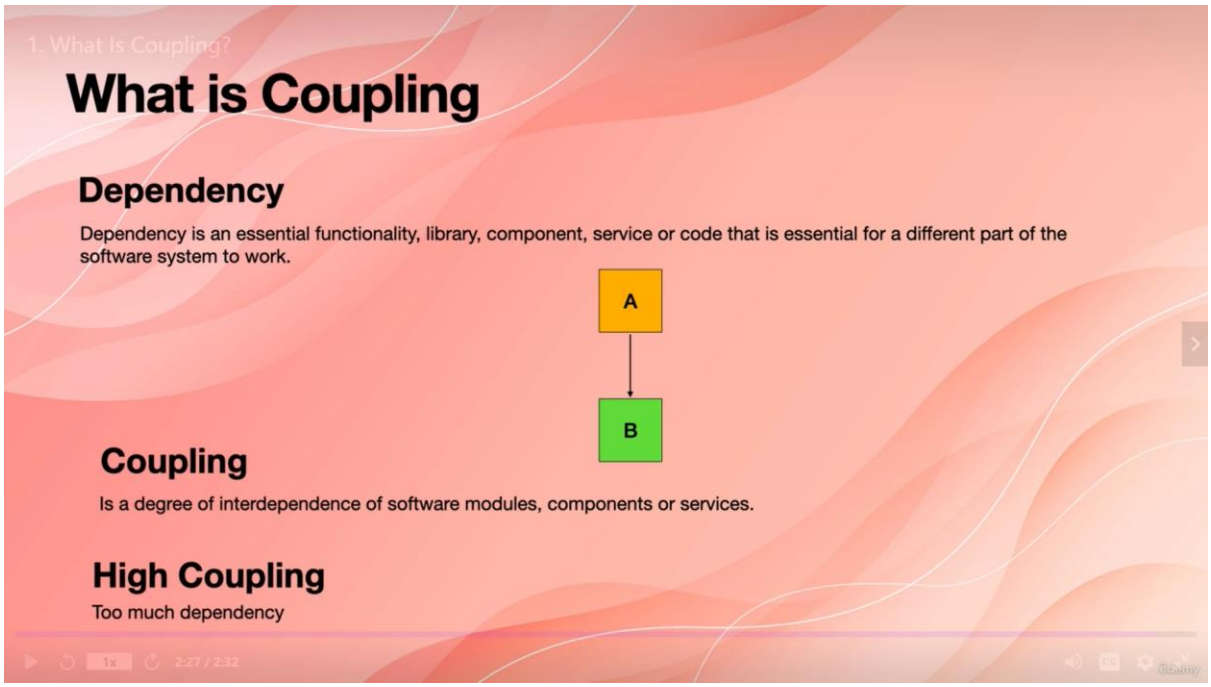
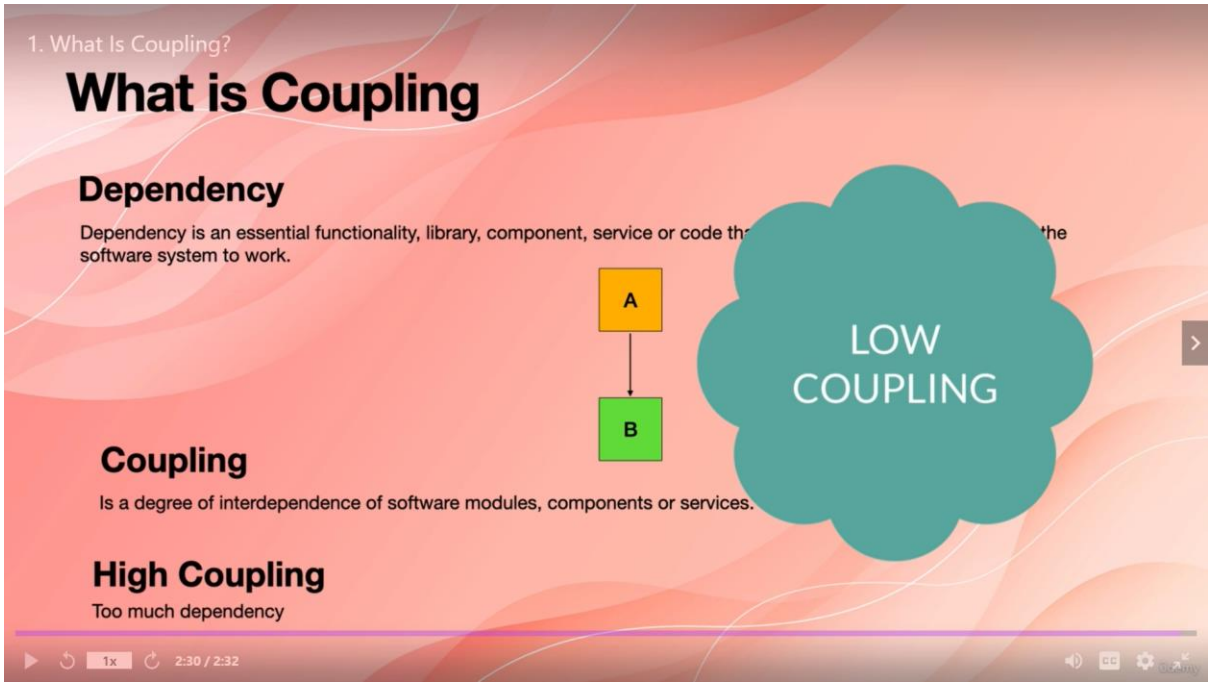
Microservices Interview Questions - Passsing Guarranteed



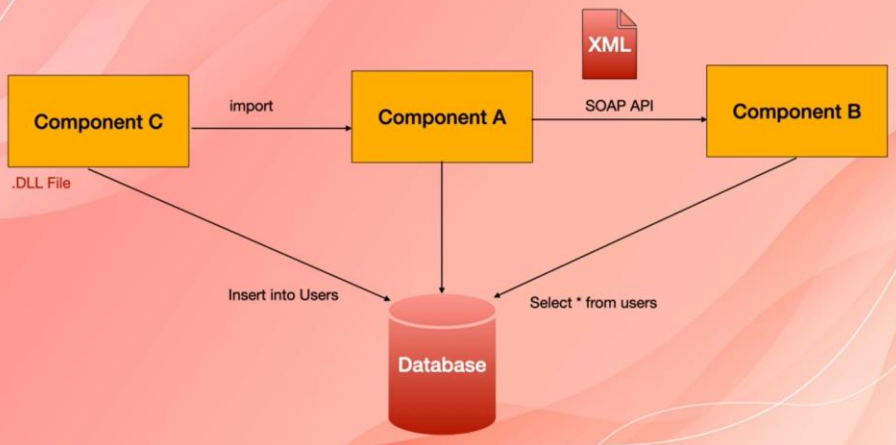


Types of Coupling

Content Coupling	One class can access the private members of another class.
Common Coupling	Two classes access the same shared data i.e. global variables and static properties
Control Coupling	When a function controls the flow of another function
Routine Coupling	When one function calls another function without passing any parameters
Data Coupling	When two systems share the same database
Type Use Coupling	When a member or property of class B is of type class A
Stamp Coupling	When in Class B a method has a parameter of type Class A
Import Coupling	When a library is imported into another program i.e. DLL files in Windows
External Coupling	When communicating with an external program

Types of Coupling

Examples



What is Cohesion?

What is better? High Cohesion or Low Cohesion?

Is a measure of the degree to which the elements of a module, software or code are related.

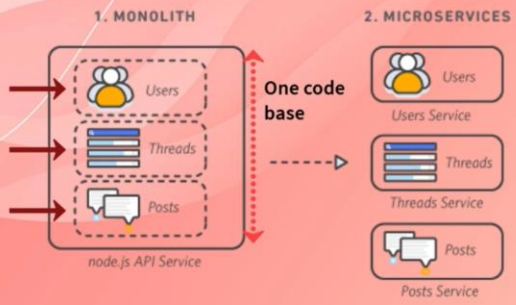
Is a degree to which all elements of a module, software or code are directed towards performing a single task.

High Cohesion is **GOOD**!

Example of low Cohesion: **Helper Classes**

```
public static class Helper
{
    public static string DateToString(DateTime dateTime) {...}
    public static SmtpConfig GetSmtpConfig();
    public static void LogError(string error);
}
```

What is a Microservice?

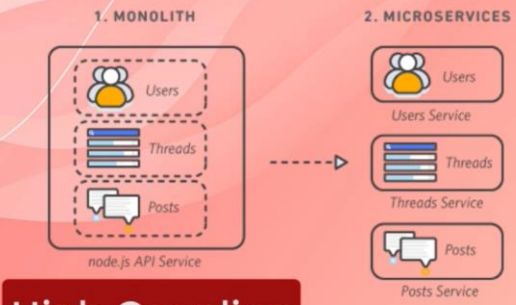


Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate with each other in a loosely coupled manner.

Microservices try to solve the problems of Monolithic Applications.

With monolithic architectures, all processes are tightly coupled and run as a single service.

What is a Microservice?



High Coupling

Low Cohesion

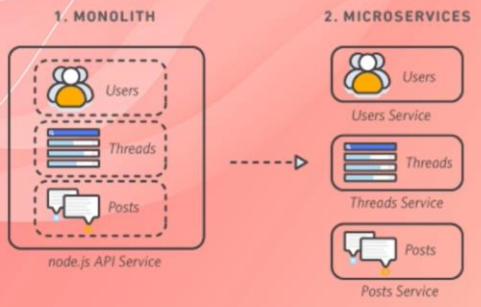
Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate with each other in a loosely coupled manner.

Microservices try to solve the problems of Monolithic Applications.

With monolithic architectures, all processes are tightly coupled and run as a single service.

4. What is a microservice

What is a Microservice?



Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate with each other in a loosely coupled manner.

Microservices try to solve the problems of Monolithic Applications.

With monolithic architectures, all processes are tightly coupled and run as a single service.

1x 2:49 / 2:50

Pros and Cons of Monolithic Applications

GOOD

Simplicity Monolithic Applications are easier to build, test and deploy.

Cross-cutting concerns One piece of code can handle monitoring, logging, security etc.

Performance Different classes or functions communicate directly and share the same memory so the application runs faster compared to Microservices where services communicate through a network.

BAD

Reliability A fatal error in any part of the application will crash the entire system.

