Introduction to Cloud

upGrad

Session: Cloud Architectures

Instructor: Siben Nayak

Cloud Architectures

SESSION INTRODUCTION

Monoliths and Microservices

Synchronous and Asynchronous Communication

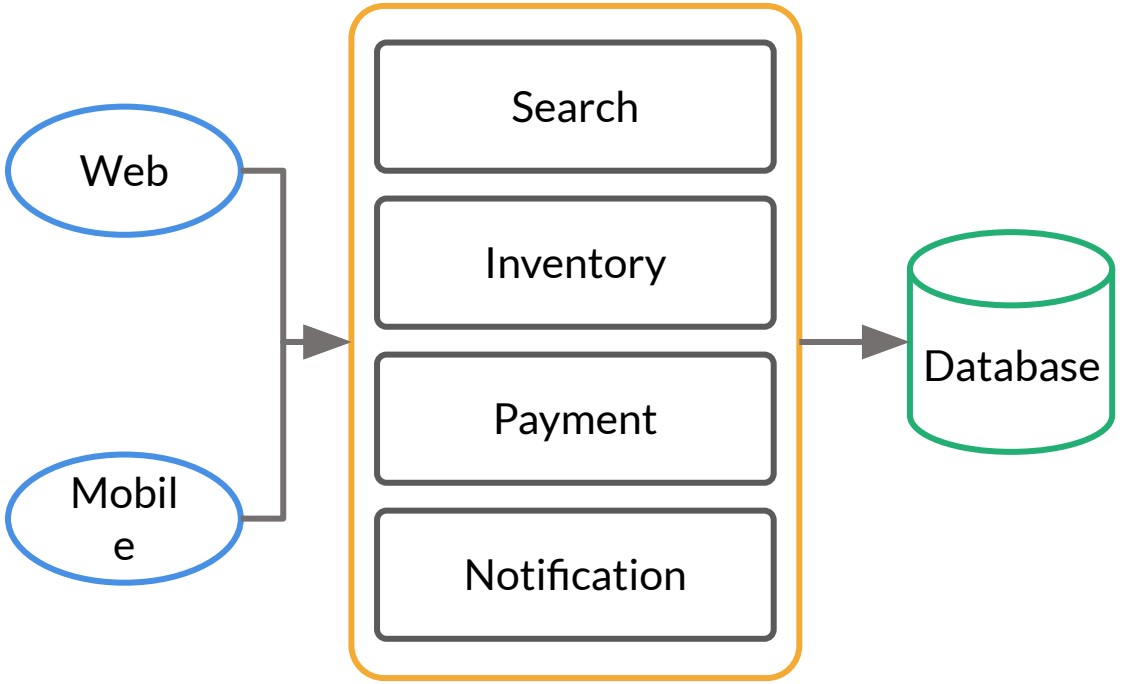
Event-Driven Architecture

Message Queues

Microservice Architecture

MONOLITHS

- An approach in which the entire application is packaged and deployed as a single unit
- Multiple independent components co-exist within the same codebase
- Single database for the entire application
- Same application deployed for all platforms (web, mobile, etc.)
- Intra-process communication between components is simple code invocation



