-count we'll have two topics:
 - stream-input
 - stream-output
- Use the docker-compose exec to create the topics:

```
x sh
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic stream-input
Created topic "stream-input".
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic stream-output
Created topic "stream-output".
LM-SJN-21001415:docker pgraff$
```

Step 4: Create a Console Consumer

- Let's create a console consumer to listen to the end result
- In a separate terminal
 - Change to the docker directory
 - Run a console consumer listening to the stream-output:
**docker-compose exec kafka
/opt/kafka_2.11-0.10.1.1/bin/kafka-console-consumer.sh
--bootstrap-server localhost:9092
--topic stream-output
--from-beginning
--property print.key=true**

Prints the key as well as the value of each message

Where Are We?

The terminal window shows the output of a docker-compose command for a Kafka service. The log message indicates the creation of a log for partition __consumer_offsets_43. The Twitter feed below discusses Docker containers and Kafka.

docker containers (kafka, zookeeper, master, worker)

console-consumer listening to stream-output

```
kafka_1    | [2017-11-12 16:38:57,810] INFO Created log for partition [__consumer_offsets,43] in /kafka/kafka-logs-0  
e43a4963505 with properties {compression.type -> producer, message.format.version -> 0.10.1-IV2, file.delete.delay.ms  
-> 60000, max.message.bytes -> 1000012, ..., offset.commit.required.acks -> 1, offset.timestamp.type -> CreateTime, min.insync.r  
eplicas -> 1, segment.jitter.ms -> 0, preallocate -> false, min.cleanable.dirty.ratio -> 0.5, index.interval.bytes ->  
4096, unclean.leader.election.enable -> true, retention.bytes -> -1, delete.retention.ms -> 86400000, cleanup.policy ->  
compact, flush.ms -> 9223372036854775807, log.cleanup.interval.ms -> 10485760, log.flush.interval.ms -> 10485760, retention.ms -> 60480  
000, message.timestamp.difference.max.ms -> 9223372036854775807, segment.index.bytes -> 10485760, flush.messages -> 9  
223372036854775807}. (kafka.log.LogManager)
```

LM-SJN-21001415:docker pgraff\$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-console-consumer.sh --boot
strap-server localhost:9092 --topic stream-output --from-beginning --property print.key=true

Colin Breck @breckcs
My talk from @reactivesummit on integrating actors and streaming workloads in
Akka: <https://www.youtube.com/watch?v=gaiwalDvayA> #akka #akkastreams

Jimmy Martin 1 day ago
yes, there is

Jimmy Martin 1 day ago
<https://docs.spring.io/spring-boot/docs/current/reference/html/howto-embedded-containers.html>

Step 5: Create a Console Producer

- We also need to be able to produce the input data
- Open a terminal and change directory to the docker directory
- Run the console-producer:

```
docker-compose exec kafka
/opt/kafka_2.11-0.10.1.1/bin/kafka-console-producer.sh
--broker-list kafka:9092
--topic stream-input
```

Where Are We?

The image displays three terminal windows and a LinkedIn post:

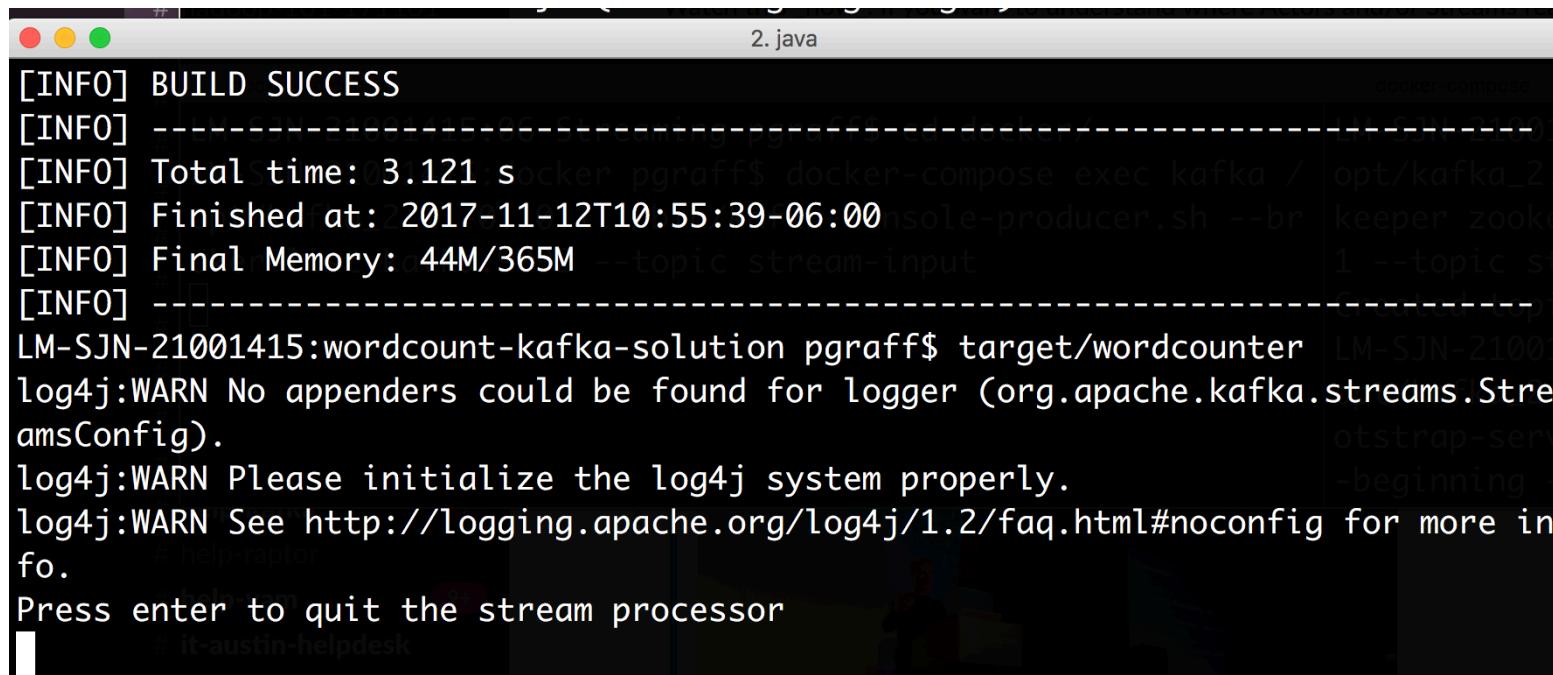
- Terminal 1 (Top Left):** Shows Kafka log output from a container named `kafka_1`. The log message indicates the creation of a log for partition `__consumer_offsets,43` with various properties like compression type, message format version, and file delete delay.
- LinkedIn Post (Top Right):** A post by Jimmy Martin and Suyash Goel titled "Understand where Actors and/or Streams really live". It includes a screenshot of a LinkedIn profile and a link to a blog post.
- Terminal 2 (Bottom Left):** Shows a user named `pgraff` executing a Kafka command to produce a stream. The command uses `docker-compose exec` to run `kafka-console-producer.sh` on a Kafka broker. The user specifies a topic named `stream-input`.
- Terminal 3 (Bottom Right):** Shows the same user `pgraff` executing a Kafka command to consume a stream. The command uses `docker-compose exec` to run `kafka-console-consumer.sh` on a Kafka broker. The user specifies a bootstrap server, a topic named `stream-output`, and a starting offset of `beginning`.

Red Overlays:

- centered text:** `docker containers (kafka, zookeeper, master, worker)`
- left terminal overlay:** `console-producer producing to stream-input`
- right terminal overlay:** `console-consumer listening to stream-output`

Step 6: Build and Run the Stream Processor

- Open a terminal and change directory to the **wordcount-kafka-solution**
- Run **mvn package** and after that **target/wordcounter**



```
# 2.java
[INFO] BUILD SUCCESS
[INFO] -----LM-SJN-21001415:00-Streaming-pgraff$ cd docker/
[INFO] Total time: 01:21 s
[INFO] Finished at: 2017-11-12T10:55:39-06:00
[INFO] Final Memory: 44M/365M --topic stream-input
[INFO] -----
LM-SJN-21001415:wordcount-kafka-solution pgraff$ target/wordcounter
log4j:WARN No appenders could be found for logger (org.apache.kafka.streams.StreamsConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Press enter to quit the stream processor
```

Where Are We?

```
for partition [__consumer_offsets,18] in /kafka/kafka-logs  
-0e43a4963505 with properties {compression.type -> produce  
r, message.format.version -> 0.10.1-IV2, file.delete.delay  
.ms -> 60000, max.message.bytes -> 1000012, min.compaction  
.lag.ms -> 60000, log.flush.interval.ms -> 10000, max.in.  
sync.replicas -> 1, segment.jitter.ms -> 0, preallocate ->  
false, min.cleanable.dirty.ratio -> 0.5, index.interval.b  
ytes -> 4096, unclean.leader.election.enable -> true, rete  
nition.bytes -> -1, delete.retention.ms -> 86400000, cleanu  
p.policy -> compact, flush.ms -> 9223372036854775807, segm
```

