Distributed Event Streaming Platform Components

Objectives

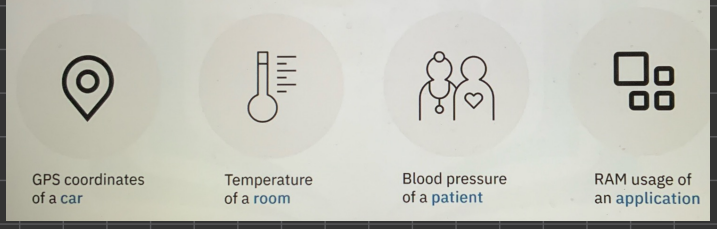
After watching this video, you will be able to:

- Describe what an Event is
- List the common Event formats
- Describe what an Event Stream Platform (ESP) is
- List the main components of an ESP
- List the popular ESPs

What is an Event?

in context of streaming

Events describe an entity's observable state updates over time



Common Event formats

three most common format

Primitive such as a plain text, number, or date

```
"Hello"
```

A key-value pair

```
Key: "car_id_1"
Value: (43.82,-79.48)
```

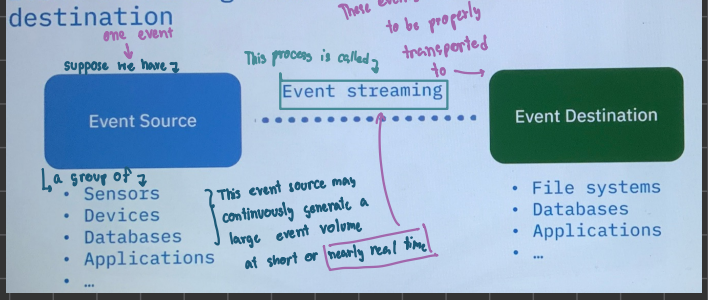
tuple

A key-value with a timestamp

```
Key: "patient_id"
Value: (125, 85)
Timestamp: 2021-07-01 12:00
```

One source to one destination

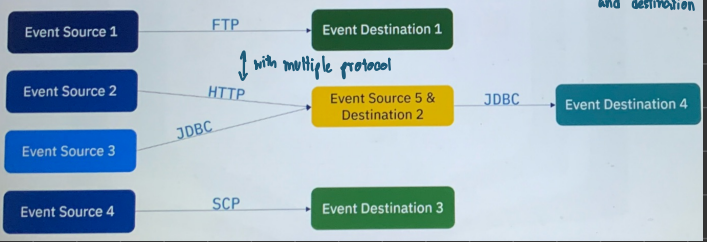
Event streaming from one event source to one destination



Many sources to many destinations

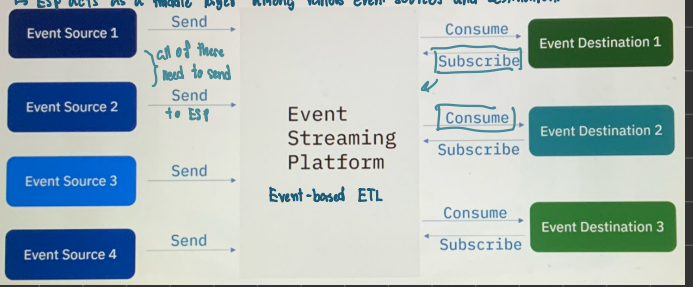
Event streaming from multiple event sources to multiple destinations

streaming can be really complicated with multiple distributed event sources and destination

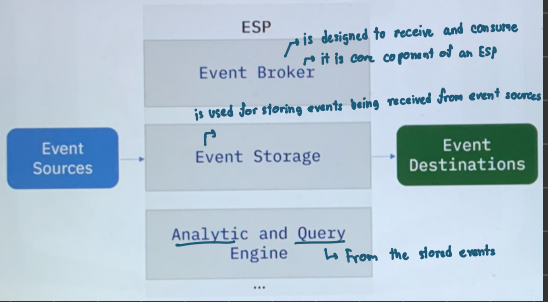


Event Streaming Platform (ESP)