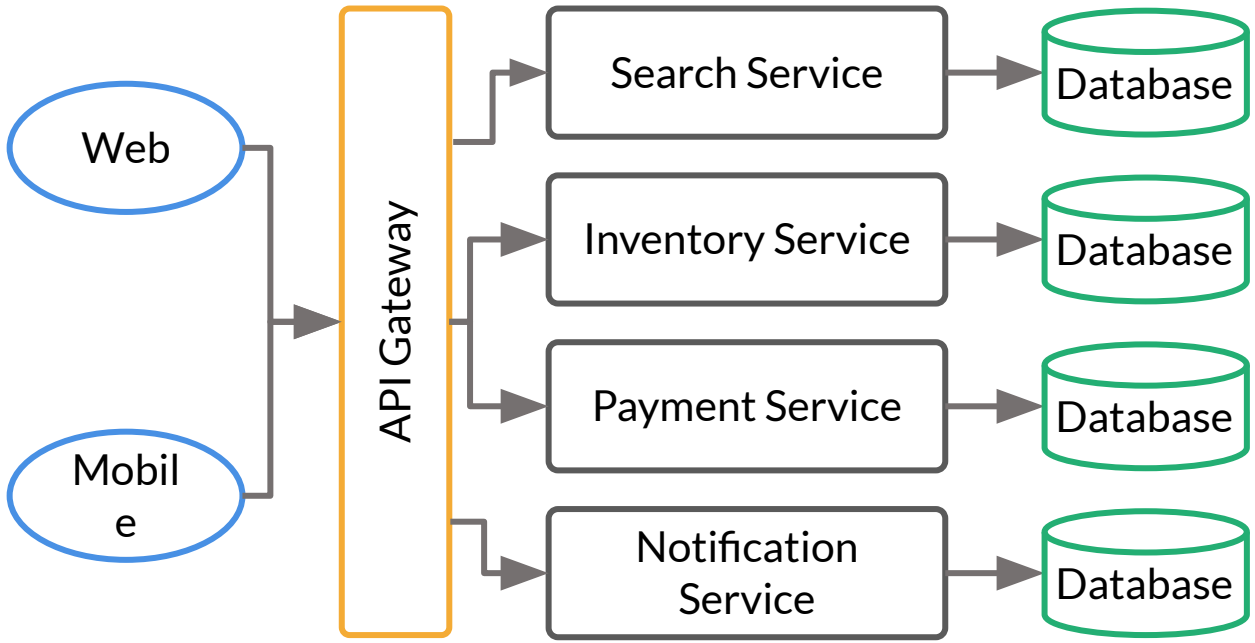
PROS AND CONS

- Since the entire application is part of a single code base, it makes it easier to import and develop in any IDE
- Having the entire code at the same place also makes it easier to test since the entire functionality is available for testing
- The entire application is bundled as a single artifact (a WAR or JAR file) and it is easy to just replace it in the runtime environment
- The application can be simply copied to run multiple instances behind a load balance to scale horizontally

- As the application grows, the code size grows and becomes difficult to maintain and understand which decreases development speed over time.
- Larger code size often overloads the IDE and the application server, leading to lesser developer productivity while making changes
- Changing any single part of the application requires the entire application to be redeployed and retested
- An issue or bug in one part of the code can potentially bring down the entire application due to the tight coupling

MICROSERVICES

- An approach in which a single application is composed of many loosely coupled and independently deployable smaller services
- Have their own stack which includes the database
- Communicate with one another over REST APIs and events
- Organised by business capability and separated by a bounded context
- Governance and data management are decentralised