Twitter | Nov 9th at 1:58 AM

```
LM-SJN-21001415:06-Streaming pgraff$ cd docker/integrating actors a  
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /  
opt/kafka_2.11-0.10.1.1/bin/kafka-producer.sh --br  
oker-list kafka:9092 --topic stream-input
```

producing to
stream-input

```
-----  
LM-SJN-21001415:wordcount-kafka-solution pgraff$ target/wor  
dcounter  
log4j:WARN No appenders could be found for logger (org.apac  
he.kafka.streams.StreamThread)  
log4j:WARN Please initialize the log4j system properly.  
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html  
#noconfig for more info.  
Press enter to quit the stream processor
```

```
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /o  
pt/kafka_2.11-0.10.1.1/bin/kafka-console-consumer.sh --boot  
strap-server localhost:9092 --topic stream-output --from-be  
ginning --property print.key=true
```

listening to
stream-output

Step 7: Enter Some Text in the Producer

The screenshot shows a terminal window with two tabs: "docker-compose" and "java".

Terminal Tab:

```
false, min.cleanable.dirty.ratio -> 0.5, index.interval.bytes -> 4096, unclean.leader.election.enable -> true, retention.bytes -> -1, delete.retention.ms -> 86400000, cleanup.policy -> compact, flush.ms -> 9223372036854775807, segm
```

Java Tab:

```
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html  
#noconfig for more info.  
Press enter to quit the stream processor
```

Message History:

- Jimmy Martin and Suvash Goel 1 day ago: Jimmy Martin Nov 10th at 9:37 AM in #help-raptor
- Hi @jamesmartin, Thanks for the reply. What's that
- LM-SJN-21001415: docker pgraff\$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic stream-output
- Created topic "stream-output".
- LM-SJN-21001415: docker pgraff\$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic stream-output --from-beginning --property print.key=true
- let me see if I can find a link
- Jimmy Martin 1 day ago: https://docs.spring.io/spring-boot/docs/current/reference/html/howto-embedded-servlet-containers.html
- I think we document how to add a property to a file for spring boot
- Jimmy Martin 1 day ago: I think we document how to add a property to a file for spring boot

After some time...

1. docker-compose

```
false, min.cleanable.dirty.ratio -> 0.5, index.interval.bytes -> 4096, unclean.leader.election.enable -> true, retention.bytes -> -1, delete.retention.ms -> 86400000, cleanup.policy -> compact, flush.ms -> 9223372036854775807, segm
```

announcements-global Akara Sucharitakul 7:20 PM

x docker-compose

```
LM-SJN-21001415:06-Streaming pgraff$ cd docker/  
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-console-producer.sh --broker-list kafka:9092 --topic stream-input  
It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way—in short, the period was so far like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only
```

log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html
#noconfig for more info.
Press enter to quit the stream processor

Jimmy Martin and Suyash Goel

Jimmy Martin Nov 10th at 9:37 AM

2. docker-compose

word	count
in	1
the	14
superlative	1
degree	1
of	12
comparison	1
only	1
charles	1
dickens	1
twin	1
cities	1
no...	1
tale	1
of	13
two	1
cities...	1

processing threads
16 replies

Suyash Goel 1 day ago
Hi @jamesmartin, Thanks for the reply. What's that limitation number? Is there a way to configure this number?

Jimmy Martin 1 day ago
yes, there is

Jimmy Martin 1 day ago
let me see if I can find a link

Jimmy Martin 1 day ago
<https://docs.spring.io/spring-boot/docs/current/reference/html/howto-embedded-servlet-containers.html>

Jimmy Martin 1 day ago
I think we document how to add a property to a file in spring boot

Jimmy Martin 1 day ago

Lab/Example: Another Problem

Devices

Name ↑	Last Update
2005 CHEVY AVALANC...	2017-11-12 07:52:57
2005 Volkswagen Golf 3918	2017-11-12 09:28:18
2007 BUICK LACROSS...	2017-11-12 10:20:46
2007 CHEVY SUBURBA...	2017-11-12 10:19:07
2007 FORD EXPLORER...	2017-11-12 07:21:23
2007 SUBARU OUTBAC...	2017-11-12 11:07:45
2008 FORD EDGE	2017-11-12 09:35:54
2008 FORD EXPEDITIO...	2017-11-12 05:56:38
2008 Honda Civic 3464	2017-11-10 09:07:11

Group Search

State

Attribute ↑	Value
-------------	-------

Map

Map showing vehicle locations across a city area. Numerous vehicles are plotted with their make, model, year, and license plate number. The map includes major roads like Loop 360, FM 973, and US 290, as well as local streets and landmarks like West Lake Hills, Rollingwood, and Dryden Airport. Some vehicles are marked with red triangles (e.g., 2010 HYUNDAI ELANTRA 2909R3, 2012 FORD EXPEDITION 4029) and others with green triangles (e.g., 2013 Chevrolet Malibu 3996). Several toll stations are indicated with pink boxes labeled 'Toll 45' and 'Toll 130'. The map also shows various road signs like 'Loop 1', 'Loop 360', and 'FM 2222'.

NORTH

Reports

Devices



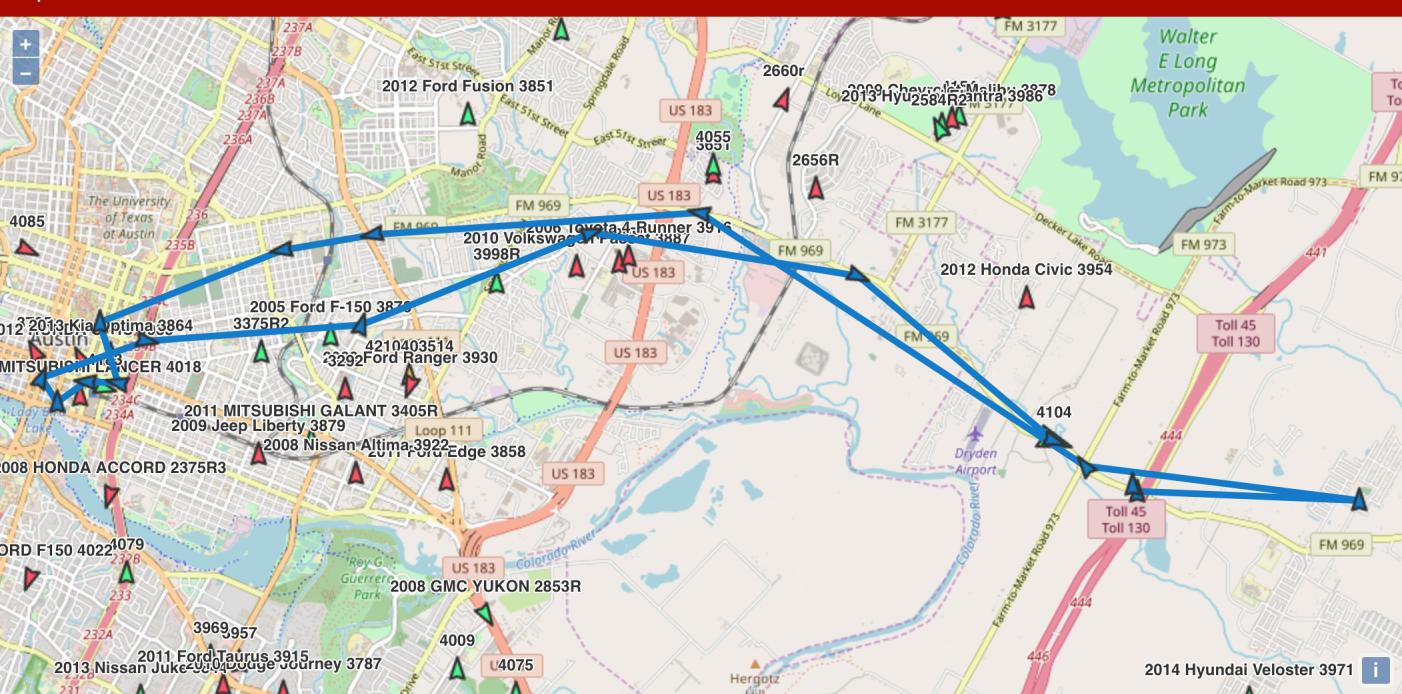
Name ↑	Last Update
2005 CHEVY AVALANC...	2017-11-12 07:52:57
2005 Volkswagen Golf 3918	2017-11-12 09:28:18
2007 BUICK LACROSS...	2017-11-12 10:20:46
2007 CHEVY SUBURBA...	2017-11-12 10:19:07
2007 FORD EXPLORER...	2017-11-12 07:21:23