Updates and Deployment Even for a small change the entire application has to be re-deployed

Technology Stack The entire application has to be developed with one single technology stack

What are alternatives of Monolith Apps

Single Codebase Monolithic Apps

The entire code of the application is compiled into one single binary file i.e. EXE in Windows

Reusability very limited and only at class-level.

Component-based Monolithic Apps

Application is made of multiple binary files (i.e. DLL files in Windows)

All libraries run as part of a single process and app.

Service-Oriented Architecture (SOA)

Application is broken down to services.

Services communicate via SOAP or Restful (HTTP) APIs.

Normally same technology stack is used.

Normally same technology stack is used.

Services use a shared database

Microservices Architecture

Services are independent and can be used in any project or application

SOAP APIs are deprecated

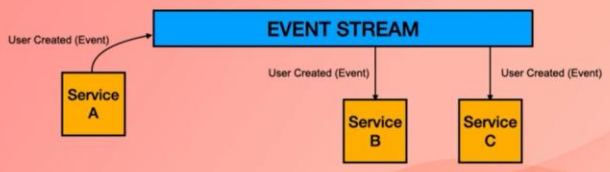
Microservices can be called in an asynchronous manner

No shared database

Can mix and match technology stacks

What are benefits of microservices architecture?

- Microservices can be deployed independently
- Microservices can be scaled out independently
- Microservices offer better fault tolerance
- A smaller code base allows new team members learn the code easier and quicker
- Different microservices can be built with different technology stacks
- Microservices can communicate asynchronously - reducing coupling



What are the drawbacks of microservices?

- Microservices-based systems are more complex to design and build
- Requires changes in how teams and organisations work
  - More teams to manage
  - Often more developers are required
  - Teams have to collaborate more efficiently
- Often more expensive to build
- Diagnosing problems become difficult
- Testing becomes more challenging

Microservices architecture is NOT always fit for purpose

How do micro services relate to the business?

- Businesses prefer to use cloud i.e.AWS as opposed to on-prem infrastructure to avoid a large Capex

How do micro services relate to the business?

- Businesses prefer to use cloud i.e.AWS as opposed to on-prem infrastructure to avoid a large Capex
- Micro services are a great fit for cloud
- Micro services enable businesses to better implement agile
- Micro services allow businesses to out-source part of their development effort, and get to market quicker

Name some design patterns and tools we use in developing microservices

- Event Streaming
- Message Broker
- Containers & Docker
- No-SQL Database
- Restful API
- API Gateway

What is Blast Radius?

Blast Radius is the degree to which the entire system is affected if a micro service fails or shuts down.

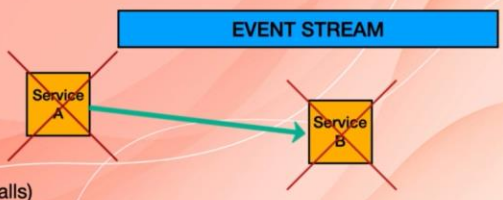
In a monolithic application the blast radius is 100%

If part of the application encounters a fatal error

If the database becomes unavailable (i.e. due to network misconfiguration, overloading etc) or breaks

In the micro services architecture we aim at reducing the blast radius with Patterns of Resilience

- Independent database for each microservice
- Timeouts
- Backoff strategy (retries)
- Circuit breaker pattern
- Bulkhead pattern
- Fallback pattern
- Asynchronous communication (no point-to-point calls)



What is Blast Radius?

Blast Radius is the degree to which the entire system is affected if a micro service fails or shuts down.

In a monolithic application the blast radius is 100%

If part of the application encounters a fatal error

If the database becomes unavailable (i.e. due to network misconfiguration, overloading etc) or breaks

In the micro services architecture we aim at reducing the blast radius with Patterns of Resilience

- Independent database for each microservice
- Timeouts
- Backoff strategy (retries)
- Circuit breaker pattern
- Bulkhead pattern
- Fallback pattern
- Asynchronous communication (no point-to-point calls)

What is Blast Radius?

Blast Radius is the degree to which the entire system is affected if a micro service fails or shuts down.

In a monolithic application the blast radius is 100%

If part of the application encounters a fatal error

If the database becomes unavailable (i.e. due to network misconfiguration, overloading etc) or breaks

In the micro services architecture we aim at reducing the blast radius with Patterns of Resilience

- Independent database for each microservice
- Timeouts
- Backoff strategy (retries)
- Circuit breaker pattern
- Bulkhead pattern
- Fallback pattern
- Asynchronous communication (no point-to-point calls)

What is Blast Radius?

Blast Radius is the degree to which the entire system is affected if a micro service fails or shuts down.

In a monolithic application the blast radius is 100%

If part of the application encounters a fatal error

If the database becomes unavailable (i.e. due to network misconfiguration, overloading etc) or breaks

In the micro services architecture we aim at reducing the blast radius with Patterns of Resilience

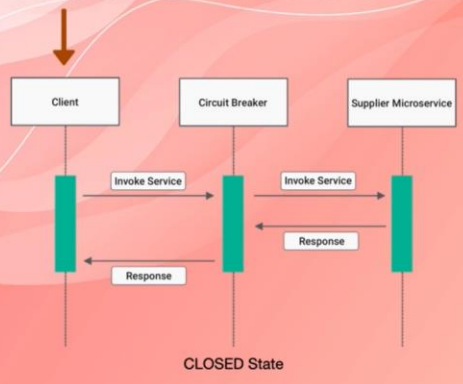
- Independent database for each microservice
- Timeouts
- Backoff strategy (retries)
- Circuit breaker pattern
- Bulkhead pattern
- Fallback pattern
- Asynchronous communication (no point-to-point calls)

Circuit Breaker Pattern

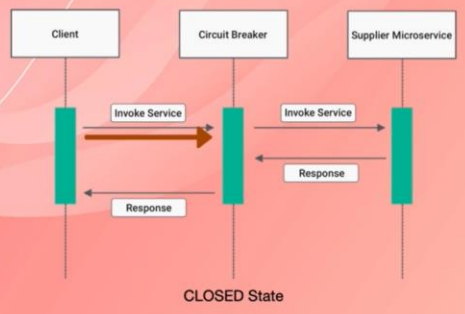
Applies to API (point to point) calls

- We set a threshold for failing API calls i.e. 3
- If failing API calls exceed the threshold we reject the API calls
- Circuit breakers are micro services themselves and act as Proxy
- Circuit breakers can have three states: Open, Closed, Half-Open