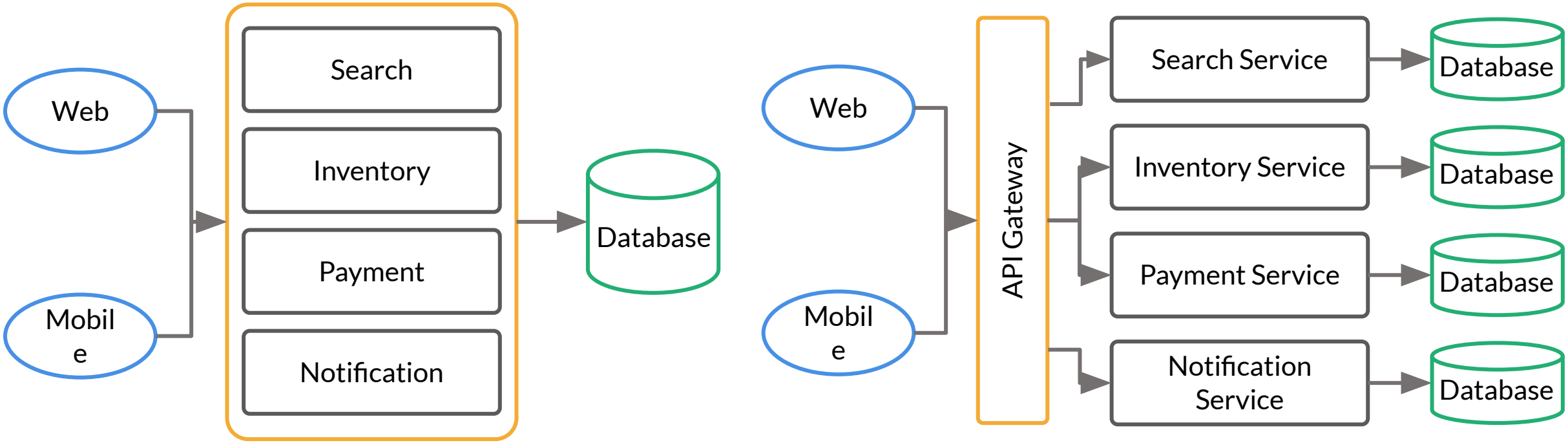


PROS AND CONS

- Since each service is independent and small, the code is easier to understand and modify
- Services are smaller, and hence faster to test and can be deployed independently
- Development can be done by multiple teams where each team completely manages a smaller service independently of other teams
- Any bug or issue is isolated to a single service and the other services continue to function and handle traffic

- Additional complexities arise due to distributed systems such as debugging and handling distributed transactions
- Requests that span multiple services are difficult to build and require strong cooperation between teams
- Operational overhead of deploying, monitoring and managing a system composed of multiple services
- Increased resource requirements since each service now requires its own infrastructure and resources

MONOLITH VS MICROSERVICE ARCHITECTURE

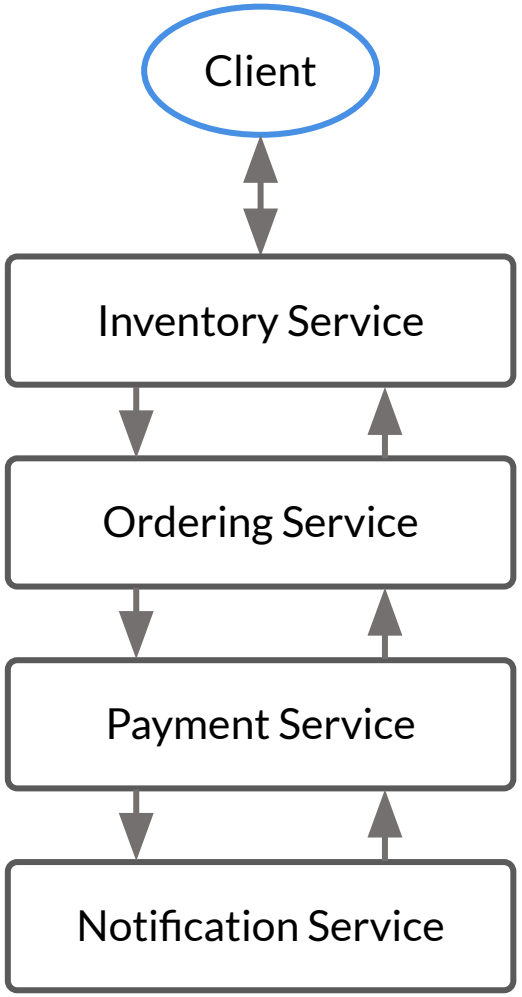


The term ‘microservice’ was discussed at a workshop of software architects near Venice in May 2011 to describe what the participants saw as a common architectural style that many of them had been recently exploring. In May 2012, the same group decided on ‘microservices’ as the most appropriate name.