Group Search

State

Attribute ↑	Value
Address	display
Battery	No
Color	BLACK
Course	E
Ignition	Yes
Ip	80.65.250.116
Latitude	30.259717
Longitude	-97.615567

Map



Device | From To | Show Clear

Event	Time	Latitude	Longitude	Speed	Address
.. moving	2017-11-12 01:22:00	30.259717	-97.615567	50.6 mph	display
.. moving	2017-11-12 01:17:00	30.278067	-97.640783	51.8 mph	display
.. moving	2017-11-12 01:12:00	30.282883	-97.675117	31.1 mph	display
.. moving	2017-11-12 01:07:00	30.272633	-97.704783	12.7 mph	display
.. moving	2017-11-12 01:02:00	30.270833	-97.732417	16.1 mph	display
.. moving	2017-11-12 00:57:00	30.266733	-97.745950	20.7 mph	display
.. moving	2017-11-12 00:52:00	30.264017	-97.743933	2.3 mph	display

Example Description

- We get a stream of vehicle locations
- Each location contains:
 - Vehicle ID
 - Data of observation
 - Speed
 - Direction
 - Latitude
 - Longitude

Problem and Plan

- We want to know where the vehicle is most likely to be parked
- Our plan
 - 1. Parse the vehicle data from an input stream
 - 2. Filter out the only those locates where the vehicle was at rest
 - 3. Convert the vehicle locate into keys that we can accumulate. The key would be based on
 - Vehicle ID
 - Geo location
 - 4. Group the locates together by key and count them
 - 5. Stream the frequent parking count to another stream

Input File: vehicle_1_to_1000.tsv

678	120	2016-08-16	16:31:21	30	162	-97.796833333334	30.466516666666666/
679	120	2016-08-16	16:31:31	5	159	-97.79646666666666	30.465716666666665
680	113	2016-08-16	16:31:37	5	159	-97.79645	30.465733333333333
681	120	2016-08-16	16:31:41	5	159	-97.79646666666666	30.465716666666665
682	120	2016-08-16	16:31:51	5	159	-97.79646666666666	30.465716666666665
683	120	2016-08-16	16:32:01	14	154	-97.7963833333334	30.465566666666668
684	120	2016-08-16	16:32:11	22	163	-97.79596666666667	30.46458333333334
685	120	2016-08-16	16:32:17	28.4	159	-97.795695	30.463865
686	120	2016-08-16	16:32:21	26	159	-97.7954833333334	30.463366666666666
687	107	2016-08-16	16:32:29	25	159	-97.7950833333334	30.4624
688	120	2016-08-16	16:32:31	24	157	-97.795	30.462216666666666
689	120	2016-08-16	16:32:41	7	160	-97.7946666666667	30.4615
690	120	2016-08-16	16:32:51	2	158	-97.7946333333334	30.46143333333332
691	113	2016-08-16	16:32:52	2.1	160	-97.79464166666666	30.46143333333332
692	120	2016-08-16	16:33:01	2	158	-97.7946333333334	30.46143333333332
693	111	2016-08-16	16:33:08	2	161	-97.7946833333334	30.46145
694	120	2016-08-16	16:33:11	2	159	07.7946333333334	30.4614333333332

Message Pump: Simulator

- We've created a message pump application to simulate the vehicle input
- You can find this implementation in the directory:
kafka-lab/labs/06-Streaming/iot-kafka-solution/gps-pump
- The implementation reads the TSV file and produces messages into a topic called:
gps-locations
- In the real world, we would have to write some edge servers that receives the data from the vehicles and publish the topic into the topic

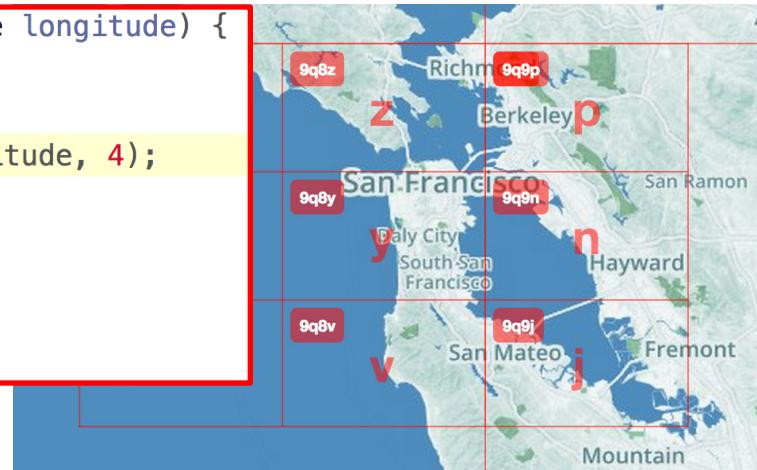
Event Processor

- We've created an event processor based on kafka-streaming (and another on Spark that we'll look at later)
- You can find this implementation in the directory:
kafka-lab/labs/06-Streaming/iot-kafka-solution/processor
- This processor uses kafka-streams to read the events from the topic **gps-locations** and produces messages on the topic **frequent-parking**
 - The messages produced:
 - Key: Vehicle and location key based on GeoHash
 - Value: parking count

GeoHash

- We build a key based on the vehicle id and a GeoHash
- GeoHash allows us to focus on an area instead of absolute locates

```
9  public LocationKey(String id, double latitude, double longitude) {  
10     this.objectId = id;  
11     try {  
12         this.geoHash = GeoHash.encodeHash(longitude, latitude, 4);  
13     }  
14     catch (Exception e) {  
15         System.out.println(e.getMessage());  
16     }  
17 }
```



Kafka Stream Solution

```
1 source
2   .map( (key,value) -> new KeyValue<>(key, value.split("\t")))
3   .filter((key,value)--> value.length > 5 && Double.parseDouble(value[2]) == 0)
4   .map((key,value) ->
5     new KeyValue<>(
6       new LocationKey(value[0],
7         Double.parseDouble(value[4]),
8         Double.parseDouble(value[5])).toString(),
9         value[0]))
10  .groupByKey()
11  .count("parking")
12  .mapValues((value) -> Long.toString(value))
13  .to("frequent-parking");
```

Run Solution in Kafka

1. Modify the docker-compose file to use your (which you may already have done in the word-count exercise)
2. Create the topics:
 - gps-locations
 - frequent-parking
3. Run a kafka-consumer to listen to the frequent-parking topic (with the show key property)
4. Build and run the message pump
5. Build and run the processor
6. Wait to see the parking counts...

Creating the Topics and Start the Console Consumer

```
1. docker-compose
policy -> compact, flush.ms -> 9223372036854775807, segment.ms -> 604800000, segment.bytes -> 104857600, retention.ms -> 604800000, message.timestamp.difference.max.ms -> 9223372036854775807, segment.index.bytes (-> 10485760, flush.messages -> 9223372036854775807}. (kafka.log.LogManager)
kafka_1 | [2017-11-12 16:38:57,904] INFO Partition [<__consumer_offsets,38] on broker 1001: No checkpointed highwatermark is found for partition [<__consumer_offsets,38] (kafka.cluster.Partition)
kafka_1 | [2017-11-12 16:38:57,918] INFO Completed load of log __consumer_offsets-35 with 1 log segments and log end offset 0 in 0 ms (kafka.log.Log)
kafka_1 | [2017-11-12 16:38:57,919] INFO Created log for partition [<__consumer_offsets,35] in /kafka/kafka-logs-0e43a4963505 with properties {compression.type -> producer, message.format.version -> 0.10.1-IV2, file.delete.delay.ms -> 60000, max.message.bytes -> 1000012, min.compaction.lag.ms -> 0, message.timestamp.type -> CreateTime, min.insync.r

x docker-compose
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic gps-locations
Created topic "gps-locations".
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic frequent-parking
Created topic "frequent-parking".
LM-SJN-21001415:docker pgraff$ docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic frequent-parking --from-beginning --property print.key=true


```

Build and Start the Message Pump

```
sent.22 2010-08-15 12:11:50 +11:59 for each((key,value) > System.out.println(key));
^CLM-SJN-21001415:gps-pump pgraff$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building gps-message-pump 1.0-SNAPSHOT
[INFO] -----
[INFO] -----
[INFO] [WARNING] [INFO] --- maven-assembly-plugin:2.6:single (make-assembly) @ gps-message-pump --- platform dependent!
[INFO] Building jar: /Users/pgraff/kafka/kafka-lab/labs/06-Streaming/iot-kafka-solu
mp-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] [INFO] --- really-executable-jar-maven-plugin:1.1.0:really-executable-jar (default)
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.559 s
[INFO] Finished at: 2017-11-12T11:51:04-06:00
[INFO] Final Memory: 26M/313M
[INFO] -----
LM-SJN-21001415:gps-pump pgraff$ 63 KafkaStreams streams = new KafkaStreams(b
LM-SJN-21001415:gps-pump pgraff$ target/gps-message-pump ../vehicle_1_1000.tsv
```