Circuit Breaker Pattern

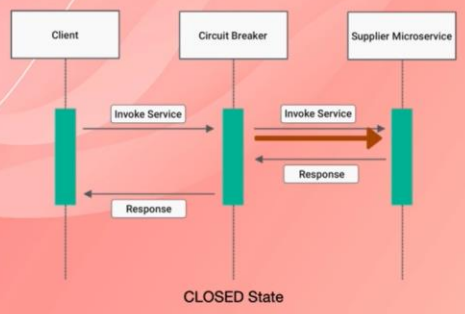


Circuit Breaker Pattern



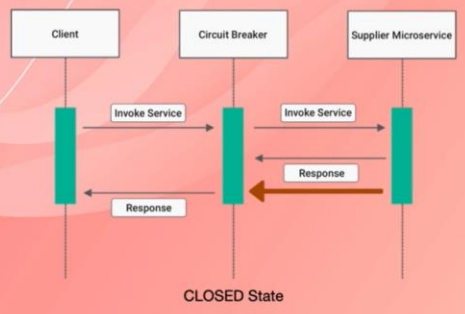
CLOSED State

Circuit Breaker Pattern



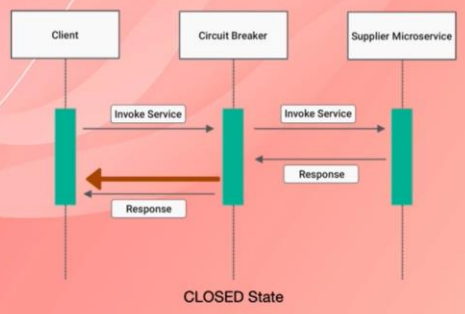
CLOSED State

Circuit Breaker Pattern



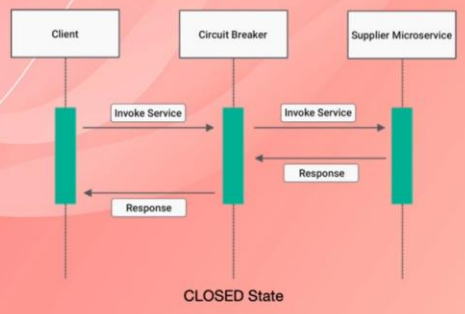
CLOSED State

Circuit Breaker Pattern



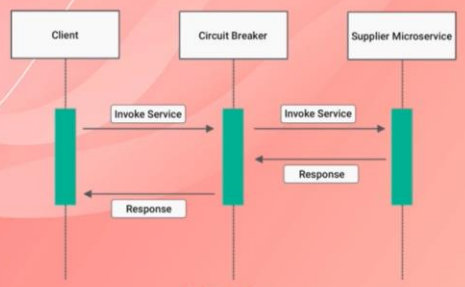
CLOSED State

Circuit Breaker Pattern

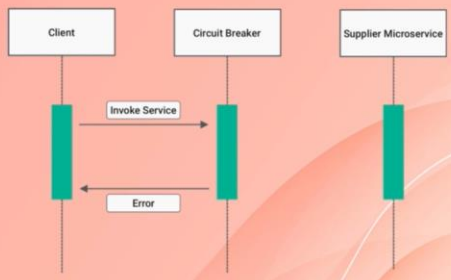


CLOSED State

Circuit Breaker Pattern

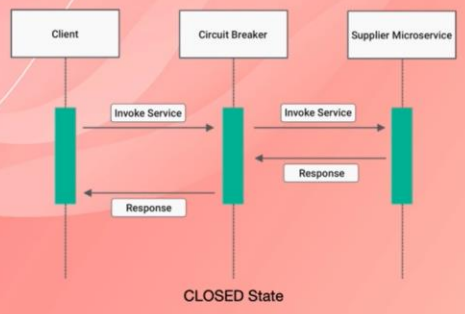


CLOSED State

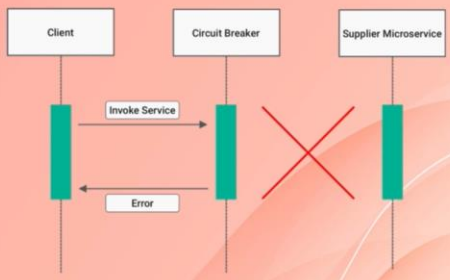


OPEN State

Circuit Breaker Pattern

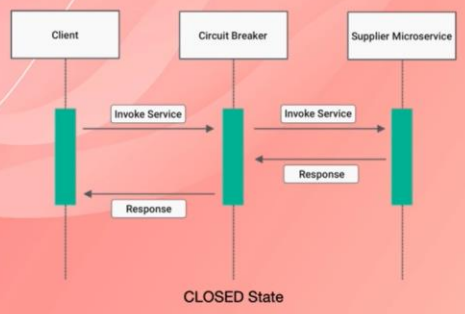


CLOSED State

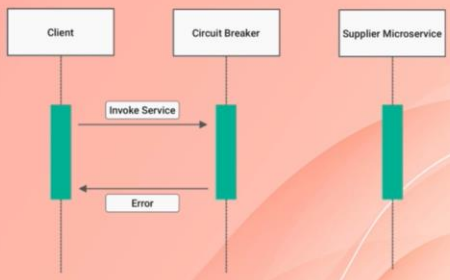


OPEN State

Circuit Breaker Pattern

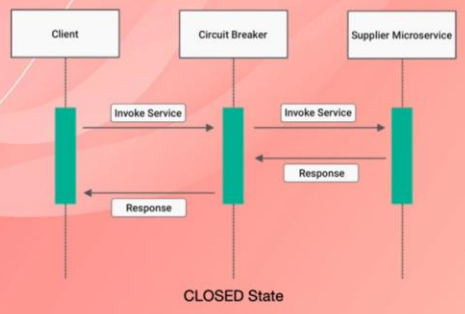


CLOSED State

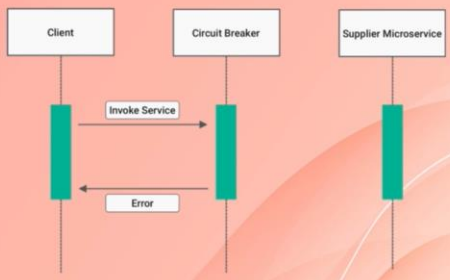


OPEN State

Circuit Breaker Pattern



CLOSED State



OPEN State

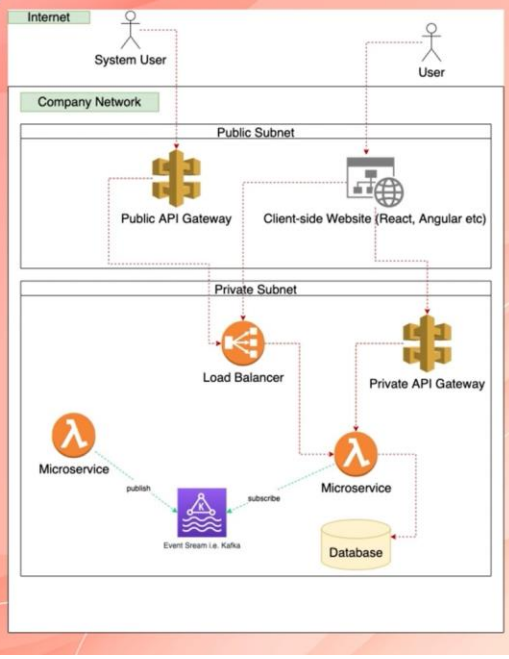
Name some ways of building micro services

- Single-tenant micro services : One micro service on one virtual server
  Good for migrating monolith to micro services
- Containerised micro services: Each micro service runs as a Docker container

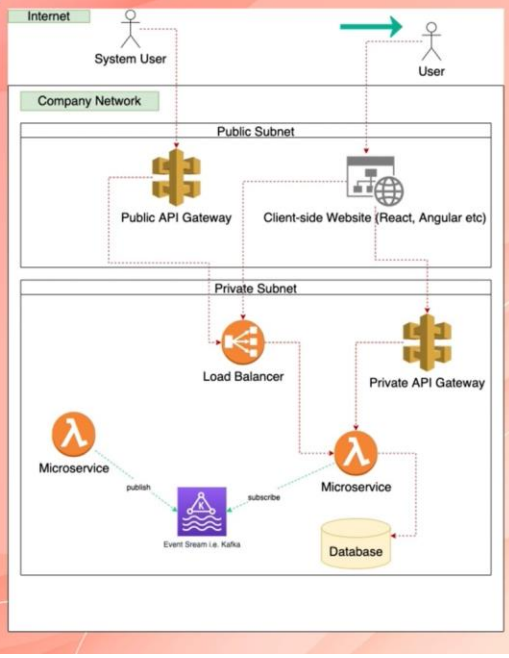
Name some ways of building micro services

- Single-tenant micro services : One micro service on one virtual server
  Good for migrating monolith to micro services
- Containerised micro services: Each micro service runs as a Docker container
- Serverless: Small micro services that run without a server in the cloud i.e. Amazon Lambda in AWS
- Spring Boot in Java