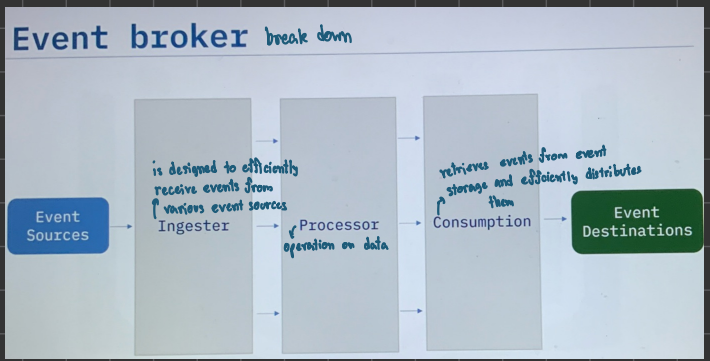
to handle different event destinations
↳ ESP acts as a middle layer among various event sources and destinations



Common components of an ESP



Event broker break down



Popular ESPs

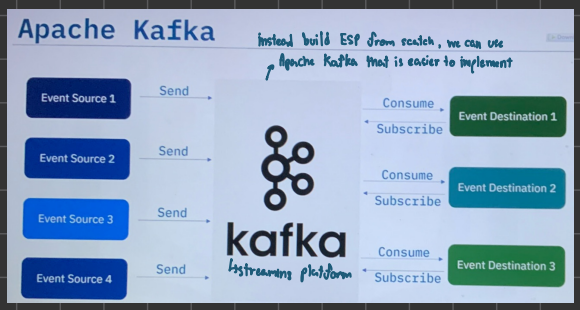
Apache Kafka Overview

Objectives

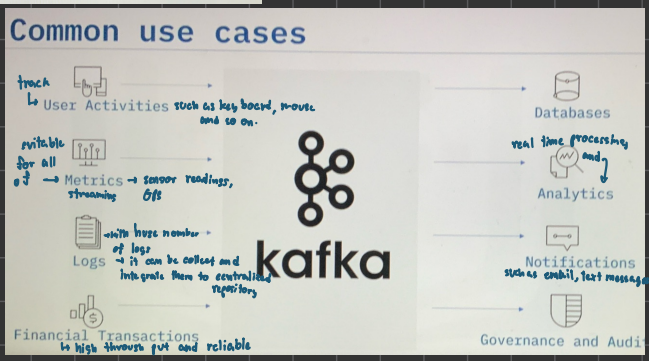
After watching this video, you will be able to:

- Recognize Apache Kafka as an Event Streaming Platform (ESP)
- Describe the architecture of Apache Kafka
- List common use cases for Apache Kafka
- Summarize the main features and benefits of Apache Kafka
- List popular ESP-as-a-Service providers

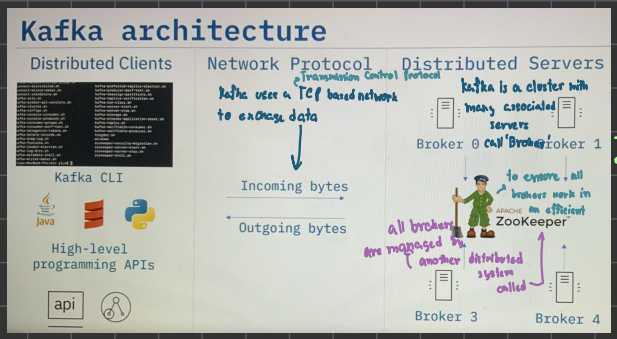
Apache Kafka



Common use cases



Kafka architecture



Main features of Apache Kafka

- Distribution system
- Highly scalable
- Highly reliable
- Permanent persistency
- Open source

Event streaming as a service

Confluent Cloud

IBM Event Streams

Amazon MSK

Building Event Streaming Pipelines using Kafka

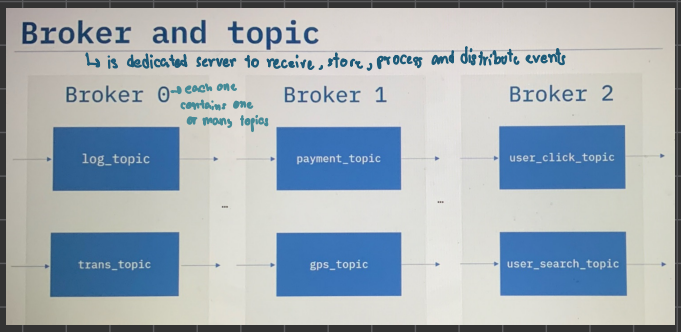
Objectives

After watching this video, you will be able to:

- Describe the core components of Kafka
- Use Kafka to publish (write) and subscribe to (read) streams of events
- Use Kafka to consume events, either as they occur or retrospectively
- Describe an end-to-end event streaming pipeline example