Output

```
x java
Sent: 120 2016-08-16 15:54:03 0 0 -97.83058333333334 30.44643333333335
Sent: 120 2016-08-16 15:54:12 0 0 -97.83058333333334 30.44643333333335
Sent: 120 2016-08-16 15:54:22 0 0 -97.83058333333334 30.44643333333335
Sent: 120 2016-08-16 15:54:32 0 0 -97.83058333333334 30.44643333333335
Sent: 120 2016-08-16 15:54:42 0 0 -97.83058333333334 30.44643333333335
Sent: 120 2016-08-16 15:54:52 0 0 -97.83058333333334 30.44643333333335
Sent: 120 2016-08-16 15:55:02 0 0 -97.83058333333334 30.44643333333335
Sent: 120 2016-08-16 15:55:12 0 0 -97.83058333333334 30.44643333333335
Sent: 113 2016-08-16 15:55:52 3.5 // 325 final -97.83055833333333 30.446461666666668
Sent: 120 2016-08-16 15:55:56 0 0 -97.83058333333334 30.44643333333335
Sent: 107 2016-08-16 15:55:58 0 0 -97.83049166666666 30.446375
Sent: 120 2016-08-16 15:56:06 0 // let's hook up the busines 30.44643333333335
Sent: 120 2016-08-16 15:56:16 0 Kafka -97.83058333333334 30.44643333333335
64 // and then... we can start the stream processing
```

Build and Start the Processor

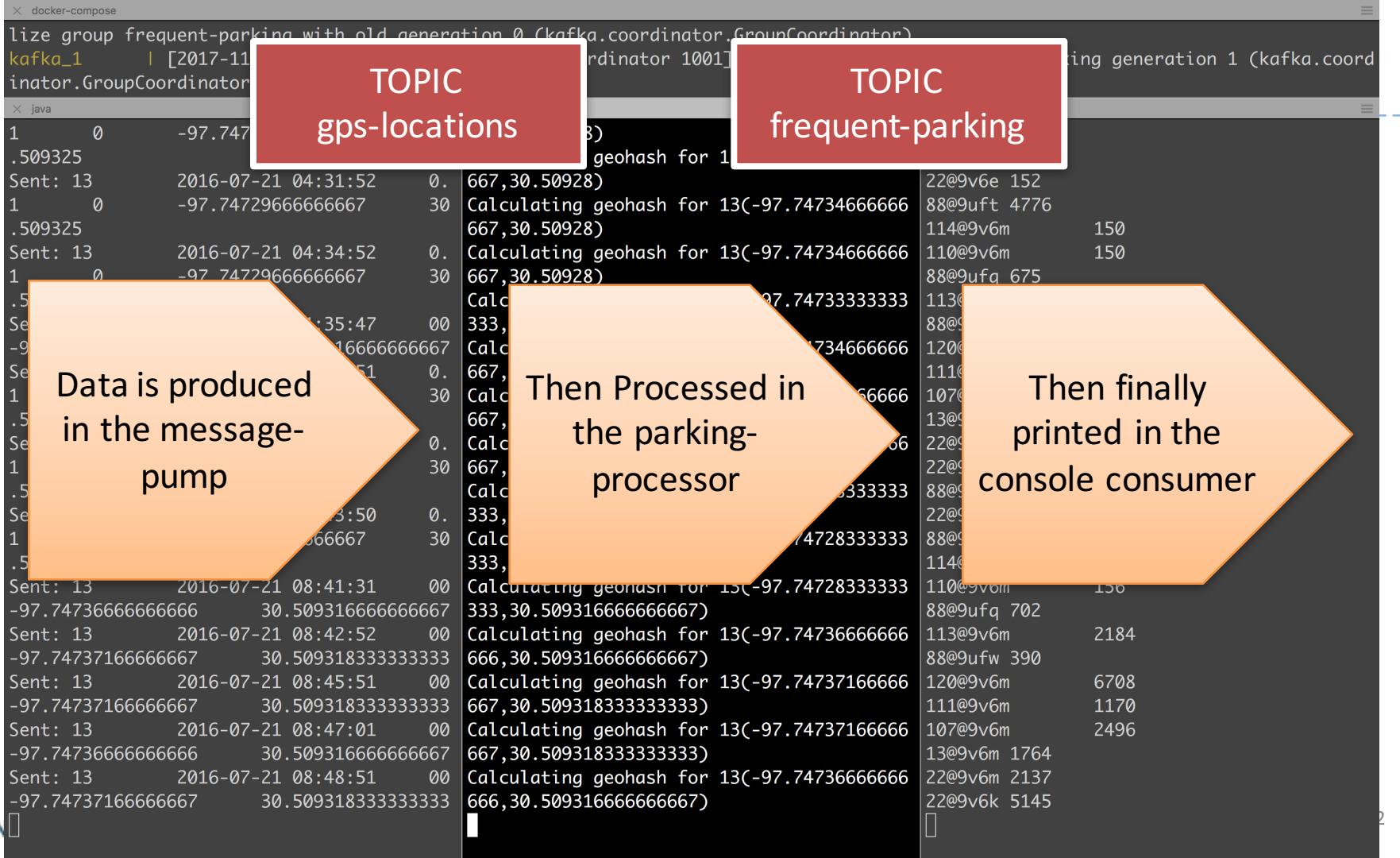
```
x sh
LM-SJN-21001415:processor pgraff$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building iot-example 1.0-SNAPSHOT
[INFO]
```

```
x sh
[INFO]
[INFO] --- really-executable-jar-maven-plugin:1.1.0:really-executable-jar (default) @ iot-example ---
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.037 s
[INFO] Finished at: 2017-11-12T11:55:59-06:00
[INFO] Final Memory: 50M/368M
[INFO] -----
LM-SJN-21001415:processor pgraff$ target/parking-processor
```

Output

```
x java
Calculating geohash for 88(-97.4026666666666,27.76536666666666)
Calculating geohash for 88(-97.4026666666666,27.76536666666666)
Calculating geohash for 88(-97.4026666666666,27.76546666666667)
Calculating geohash for 88(-97.4026666666666,27.76546666666667)
Calculating geohash for 22(-97.8934666666667,30.36988666666666)
Calculating geohash for 22(-97.8934666666667,30.36988333333334)
Calculating geohash for 22(-97.9058,30.37845)
Calculating geohash for 22(-97.9058,30.37841666666666)
Calculating geohash for 88(-97.40265,27.76543333333334)
Calculating geohash for 88(-97.40266333333334,27.76544666666666)
Calculating geohash for 88(-97.4026333333333,27.7654666666667)
Calculating geohash for 88(-97.40264833333333,27.765475)
Calculating geohash for 88(-97.4027,27.7654)
Calculating geohash for 88(-97.402715,27.76541)
Calculating geohash for 88(-97.4026666666666,27.76545)
Calculating geohash for 88(-97.40267333333334,27.765455));
Calculating geohash for 88(-97.40267333333334,27.765455)
```

x docker-compose	
120@9v6m	5934
111@9v6m	1035
107@9v6m	2208
13@9v6m	1575
22@9v6m	1903
22@9v6k	4588
88@9v6m	2464
22@9v6e	146
88@9uft	4587
114@9v6m	144
110@9v6m	144
88@9ufq	648
113@9v6m	2016
88@9ufw	360
120@9v6m	6192
111@9v6m	1080
107@9v6m	2304
13@9v6m	1638
22@9v6m	1981
22@9v6k	4773