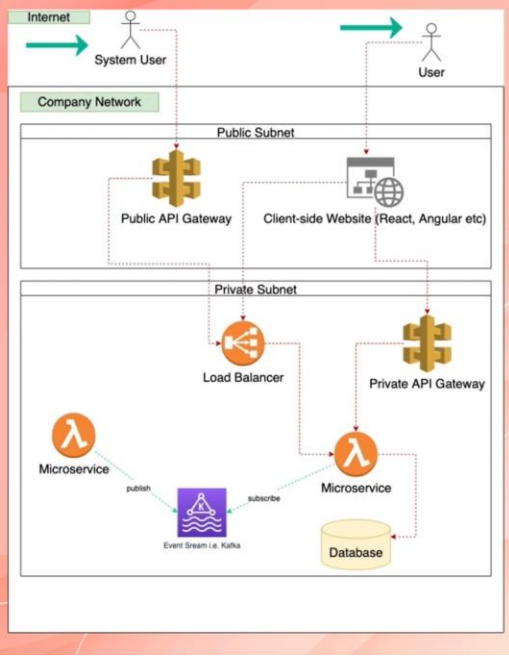
Anatomy of a micro-service-based system



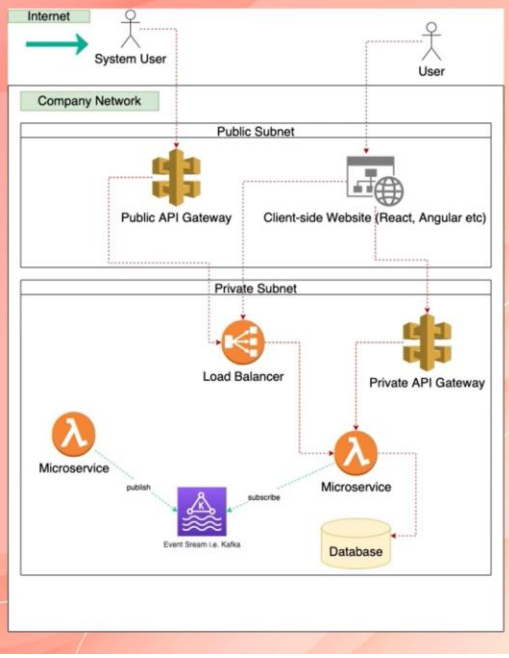
Anatomy of a micro-service-based system



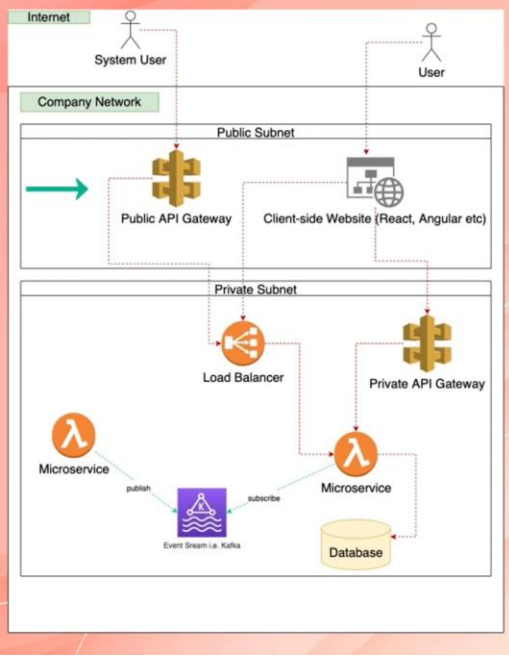
Anatomy of a micro-service-based system



Anatomy of a micro-service-based system

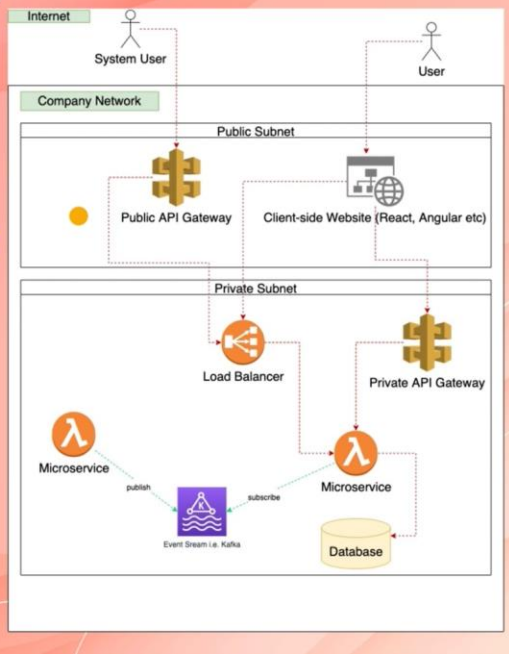


Anatomy of a micro-service-based system



Anatomy of a micro-service-based system

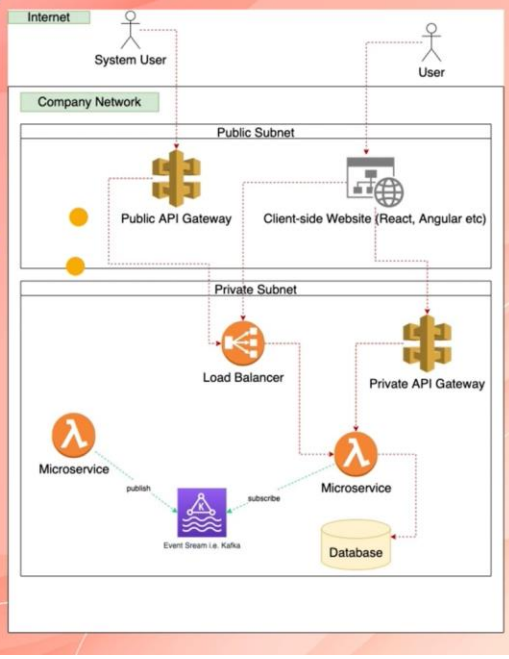
ApiGee



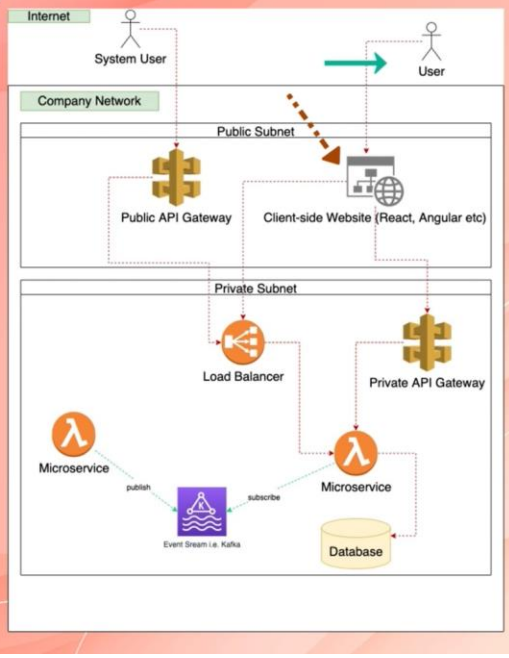
Anatomy of a micro-service-based system

ApiGee

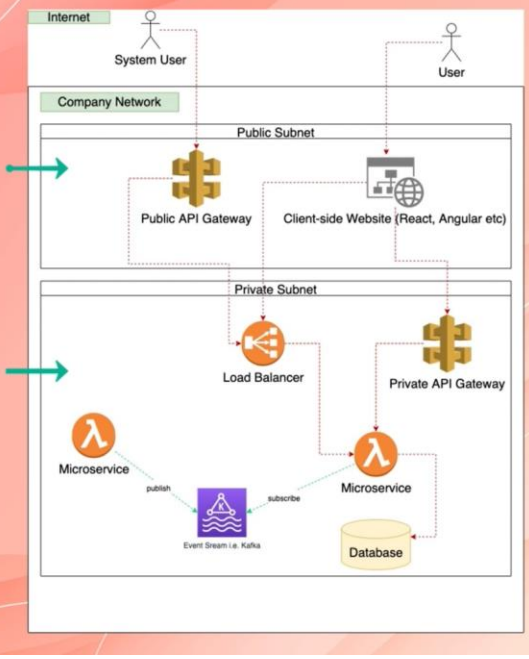
Kong



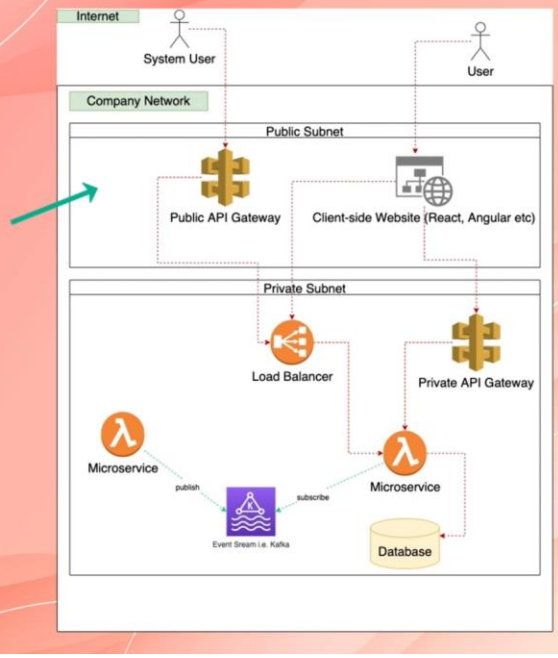
Anatomy of a micro-service-based system



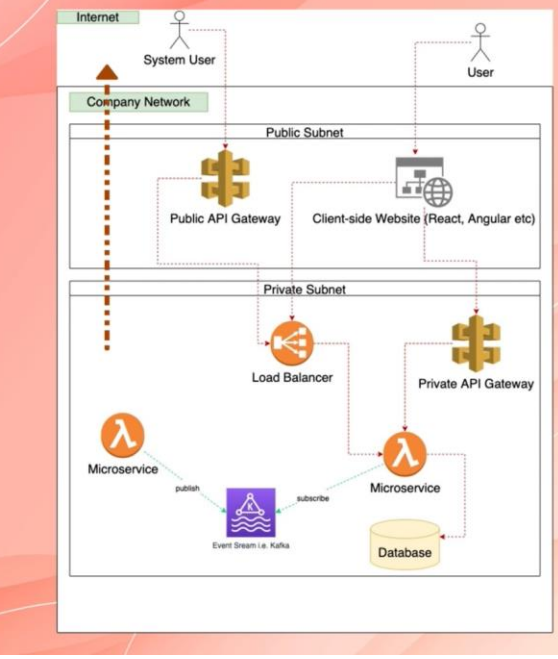
Anatomy of a micro-service-based system



Anatomy of a micro-service-based system

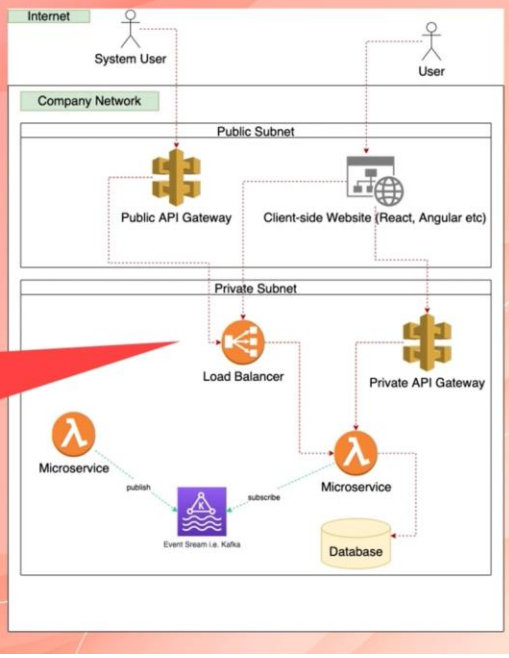


Anatomy of a micro-service-based system

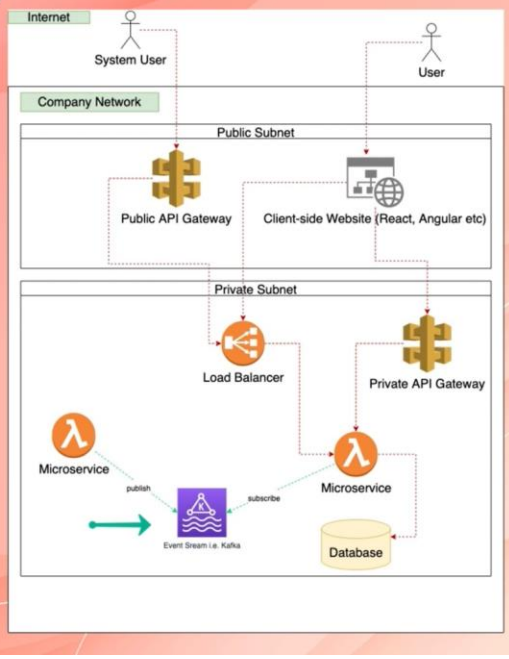


Anatomy of a micro-service-based system

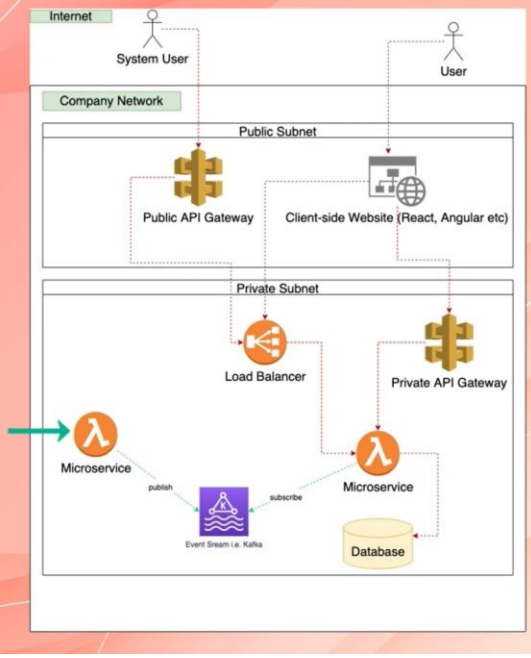
May not be needed for Server-Less Microservices. Good to have as an abstraction layer