Broker and topic

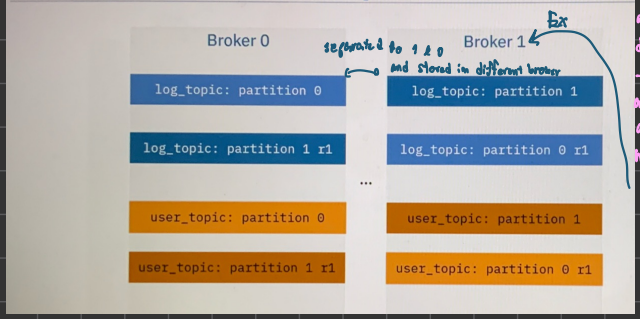
↳ is dedicated server to receive, store, process and distribute events



each one contains one or many topics

Partition and replications

to increase fault-tolerance and throughput
↳ so that event publication and consumption can be done in parallel
↳ even if some brokers are down, Kafka clients are still able to work with target topics replicated



Kafka topic CLI

Command line interface
↳ provides a collection of powerful script utils to build an event streaming pipeline

Create a topic:

```
kafka-topics --bootstrap-server localhost:9092 --topic log_topic --create
--partitions 2 --replication-factor 2
```

need host and port
create topic

List topics:

```
kafka-topics --bootstrap-server localhost:9092 --list
```

check all created topics in server

Get topic details:

```
kafka-topics --bootstrap-server localhost:9092 --describe log_topic
```

check detail of specify topic

Delete a topic:

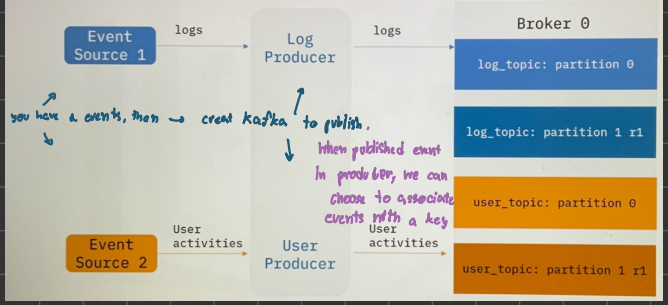
```
kafka-topics --bootstrap-server localhost:9092 --topic log_topic --delete
```

Kafka producer

- Client applications that publish events to topic partition
- An event can be optionally associated with a key
- Events associated with the same key will be published to the same topic partition
- Events not associated with any key will be published to topic partitions in rotation

How to publish events to topic partitions

Kafka producer in action



you have a events, then → creat kafka to publish.
When published event in producer, we can choose to associate events with a key

Kafka producer CLI

start a producer and point it to log

Start a producer to a topic, without keys:

```
kafka-console-producer --broker-list localhost:9092 --topic log_topic
> log1
> log2
> log3
```

← type some to start publishing events

provide to ensure the events with the same key will go to the same partition

Start a producer to a topic, with keys:

```
kafka-console-producer --broker-list localhost:9092 --topic user_topic
--property parse.key=true -property key.separator=,
> user1, login website
> user1, click the top item
> user1, logout website
```

↳ user1 will be saved in the same partition to facilitate the reading for consumers

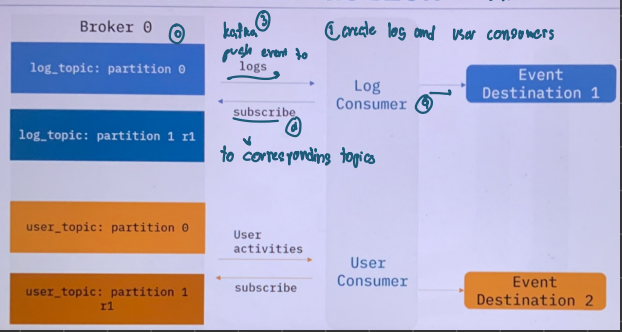
Kafka consumer

← Create to read topic partition

- Consumers are clients subscribed to topics
- Consume data in the same order
- Store an offset record for each partition
- Offset can be reset to zero to read all events from the beginning again