Event-Driven Architecture

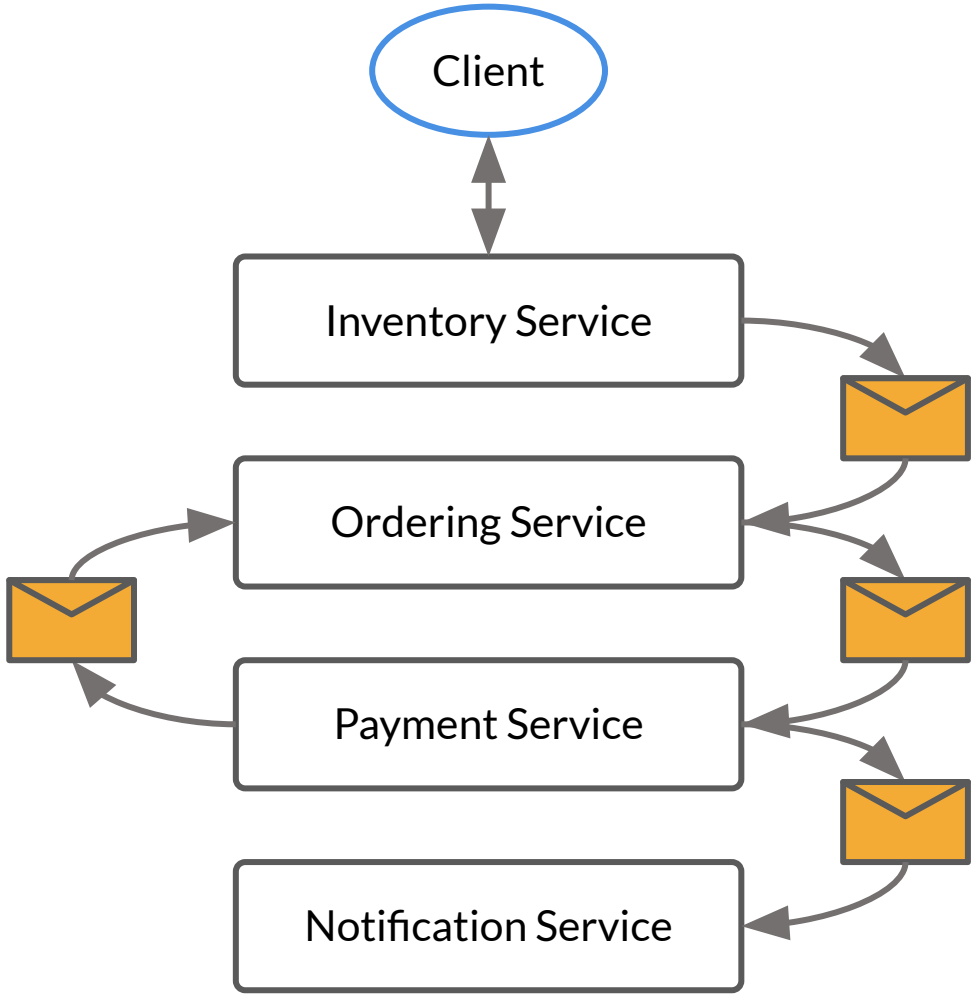
SYNCHRONOUS COMMUNICATION

- Exchange of information in real-time
- Client sends a request and waits for a response from the server
- Client is blocked till response is received
- Needs well-defined contract between client and server
- HTTP/HTTPS are synchronous communication protocols