Kafka Administration and Integrations

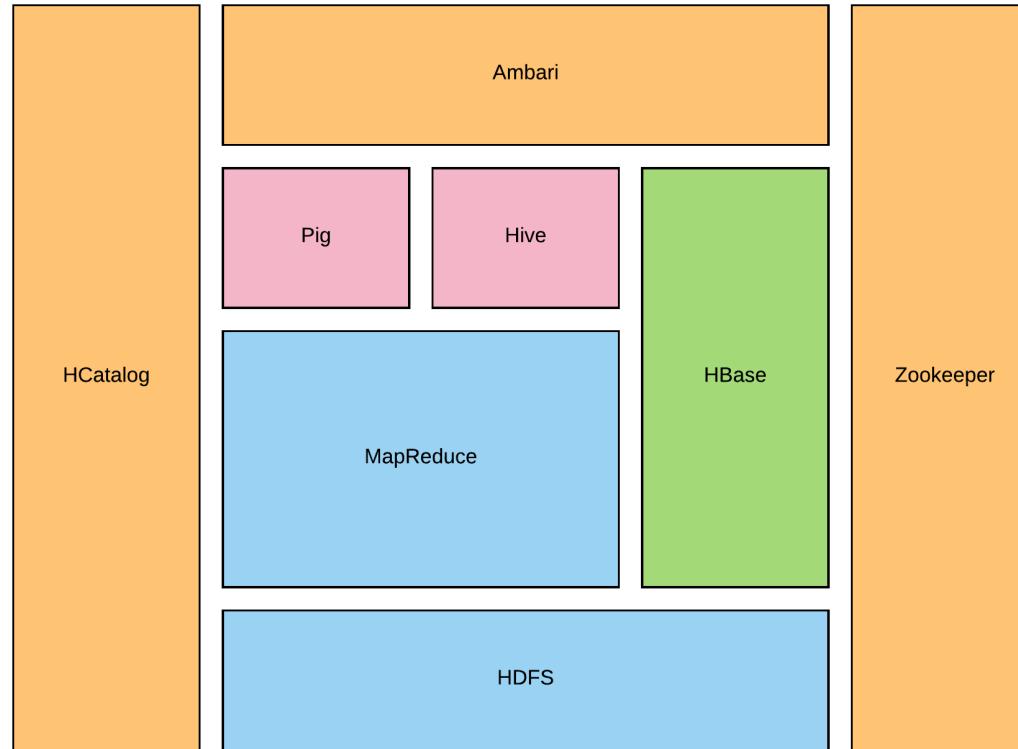
Kafka Integration with Hadoop

- Another common use case is consuming Kafka data directly into **Hadoop**
- **Hadoop** is a large scale distributed batch processing framework which parallelizes data processing across many servers
- Its strength is in the ability to scale across thousands of commodity servers that don't share memory or disk space

Major Functional Components

- **Hadoop** can be thought of as an ecosystem comprised of many different components that work together to create a single platform
- Two key functional components:
 - **HDFS** – a scalable file system that distributes and stores data across all machines in Hadoop cluster
 - **MapReduce** – the system used to efficiently process the large amount of data stored in HDFS. Large data is divided into smaller and smaller tasks in a pipeline.

Hadoop Cluster Components



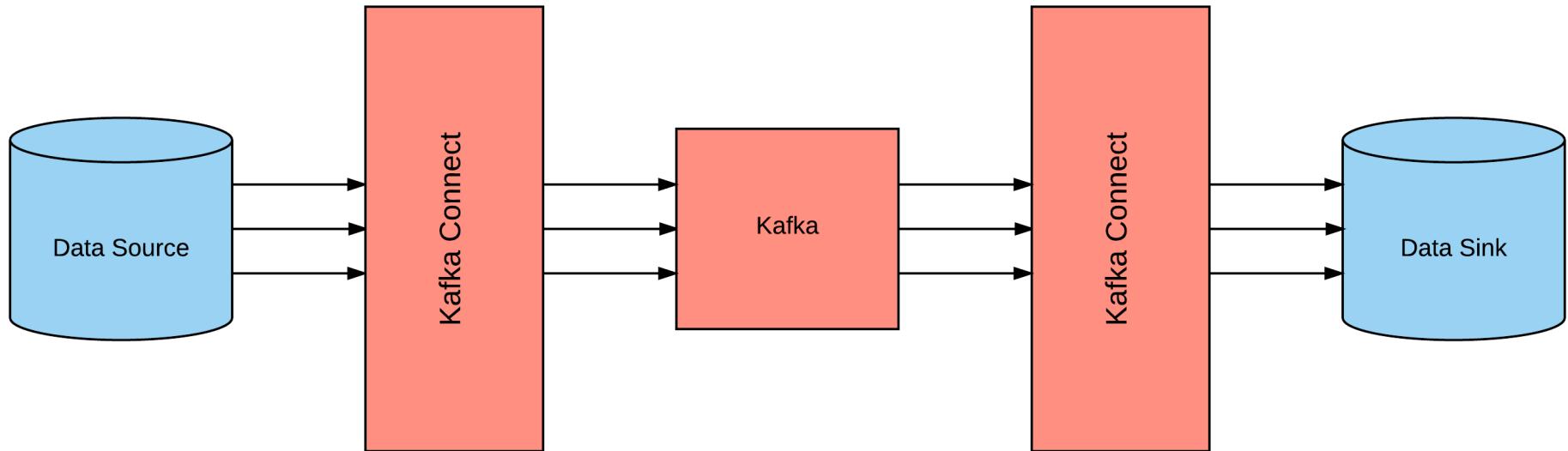
Kafka Integration with Hadoop

- In earlier versions of Kafka (0.8), integration with Hadoop could be achieved with an extension directly in the Kafka code base
- This was great news for Hadoop users but a lot of code would need to be written for other integrations
 - S3
 - HBase
 - JDBC
 - Cassandra

Kafka Connect and Confluent

- **Confluent** is a startup that was founded in 2014 by the creators of Kafka
- **Kafka Connect** is a feature in Apache Kafka 0.9+ developed by **Confluent** that makes building and managing stream data pipelines easier
- **Kafka Connect** abstracts away common problems that every connector to Kafka needs to solve such as
 - fault tolerance
 - partitioning
 - offset management
 - monitoring

High Level Look at Kafka Connect



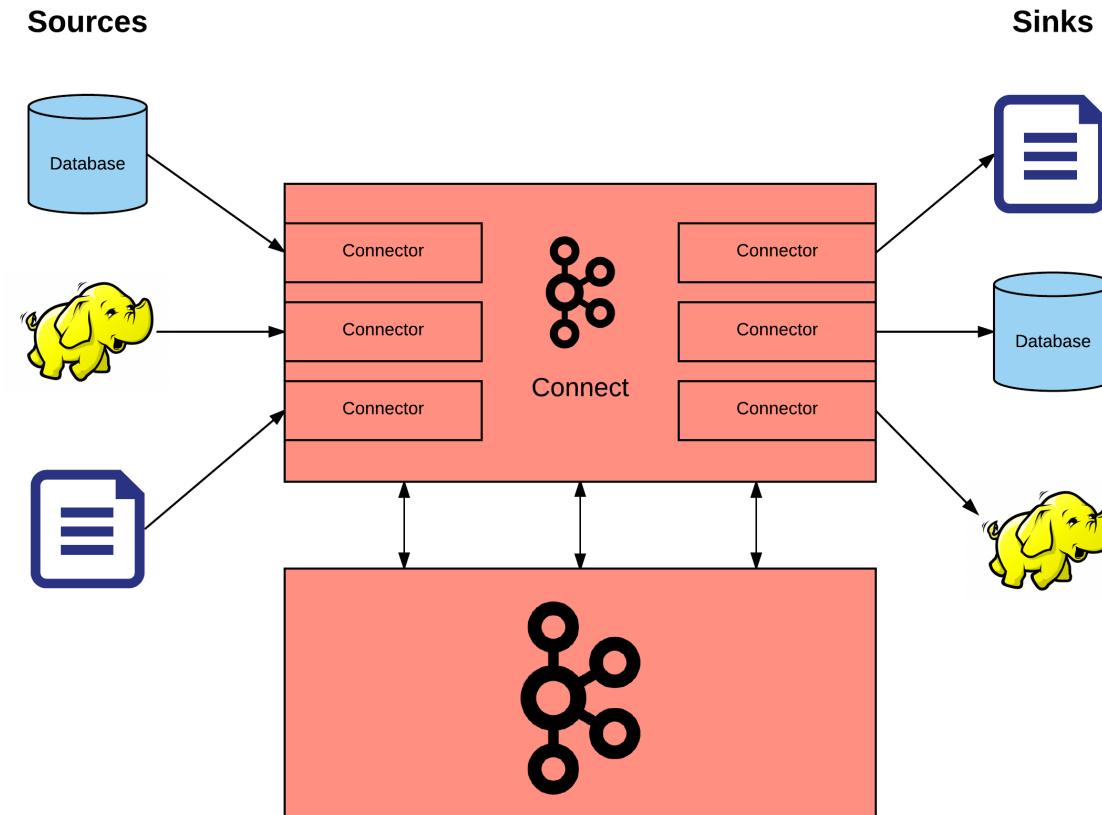
Certified Connectors

- Confluent maintains the **HDFS (Sink) connector** using the Kafka Connect framework
- Other certified connectors from Confluent include
 - JDBC (Source)
 - Elastic Search (Sink)
 - Attunity (Source)
 - Couchbase (Source)
 - GoldenGate (Source)
 - JustOne (Sink)
 - Syncsort DMX (Source)
 - Syncsort DMX (Sink)
 - Vertica (Source)
 - Vertica (Sink)

Additional Community Connectors

- Other notable community connectors utilizing Kafka Connect
 - Cassandra (Source and Sink)
 - Elastic Search (Sink)
 - FTP (Source)
 - HBase (Sink)
 - InfluxDB (Sink)
 - Bloomberg Ticker (Source)
 - MongoDB (Source)
 - Mixpanel (Source)
 - MQTT (Source)
 - Solr (Sink and Source)
 - And more!