Kafka consumer in action

to read



create log and user consumers

Kafka consumer CLI

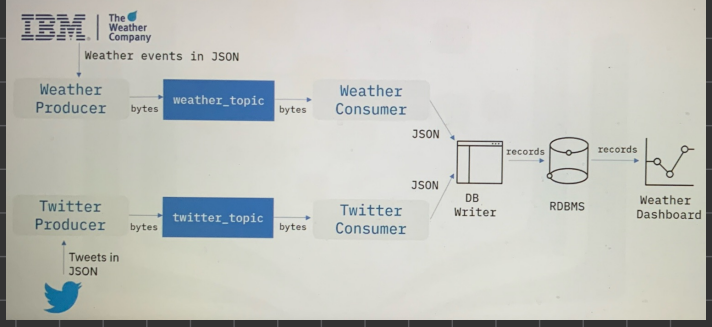
Read new events from the *log_topic*, *offset 1*:

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic log_topic
> (offset 2) log3
> (offset 3) log4
```

Read all events from the *log_topic*, *offset 0*:

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic log_topic
--from-beginning
> (offset 0) log1
> (offset 1) log2
> (offset 2) log3
> (offset 3) log4
```

A weather pipeline example



LAB

Exercise 1 - start ZooKeeper

ZooKeeper is required for Kafka to work.

1. Start the ZooKeeper server.

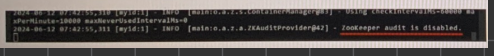
```
docker run --name myzookeeper --restart always zookeeper
```

Zookeeper runs on port `2181` at which the server will connect to the zookeeper. When ZooKeeper starts you should see an output that indicates that zookeeper is up and running.

You can be sure it has started when you see no error or exception that makes the server exits and return to command prompt.

ZooKeeper, is not required for Kafka to work in the latest version. However, you will use it to in this lab. ZooKeeper is responsible for the overall management of Kafka cluster. It monitors the Kafka brokers and notifies Kafka if any broker or partition goes down, or if a new broker or partition goes up.

Note: If you see **ZooKeeper audit is disabled** when starting the Zookeeper server, please ignore it and proceed further to next command. It will not hinder your progress.



Exercise 2 - Start the Kafka broker service

1. Start a new terminal.
2. Run the commands below. This will start the Kafka message broker service.

```
docker run --name mykafkaserver --link myzookeeper:zookeeper apache/kafka
```

Kafka runs on port `9092`. When Kafka starts, you should see an output indicating that the server is up and running.

You can be sure it has started when you see no error or exception that makes the server exits and return to command prompt.

Exercise 3 - Create a topic

You need to create a topic before you can start to post messages.

1. Start a new terminal.
2. To create a topic named `news`, start a new terminal and run the command below.

```
docker exec -it mykafkaserver /opt/kafka/bin/kafka-topics.sh --create --topic news --bootstrap-server localhost:9092
```

You will see the message: 'Created topic news.'

Exercise 4 - Start Producer

You need a producer to send messages to Kafka.

1. Run the command below to start a producer.

```
docker exec -it mykafkaserver /opt/kafka/bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic news
```

Once the producer starts, and you get the '>' prompt, type any text message and press enter. Or you can copy the text below and paste. The below text sends three messages to kafka.

```
Good morning
Good day
Enjoy the Kafka lab
```

Exercise 5 - Start Consumer

You need a consumer to read messages from kafka.

1. Open a new terminal.
2. Run the command below to listen to the messages in the topic `news`.

```
docker exec -it mykafkaserver /opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic news --from-beginning
```

You should see all the messages you sent from the producer appear here.

You can go back to the producer terminal and type some more messages, one message per line, and you will see them appear here.

Exercise 6 - Explore Kafka directories.

Kafka uses the directory /tmp//tmp/kraft-combined-logs to store the messages.

1. Open a new terminal.
2. Run the following command to open the kafka server in bash mode.

```
docker exec -it mykafkaserver /bin/bash
```

4. Explore the root directory of the server.

```
ls
```

5. You will observer that there is a `tmp` folder. The `kraft-combine-logs` inside the `tmp` folder contains all the logs. You can check the ones generated for topic `news` that you have created by running the following command.

```
ls /tmp/kraft-combined-logs/news-0
```

This is where all the messages are stored.

6. To exit from the bash, use the following command:

```
exit
```

Exercise 8 - Clean up

1. To stop the producer, in the terminal where you are running producer press `CTRL+C`.
2. To stop the consumer, in the terminal where you are running producer press `CTRL+C`
3. Stop the zookeeper and kafka server instance, run the following command.

```
docker stop myzookeeper mykafkaserver
```