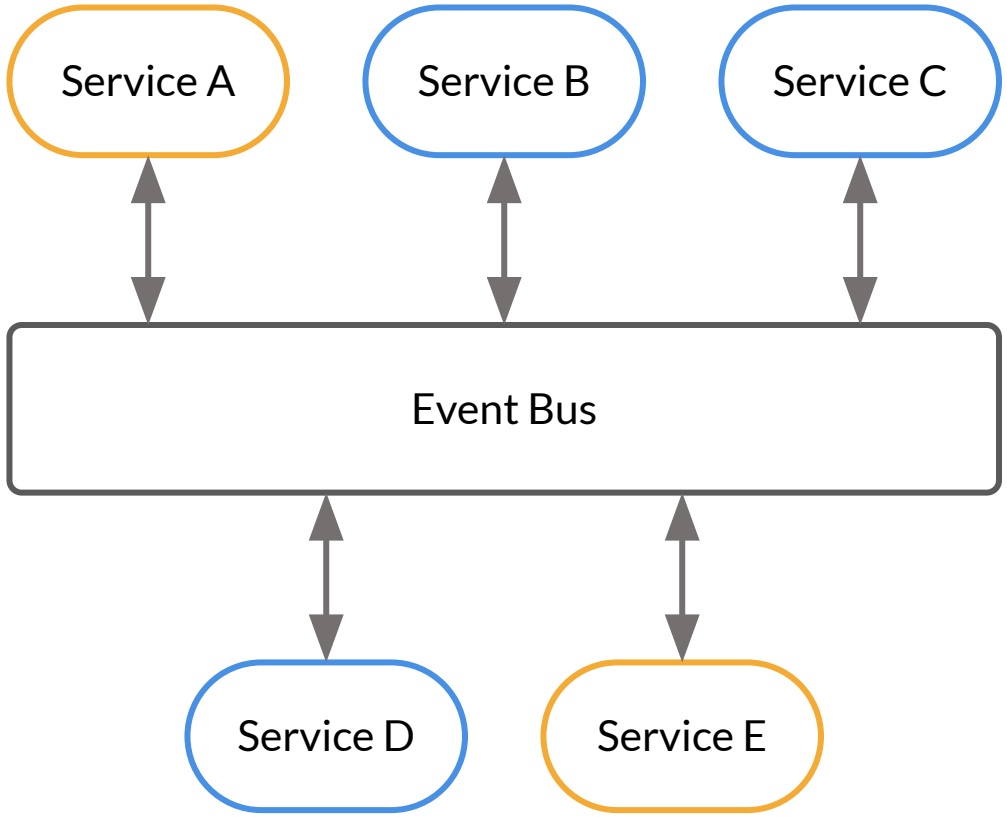
ASYNCHRONOUS COMMUNICATION

- Non-real-time exchange of information
- Client sends a request and continues to process other tasks
- A non-blocking mechanism where client does not have to wait for the response from server
- Does not need a well-defined contract between two services
- Asynchronous communication can be achieved through message brokers such as Apache Kafka, RabbitMQ and AWS SQS



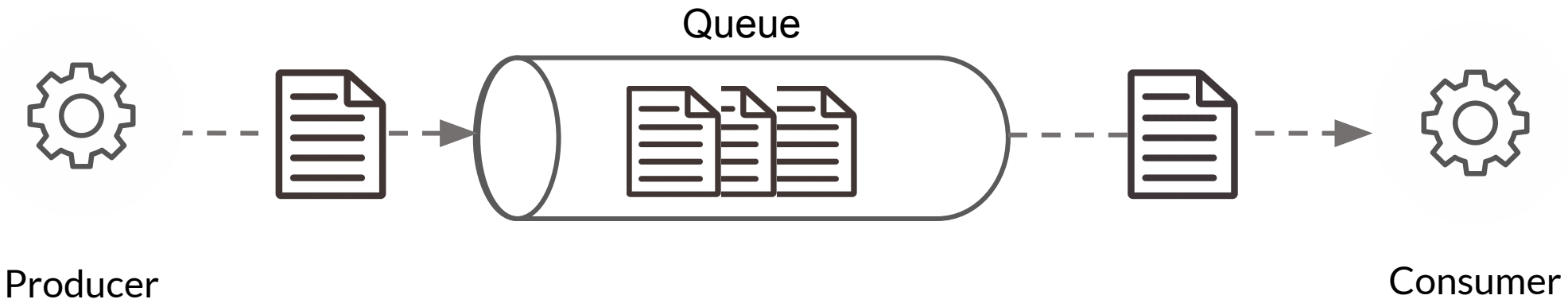
EVENT DRIVEN ARCHITECTURE

- Use of events to communicate between services
- Events signify a change in the state of the system
- Events can either carry the entire state, or just an identifier for the state change
- Publisher/producer publishes the state change event
- Router/broker/bus/queue holds the events sent by the publisher
- Subscriber/Consumer listens to the state change event and takes appropriate action