Another Kafka Connect View



Kafka Administration

- There are a number of useful operations that are not automated and must be triggered using one of the tools that ship with Kafka
 - Adding/deleting topics
 - Modifying topics
 - Graceful shutdown
 - Balancing leadership
 - Checking consumer position
 - Mirroring data between clusters
 - Expanding a cluster
 - Decommissioning brokers
 - Increasing replication factor

Adding and Deleting Topics

- Topics can be added manually or you can have them be created automatically when data is first published to a non-existent topic
- Topics are added and removed with the topic tool
 - `kafka-topics.sh --zookeeper zk_host:port --create --topic my_topic_name --partitions 20 --replication-factor 3 --config x=y`
 - `kafka-topics.sh --zookeeper zk_host:port --delete --topic my_topic_name`

Modifying Topics

- Changing the configuration or partitioning of a topics can be done with the same topic tool
 - `kafka-topics.sh --zookeeper zk_host:port/chroot --alter --topic my_topic_name --partitions 40`
- The partition count controls how many logs the topic will be sharded into

Graceful Shutdown

- A Kafka cluster will automatically detect any broker shutdown or failure and elect new leaders for the partitions on that machine
- When a broker is brought down intentionally for configuration changes, Kafka supports a more graceful shutdown that comes with two optimizations
 - All logs are synced to disk to avoid needing to validate checksums when restarting. This speeds up intentional restarts
 - Partitions the server is the leader for will be migrated to other replicas prior to shutting down. This will make leadership transfer faster and minimize the time each partition is unavailable to a few milliseconds

Graceful Shutdown Continued

- To take advantage of this graceful shutdown, the following setting is required
 - `controlled.shutdown.enable=true`

Checking Consumer Position

- The following command will show the position of all consumers in a consumer group and how far behind the end of the log they are
 - `kafka-run-class.sh kafka.tools.ConsumerOffsetChecker --zookeeper localhost:2181 --group test`

Mirroring Data Between Clusters

- Replicating data *between* Kafka clusters is known as **mirroring** which is not to be confused with the replication that happens amongst the nodes in single cluster
- Kafka comes with a tool for this purpose called `kafka-mirror-maker`
- A common use case is to provide a replica in another datacenter

Expanding a Cluster

- To add servers to a Kafka cluster, just assign them a unique broker id and start
- The new servers will not automatically be assigned any data partitions so unless partitions are moved to them they won't be doing any work until new topics are created
- Most of the time when you add machines to the cluster you will migrate some existing data with the `kafka-reassign-partitions` tool
- Check out the documentation for examples and usage!

Decommissioning Brokers

- The partition reassignment tool cannot automatically generate a reassignment plan for decommissioning brokers
- The admin must come up with a plan to move the replica for all partitions hosted on the broker to be decommissioned to the rest of the brokers
 - Can be **tedious!**
- Tooling support for this task is planned in the future

Increasing Replication Factor

- To increase the replication factor of an existing partition, the extra replicas must be specified in a JSON file
- The following example increases the replication factor of partition 0 of topic test from 1 to 3
 - increase-replication-factor.json:

```
{"version":1,"partitions":[{"topic":"test","partition":0,"replicas":[5,6,7]}]}
```
 - `kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file increase-replication-factor.json --execute`

Monitoring

- Kafka uses **Yammer Metrics** for metrics reporting in both the server and client
- You can use jconsole and attach to a running Kafka client or server to see all the metrics available with JMX

Security

- In release 0.9, Kafka added a number of security related features to Kafka
 - Authentication of connection to brokers from clients using SSL or SASL (Kerberos)
 - Authentication of connections from brokers to Zookeeper
 - Encryption of data transferred between brokers and clients, between brokers, or between brokers and tools using SSL
 - Authorization of read/write operations by clients
 - Authorization is pluggable and can be integrated with external authorization services

Summary

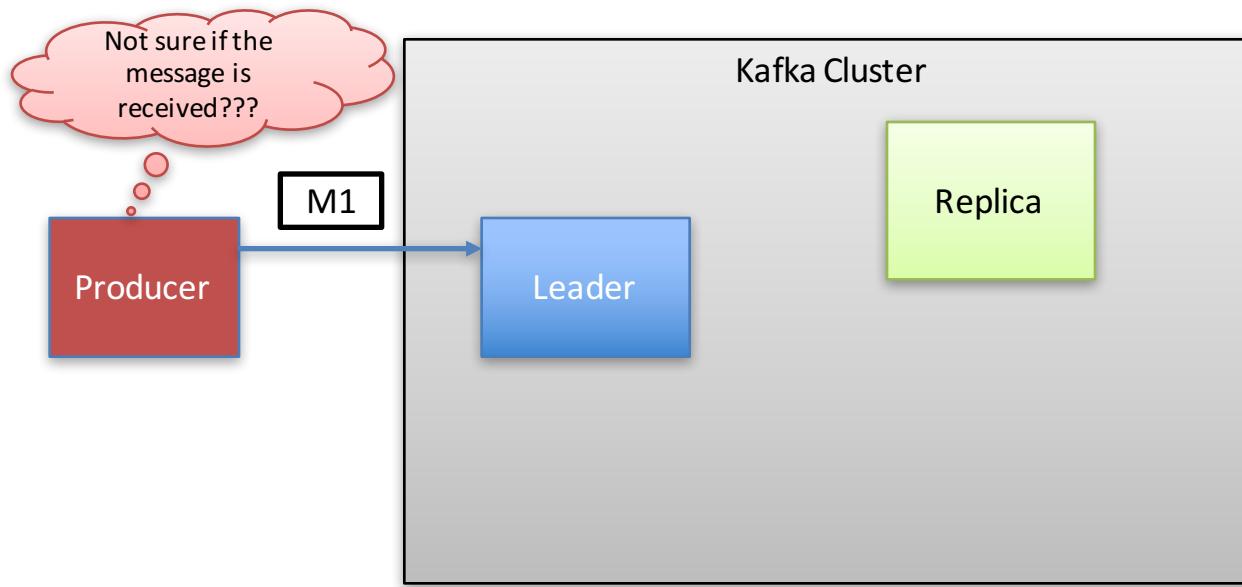
- Kafka Integrations
 - Apache Hadoop
- Kafka Connect
- Administration
 - Command line tools for modifying topics, graceful shutdown, mirroring data, checking consumer position, etc.
 - Monitoring
 - Security

Kafka Delivery Guarantees

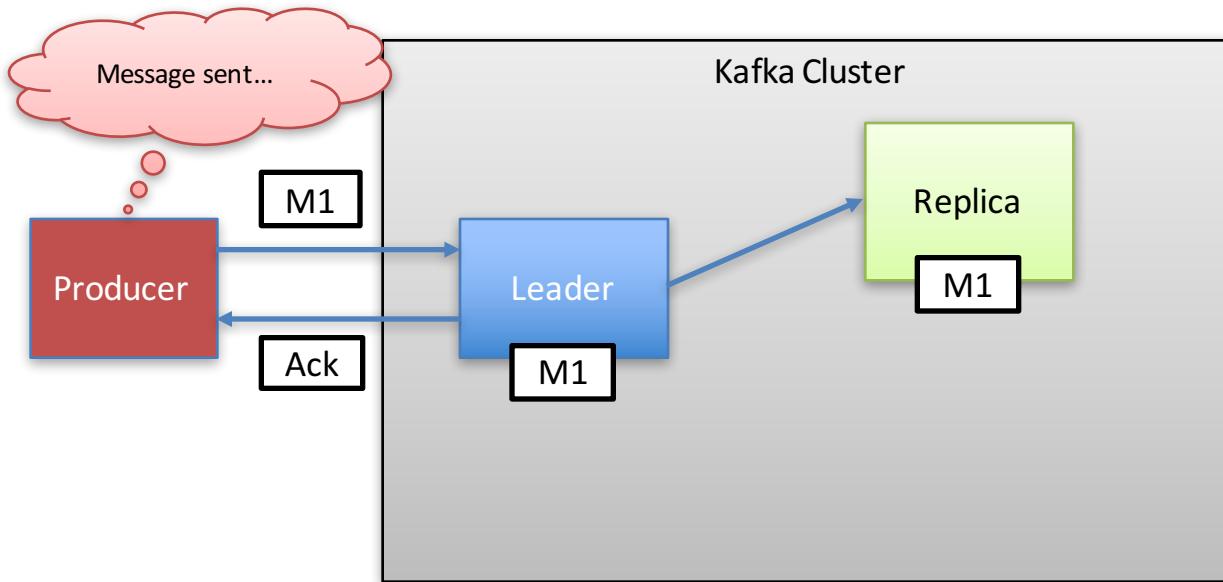
Introduction

- Kafka's delivery model leaves potential gaps in typical enterprise deployment scenarios
- Naïve programming may lead to
 - Messages being processed multiple times
 - Messages not being processed
- We will look at a few different scenarios where such failures may occur
- We'll also look at some technique that can be used to minimize or avoid the failures
 - Idempotent services
 - Kafka's new exactly once processing guarantee

