# What is kafka?

https://kafka.apache.org/

## What is distributed

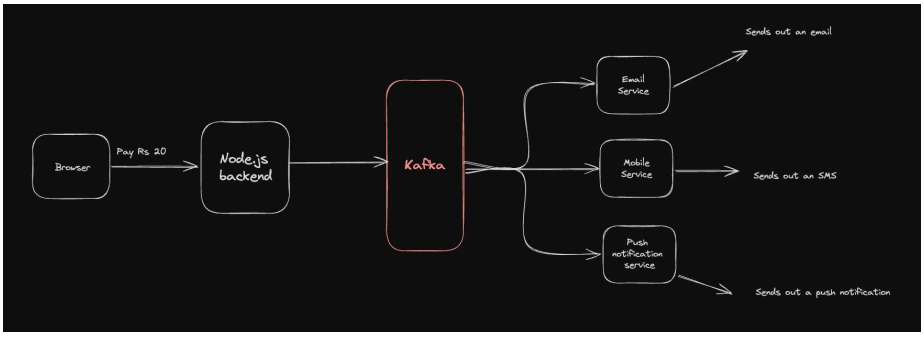
You can scale kafka horizontally by adding more nodes that run your kafka brokers

## Event streaming

If you want to build a system where one process produces events that can be consumed by multiple consumers

## Examples of apps

Payment notifications



# Jargon

# Start kafka locally

Ref - https://kafka.apache.org/quickstart

## Using docker

```
docker run -p 9092:9092 apache/kafka:3.7.1
```

## Get shell access to container

```
docker ps
docker exec -it container_id /bin/bash
cd /opt/kafka/bin
```

## Create a topic

```
./kafka-topics.sh --create --topic quickstart-events --bootstrap-server localh
```

## Publish to the topic

```
./kafka-console-producer.sh --topic quickstart-events --bootstrap-server loc
```

## Consuming from the topic

```
./kafka-console-consumer.sh --topic quickstart-events --from-beginning --b
```

# Kafka in a Node.js process

Ref https://www.npmjs.com/package/kafkajs

- Initialise project

```
npm init -y
npx tsc --init
```

- Update package.json

```
"rootDir": "./src",
"outDir": "./dist"
```

- Add src/index.ts

```
import { Kafka } from "kafkajs";

const kafka = new Kafka({
  clientId: "my-app",
  brokers: ["localhost:9092"]
})

const producer = kafka.producer();

const consumer = kafka.consumer({groupId: "my-app3"});


async function main() {
  await producer.connect();
  await producer.send({
    topic: "quickstart-events",
    messages: [{
      value: "hi there"
    }]
  })

  await consumer.connect();
  await consumer.subscribe({
    topic: "quickstart-events", fromBeginning: true
  })

  await consumer.run({
    eachMessage: async ({ topic, partition, message }) => {
      console.log({
        offset: message.offset,
        value: message?.value?.toString(),
```

```
  })
}


main();
```

- Update package.json

```
"scripts": {
   "start": "tsc -b && node dist/index.js"
},
```

- Start the process

```
npm run start
```

# Breaking into prodcuer and consumer scripts

Lets break our logic down into two saparate files

- producer.ts

```
import { Kafka } from "kafkajs";

const kafka = new Kafka({
  clientId: "my-app",
  brokers: ["localhost:9092"]
})

const producer = kafka.producer();

async function main() {
  await producer.connect();
  await producer.send({
    topic: "quickstart-events",
    messages: [{
      value: "hi there"
    }]
  });
}


main();
```

- consumer.ts

```
                    n "kafkajs";
```

```
const kafka = new Kafka({
  clientId: "my-app",
  brokers: ["localhost:9092"]
})

const consumer = kafka.consumer({ groupId: "my-app3" });


async function main() {
  await consumer.connect();
  await consumer.subscribe({
    topic: "quickstart-events", fromBeginning: true
  })

  await consumer.run({
    eachMessage: async ({ topic, partition, message }) => {
      console.log({
        offset: message.offset,
        value: message?.value?.toString(),
      })
    },
  })
}


main();
```

- Update package.json

```
"scripts": {
  "start": "tsc -b && node dist/index.js",
  "produce": "tsc -b && node dist/producer.js",
  "consume": "tsc -b && node dist/consumer.js"
},
```

- Try starting multiple consumers, and see if each gets back a message for the messages produced

Notice we specified a consumer group (my-app3)

# Consumer groups and partitions

## Consumer group

A consumer group is a group of consumers that coordinate to consume a topic.

**Purpose:**

- **Load Balancing:** Distribute the processing load among multiple consumers.
- **Fault Tolerance:** If one consumer fails, Kafka automatically redistributes the partitions that the failed consumer was handling to the remaining consumers in the group.
- **Parallel Processing:** Consumers in a group can process different partitions in parallel, improving throughput and scalability.

### Partitions

Partitions are subdivisions of a Kafka topic. Each partition is an ordered, immutable sequence of messages that is appended to by producers. Partitions enable Kafka to scale horizontally and allow for parallel processing of messages.

### How is a partition decided?

When a message is produced to a Kafka topic, it is assigned to a specific partition. This can be done using a round-robin method, a hash of the message key, or a custom partitioning strategy.

Usually you'll take things like `user id` as the `message key` so all messages from the same user go to the same consumer (so a single user doesnt starve everyone lets say)

### Multiple consumer groups

# Partitions in kafka

In this slide, we'll talk about what are partitions in Kafka

- Create a new topic with 3 partitions

```
./kafka-topics.sh --create --topic payment-done --partitions 3 --bootstrap-s
```

itions

```
./kafka-topics.sh --describe --topic payment-done --bootstrap-server localh
```

- Update the topic in the node.js script to use `payment-done`

```
async function main() {
  await producer.connect();
  await producer.send({
    topic: "payment-done",
    messages: [{
      value: "hi there",
      key: "user1"
    }]
  });
}
///
await consumer.subscribe({
  topic: "payment-done", fromBeginning: true
})
```

- Consume messages in 3 terminals

```
npm run consume
```

- produce messages

```
npm run produce
```

- Notice the messages get consumed by all 3 consumers

### Current architecture

### Assignment

Try deleting a few consumers and notice the partition gets re-assigned to a different one

Use the following to see which consumer is consuming on which topic

```
./kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --
```

# Three cases to discuss

### partitions and consumers

### More partitions

### More consumers

## Partitioning strategy

When producing messages, you can assign a key that uniquely identifies

Kafka will hash this key and use the hash to determine the partition. This ensures that all messages with the same key (lets say for the same user) are sent to the same partition.

> Why would you want messages from the same user to go to the same partition?
> Lets say a single user has too many notifications, this way you can make sure they only choke a single partition and not all the partitions

- Create a new `producer-user.ts` file, pass in a `key` when producing the message

```
import { Kafka } from "kafkajs";

const kafka = new Kafka({
  clientId: "my-app",
  brokers: ["localhost:9092"]
})

const producer = kafka.producer();

async function main() {
  await producer.connect();
  await producer.send({
    topic: "payment-done",
    messages: [{
      value: "hi there",
      key: "user1"
    }]
  });
}

main();
```

- Add `produce:user` script

```
"produce:user": "tsc -b && node dist/producer-user.js",
```

- Start 3 consumers and one producer. Notice all messages reach the same consumer

```
npm run produce:user
```