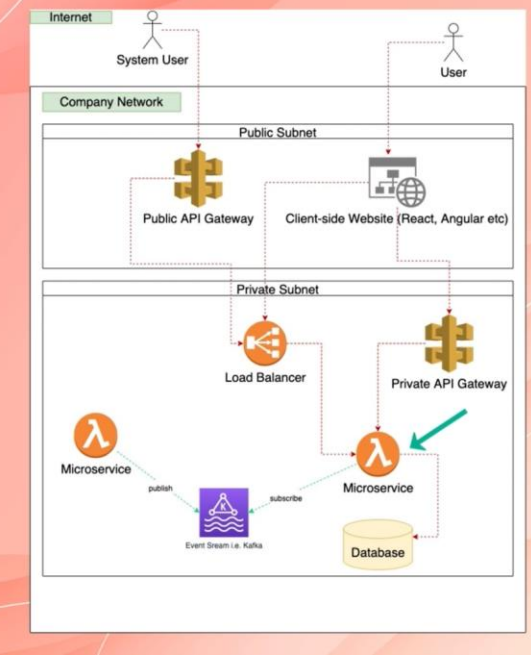
Anatomy of a micro-service-based system

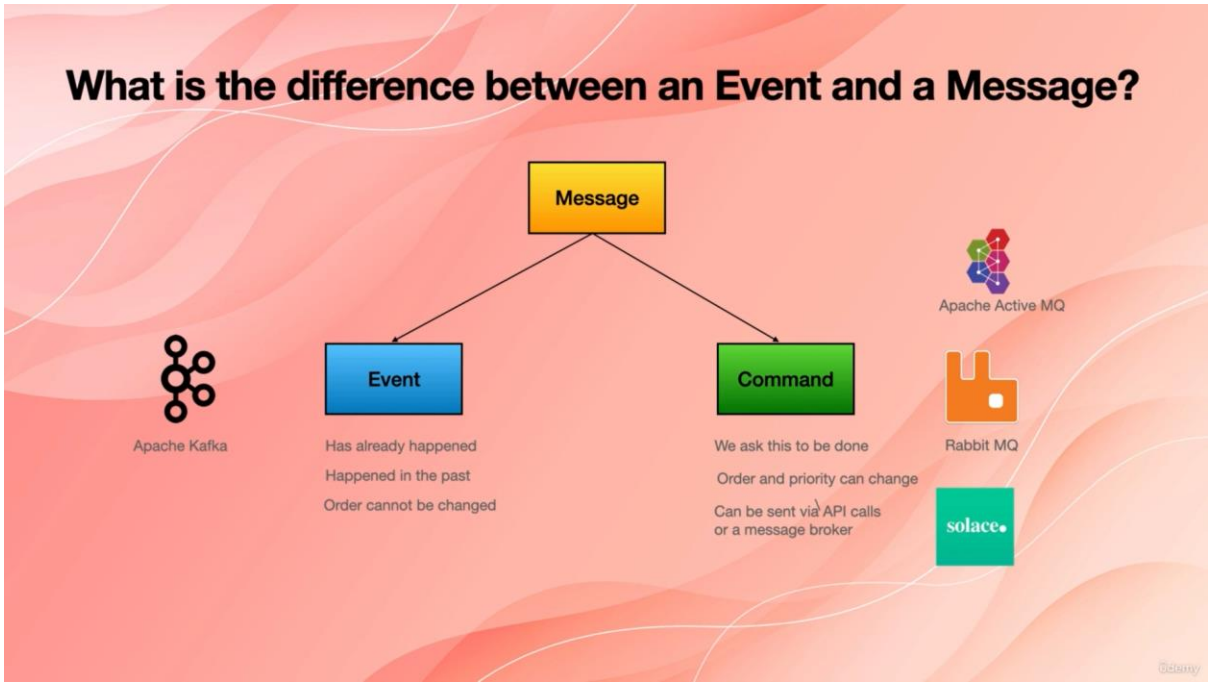


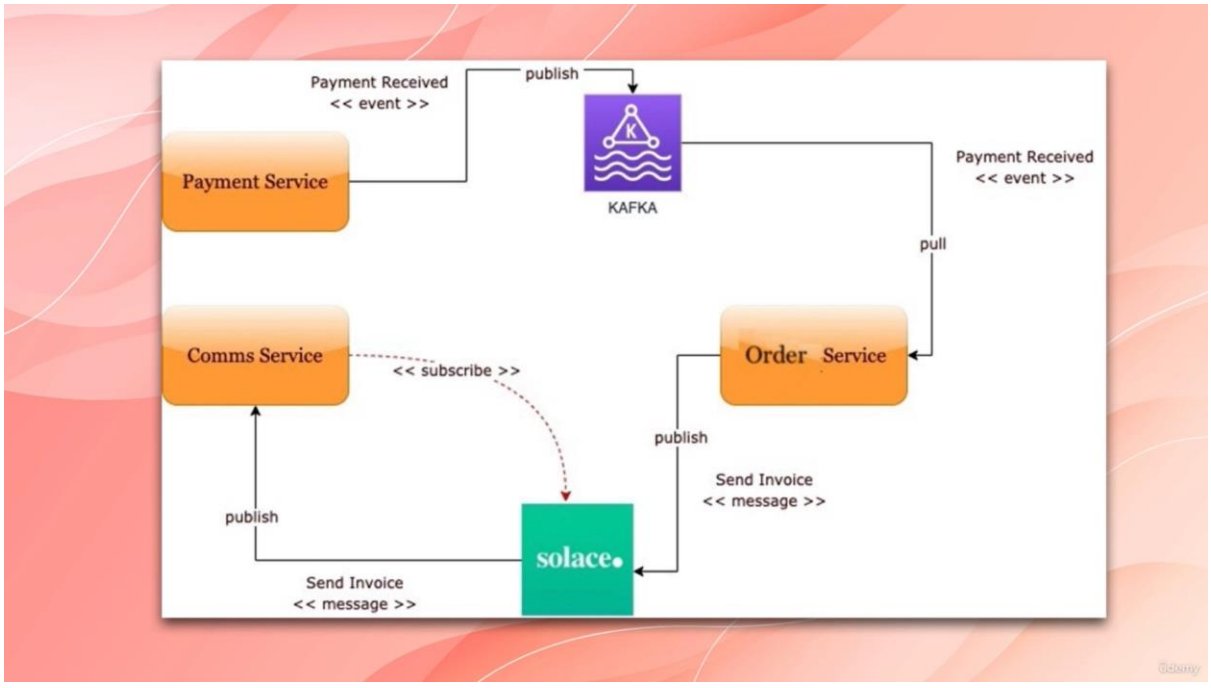
Anatomy of a micro-service-based system

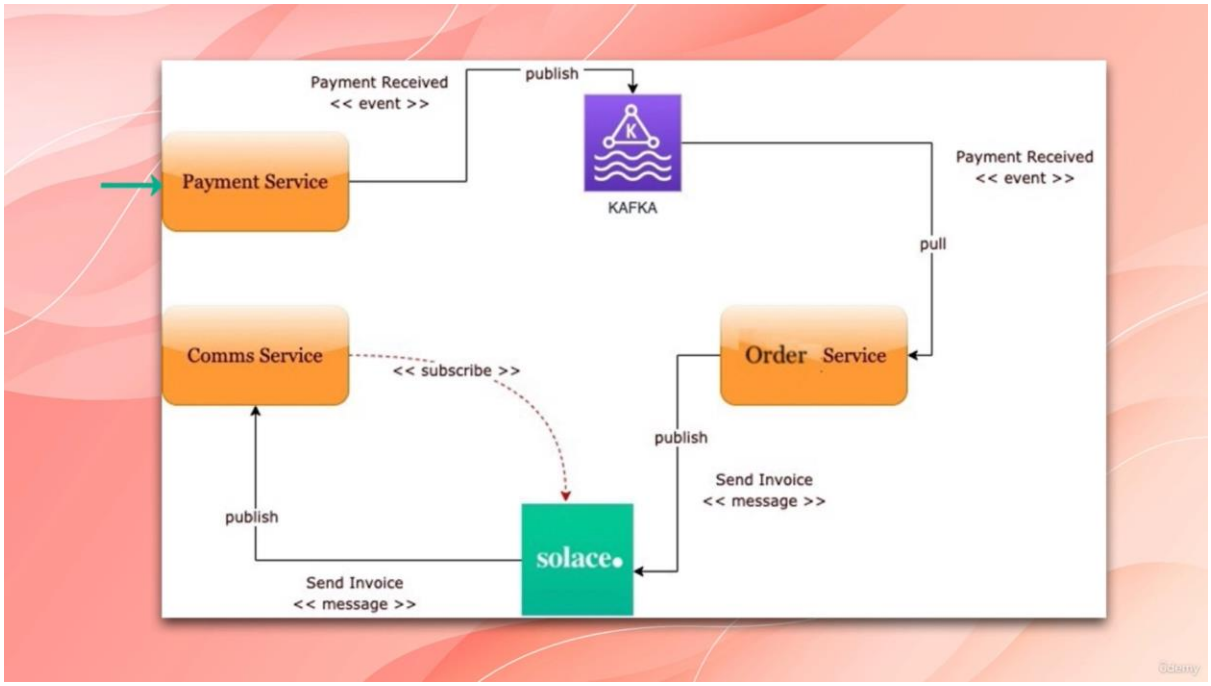


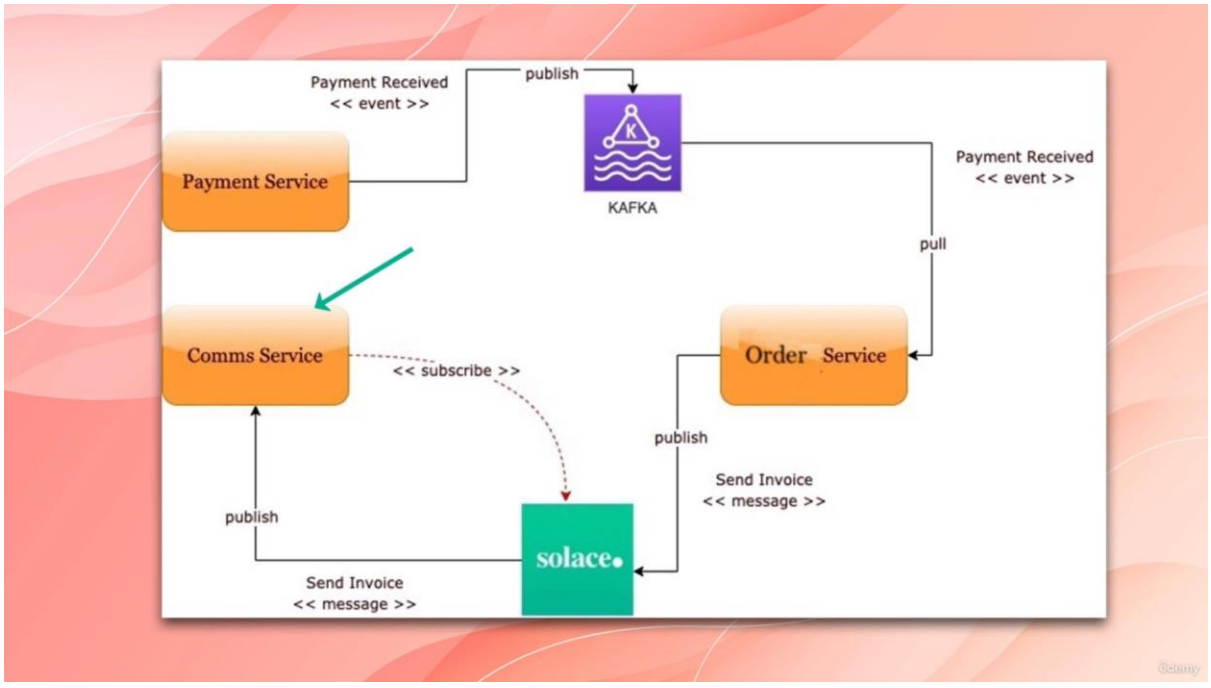
Anatomy of a micro-service-based system

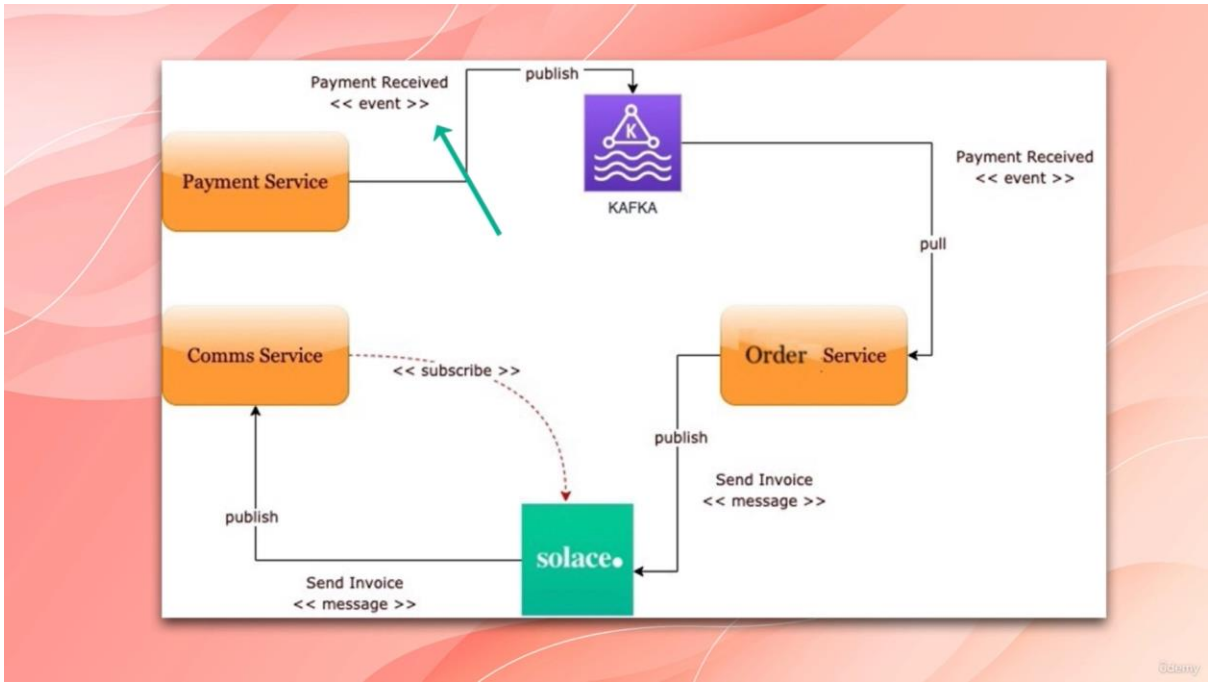


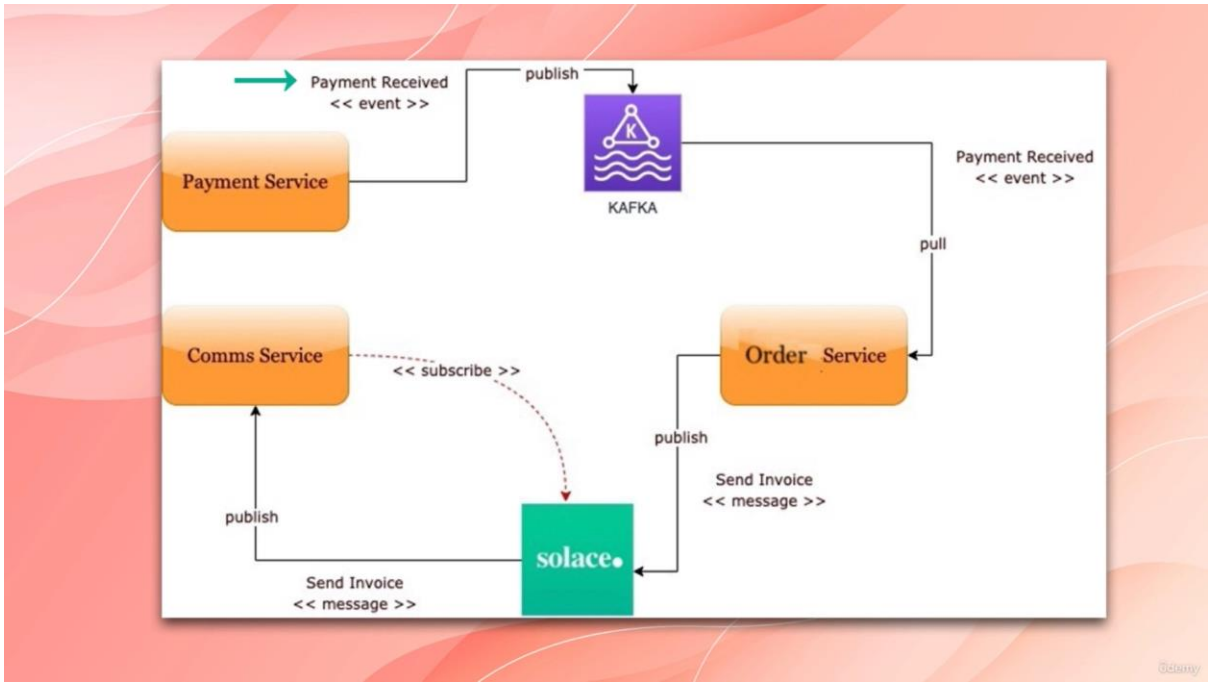

What is the difference between an Event and a Message?
Message
Event
Command
Apache Kafka
Apache Active MQ
Rabbit MQ
solace
Has already happened
Happened in the past
Order cannot be changed