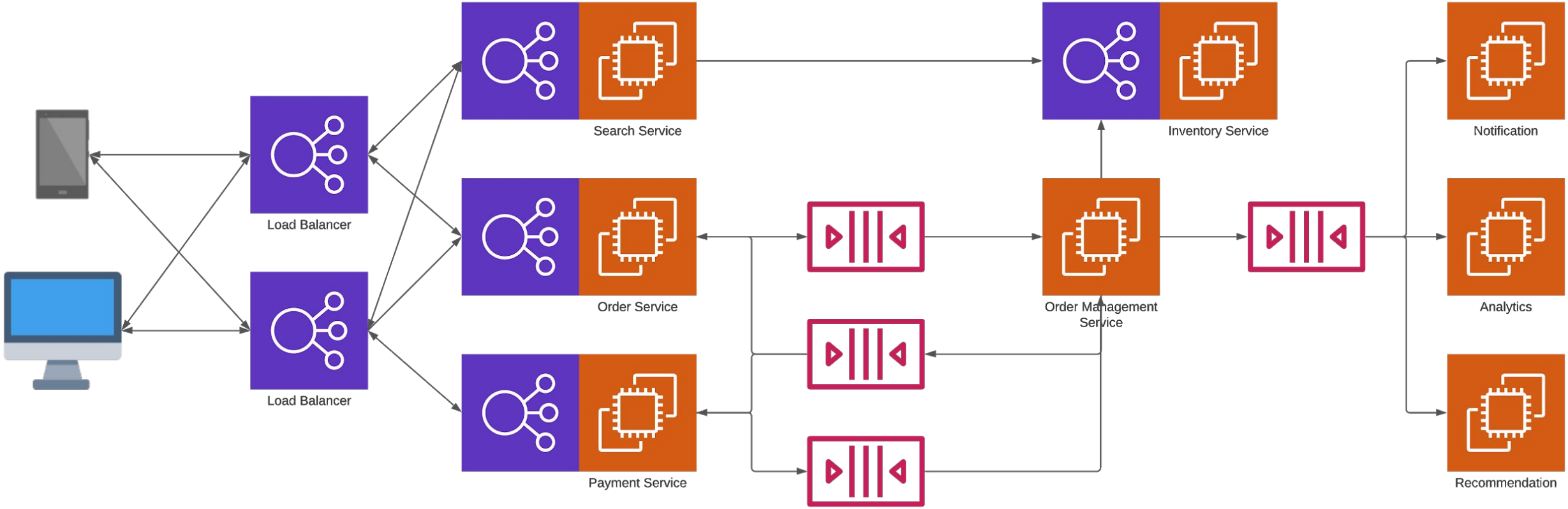


MESSAGE QUEUES

- A temporary queue like storage that holds messages until they are processed (for example: Amazon SQS, Apache Kafka and RabbitMQ)
- Allow different components and services to communicate and process tasks asynchronously
- Used to decouple heavy processing, batch work and handle burst traffic
- Ability to define the order in which messages are processed
- Multiple producers and consumers can use the queue at the same time



EXAMPLE - AMAZON



Summary

SUMMARY

- **Monolith** is an approach in which the entire application is packaged and deployed as a **single unit**
- **Microservices** is an approach in which a single application is composed of **many loosely coupled** services
- In microservices, communication happens **synchronously** or **asynchronously**
- Event driven architecture uses **queues** to communicate **asynchronously**