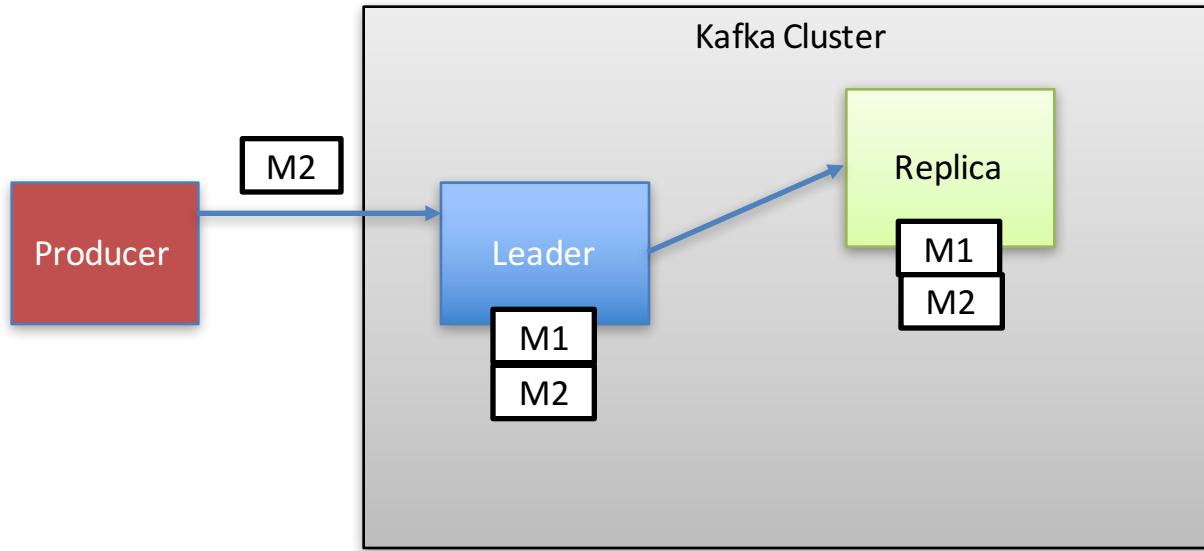
Message Duplication: 1. Send the message



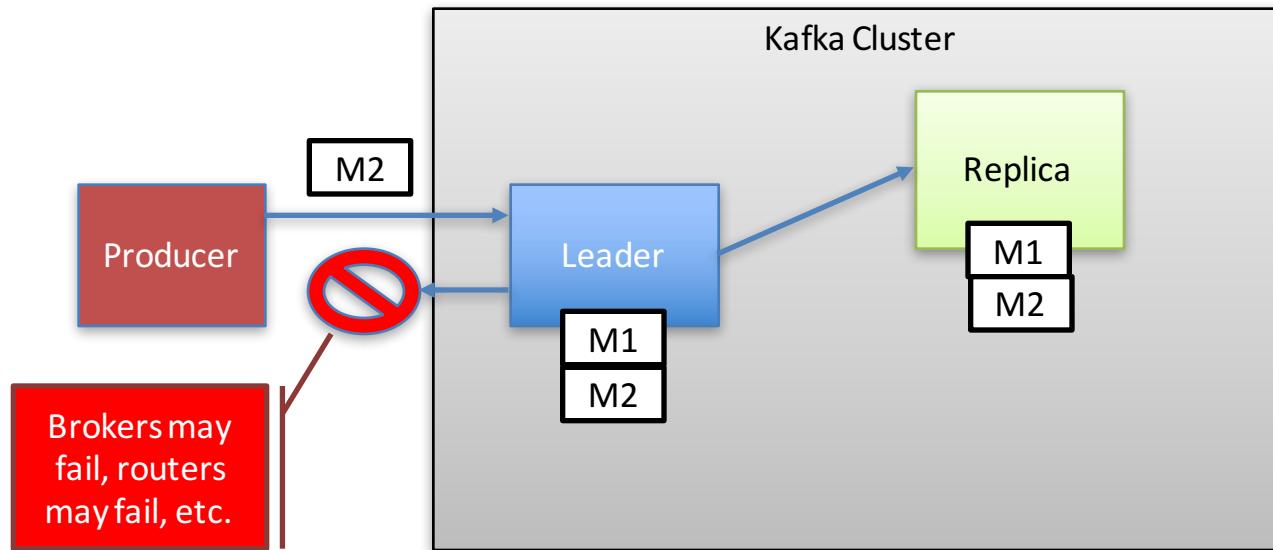
Message Duplication: 2. Message ack



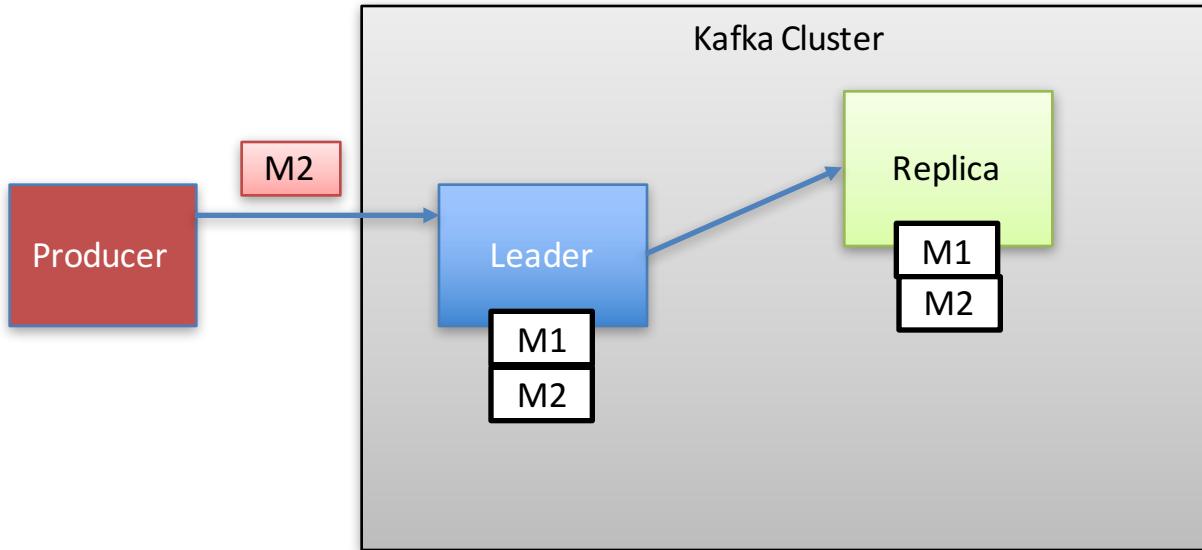
Message Duplication: 3. M2 being sent



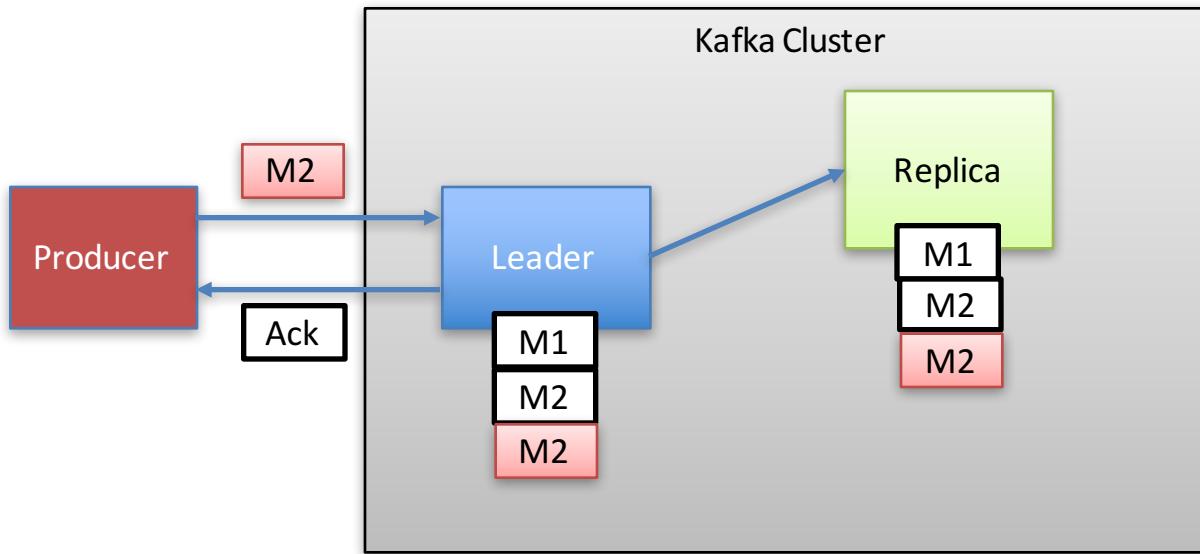
Message Duplication: 4. Message never ack-ed