We ask this to be done
Order and priority can change
Can be sent via API calls
or a message broker

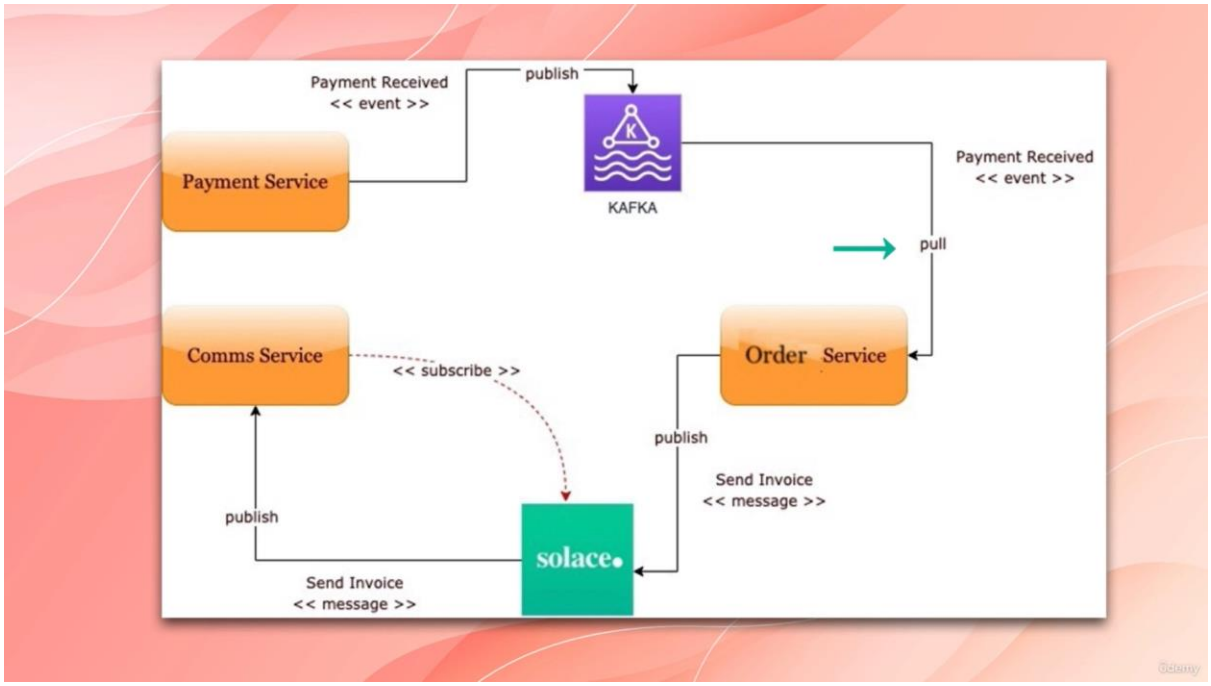

Payment Received
<< event >>
publish
Payment Service
KAFKA
Payment Received
<< event >>
pull
Comms Service
<< subscribe >>
Order Service
publish
Send Invoice
<< message >>
publish
solace
Send Invoice
<< message >>

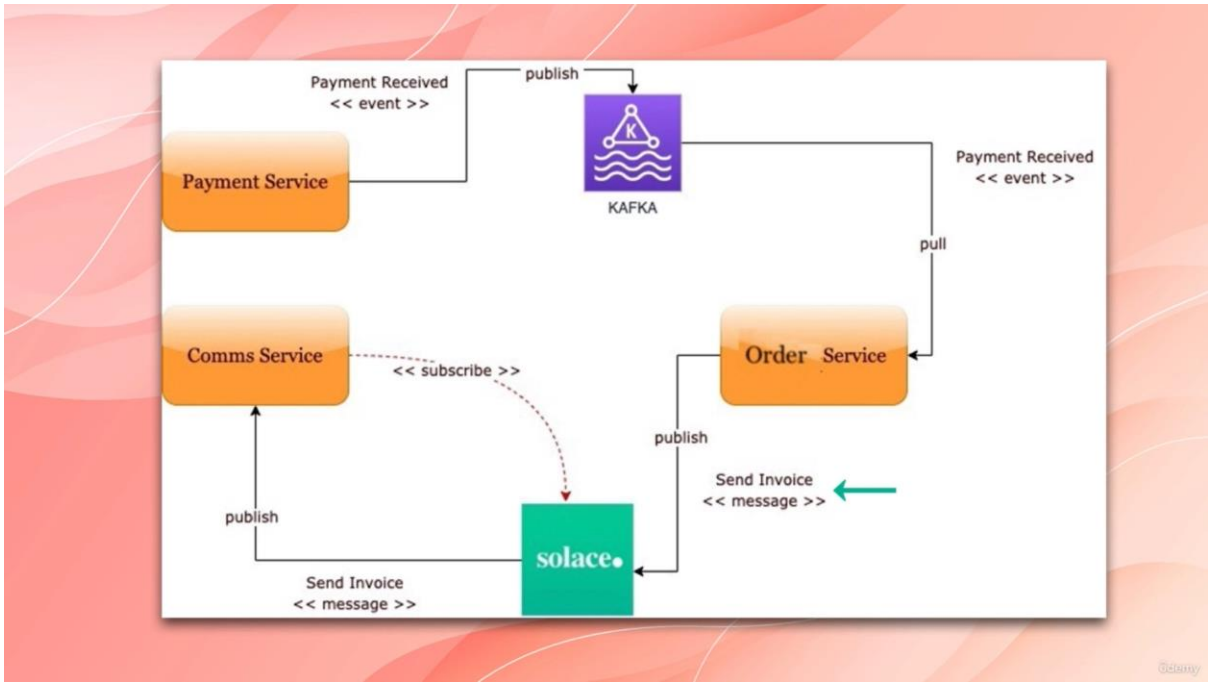

Payment Received
<< event >>
publish
Payment Service
KAFKA
Payment Received
<< event >>
pull
Comms Service
<< subscribe >>
Order Service
publish
Send Invoice
<< message >>
publish
solace.
Send Invoice
<< message >>