Message Duplication: 5. M2 is resent



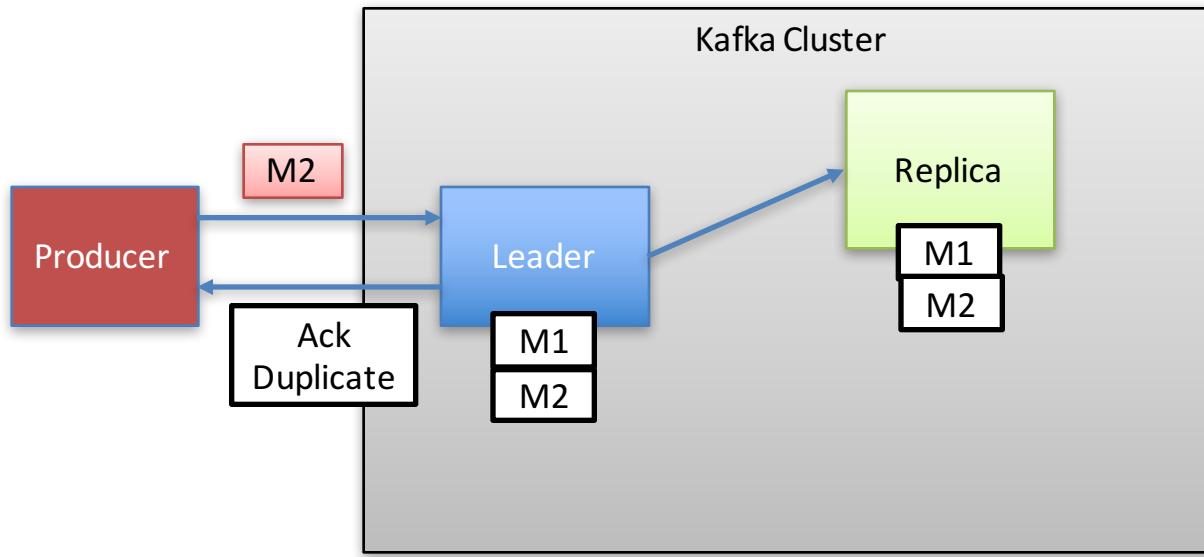
Message Duplication: 5. M2 is resent and stored again



Kafka Exactly Once Producer Solution

- Idempotent producer (per partition)
 - Exactly once
 - In order
- Transactions
 - Atomic writes across partitions

Idempotent Producer



Transaction API

- Atomic writes across multiple partitions
- Either all records are visible, or none

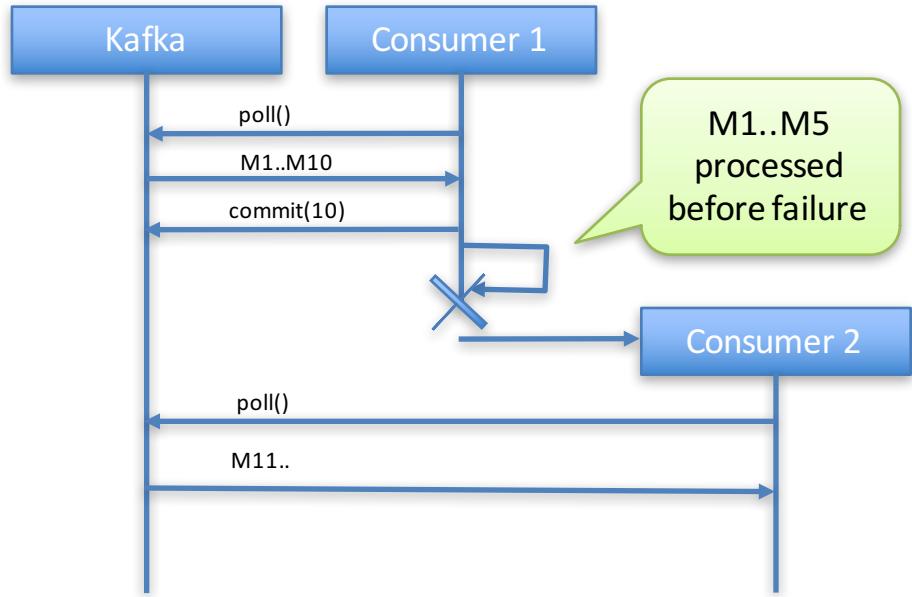
```
producer.initTransaction();
try {
    producer.beginTransaction();
    producer.send(record1);
    producer.send(record2);
    ...
    producer.commitTransaction();
}
catch (KafkaException e) {
    producer.abortTransaction();
}
```

Consumer Problems

- Messages missed
 - Offset is committed prior to processing the messages
 - No or only partial processing of incoming messages
 - Consumer failure
- Messages double processed
 - Processing of messages prior to commit
 - Consumer failure before commit
- There is now real failsafe way to ensure that messages are processed exactly once
 - Requires distributed transaction

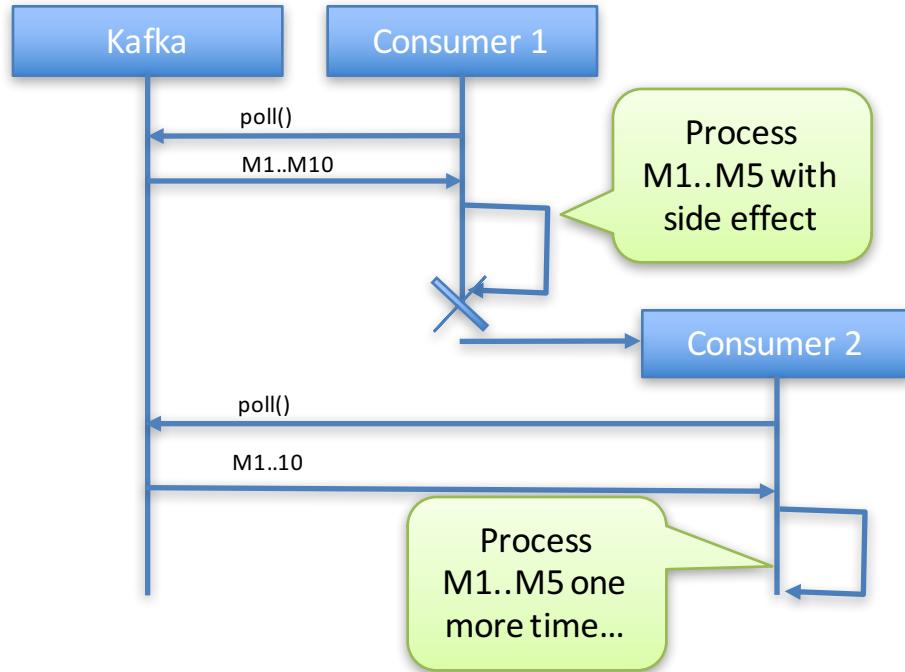
Consumer Message Loss

1. Messages received (M1...M10)
2. Offset committed (M10)
3. Processing M1..M5
4. Consumer crashes
5. Consumer resumes
6. Message M11.. is delivered



Consumer Double Processing

1. Messages received (M1...M10)
2. Processing M1..M5
3. Consumer crashes
4. Consumer resumes
5. Message M11.. is delivered



Kafka Exactly Once Stream Processing

- Kafka provides an exactly once guarantee for stream processing
- Important to understand the guarantee
 - The effect on the Kafka stream state is AS IF each message was processed exactly once
 - The message may actually be processed multiple times
 - However, the offset is moved with the state changes
- This means:
 - If the processing have side effect OUTSIDE KAFKA this side effect may have to be idempotent!

Configuration of Exactly Once

Producer Configuration

- enable.idempotence=true
- **Also recommended**
 - max.inflight.requests.per.connection=1
 - acks="all"
 - retries= MAX_INT

Consumer Configuration

- isolation.level = read_committed
or
- isolation.level = read_uncommitted
- processing.mode = "exactly_once"

Summary

- Kafka supports an exactly once message delivery guarantee
- Solved with
 - Idempotent producer
 - Transactions
 - Stream processing
- It's important to know that the exactly once guarantee on the consumer is only guaranteed within the context of Kafka!