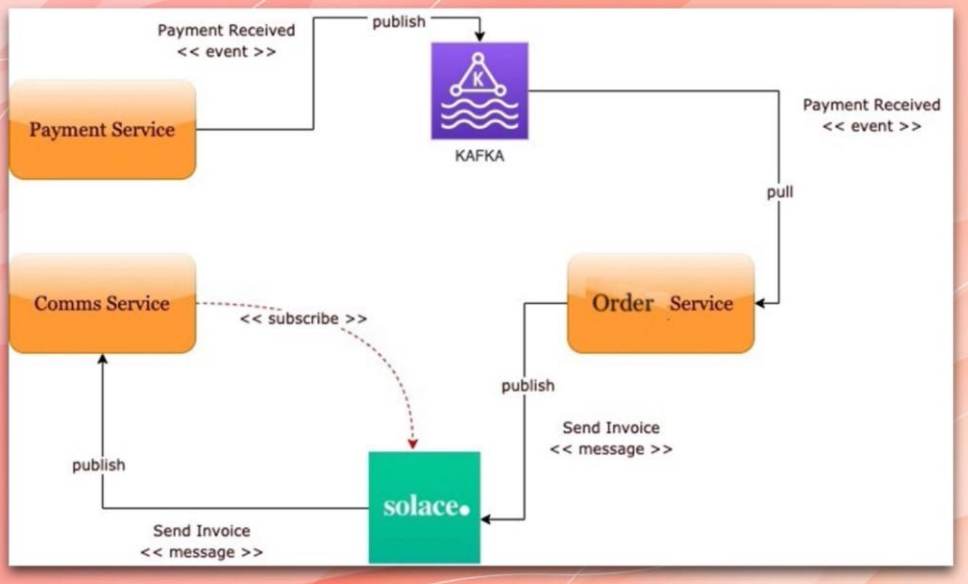

Payment Received
<< event >>
publish
Payment Service
KAFKA
Payment Received
<< event >>
pull
Comms Service
<< subscribe >>
Order Service
publish
Send Invoice
<< message >>
publish
solace.
Send Invoice
<< message >>


Payment Received
<< event >>
publish
Payment Service
KAFKA
Payment Received
<< event >>
pull
Comms Service
<< subscribe >>
Order Service
publish
Send Invoice
<< message >>
publish
solace.
Send Invoice
<< message >>


Payment Received
<< event >>
publish
Payment Service
KAFKA
Payment Received
<< event >>
pull
Comms Service
<< subscribe >>
Order Service
publish
Send Invoice
<< message >>
publish
solace.
Send Invoice
<< message >>

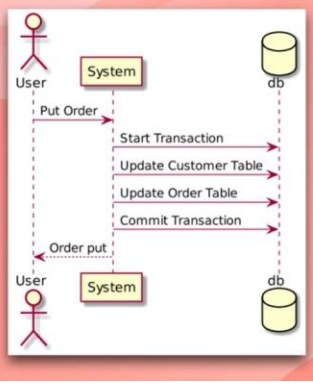

Payment Received
<< event >>
publish
Payment Service
KAFKA
Payment Received
<< event >>
pull
Comms Service
<< subscribe >>
Order Service
publish
Send Invoice
<< message >>
publish
solace.
Send Invoice
<< message >>


Payment Received
<< event >>
publish
Payment Service
KAFKA
Payment Received
<< event >>
pull
Comms Service
<< subscribe >>
Order Service
publish
Send Invoice
<< message >>
publish
solace.
Send Invoice
<< message >>



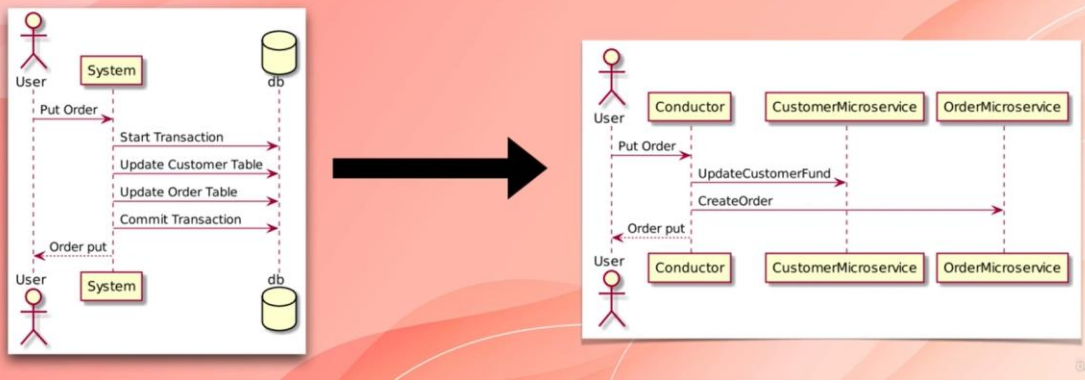
What is a Distributed Transaction?

- In Microservice-based architecture each service has its own database
- We need to make sure transactions (insert, update, delete) are ATOMIC across all services
- Managing transactions across multiple services is Distributed Transactions



What is a Distributed Transaction?

- In Microservice-based architecture each service has its own database
- We need to make sure transactions (insert, update, delete) are ATOMIC across all services
- Managing transactions across multiple services is Distributed Transactions



How is Distributed Transactions Handled?

- There are two patterns for handling distributed transactions
  - Two-Phase Commit Transactions (2PC) (not recommended)
  - Saga Pattern

Two-Phase Commit Pattern (2PC)

- Widely used in databases
- Sometimes useful in Microservices (only a few services in the system)
- It has a Prepare phase and a Commit phase

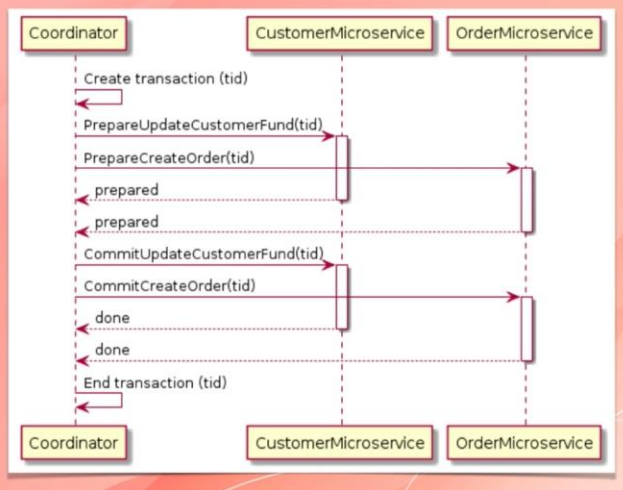
Two-Phase Commit Pattern (2PC)

- Widely used in databases
- Sometimes useful in Microservices (only a few services in the system)
- It has a Prepare phase and a Commit phase

Two-Phase Commit Pattern (2PC)

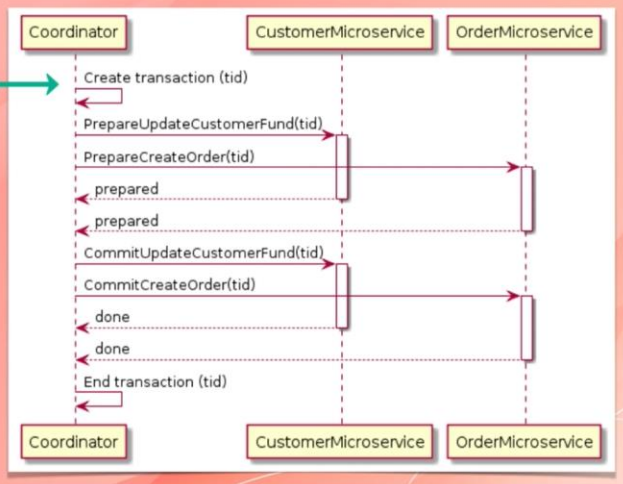
- Widely used in databases
- Sometimes useful in Microservices (only a few services in the system)
- It has a Prepare phase and a Commit phase
    - Microservices are asked to prepare for change
      Create a record in the database and set its Status field or column to 0 (0 means not stable)
    - Microservices are asked make the final and actual change
      Set the Status field or column to 1 (finalised)
- A Coordinator microservices is needed to maintain the life-cycle of the transaction

Successful Two-Phase Commit

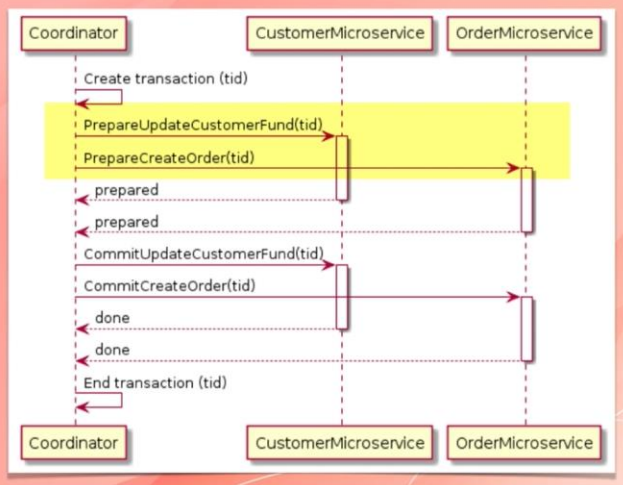


Returns an ID

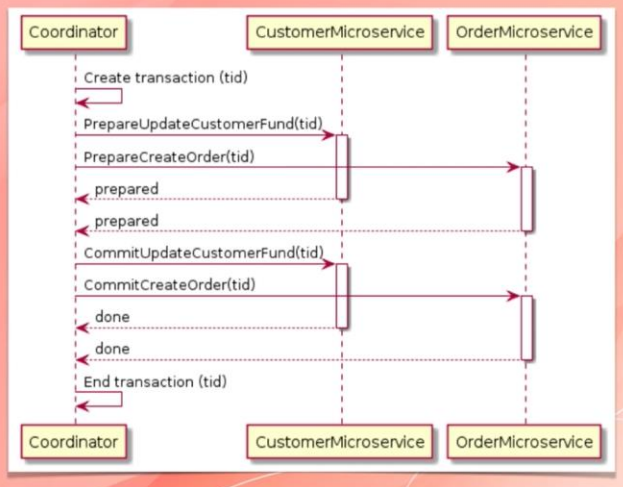
Successful Two-Phase Commit



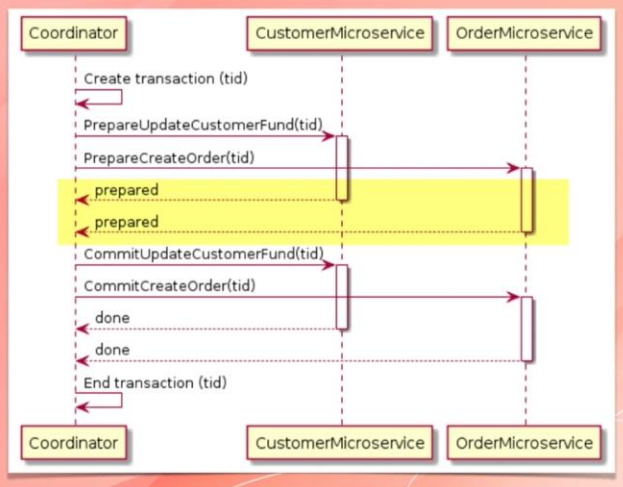
Successful Two-Phase Commit



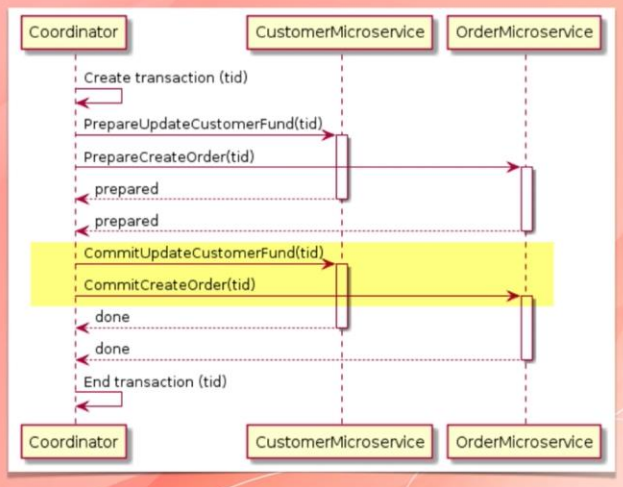
Successful Two-Phase Commit



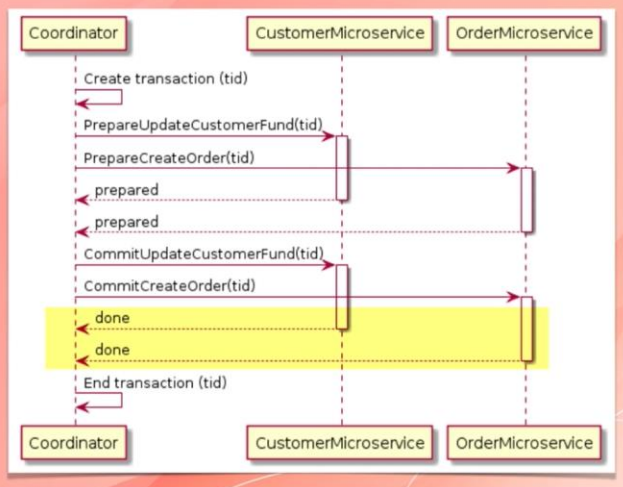
Successful Two-Phase Commit



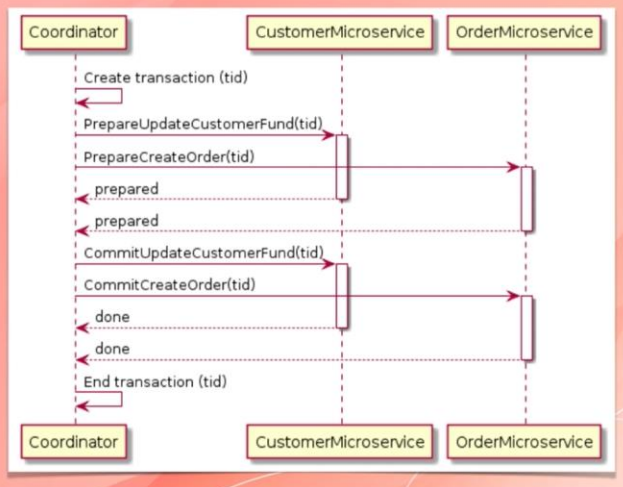
Successful Two-Phase Commit



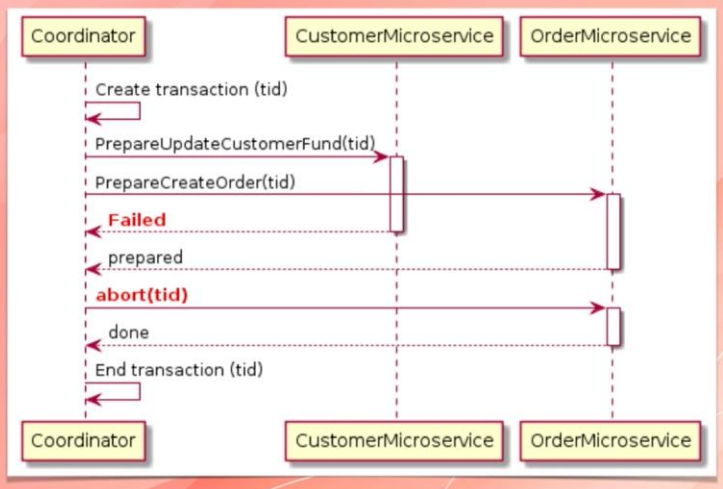
Successful Two-Phase Commit



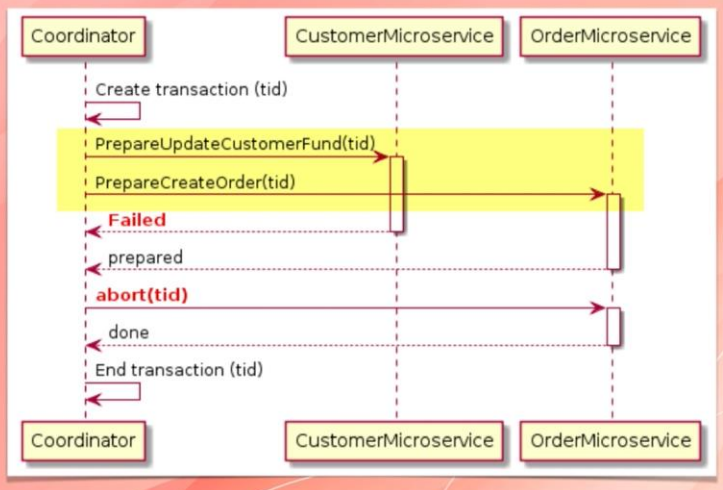
Successful Two-Phase Commit



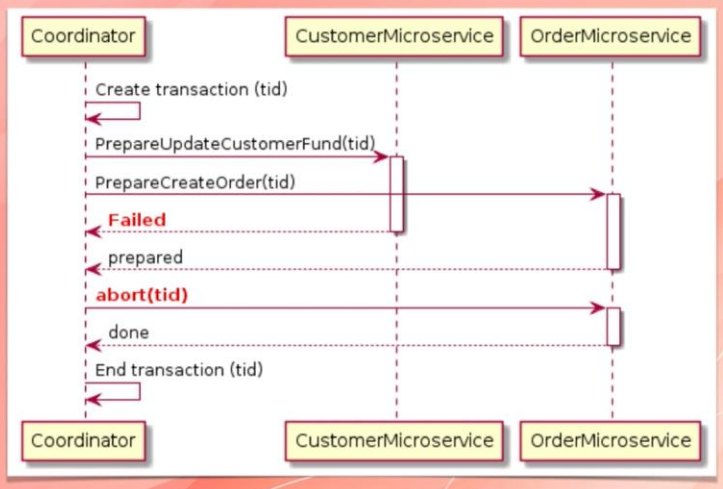
Failed Two-Phase Commit



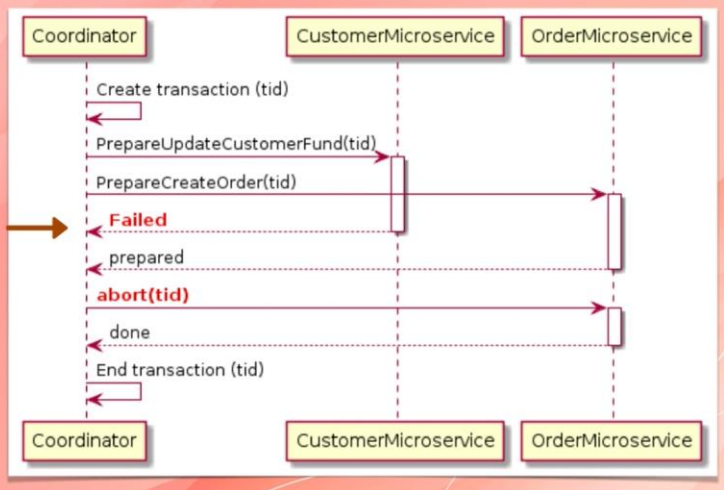
Failed Two-Phase Commit



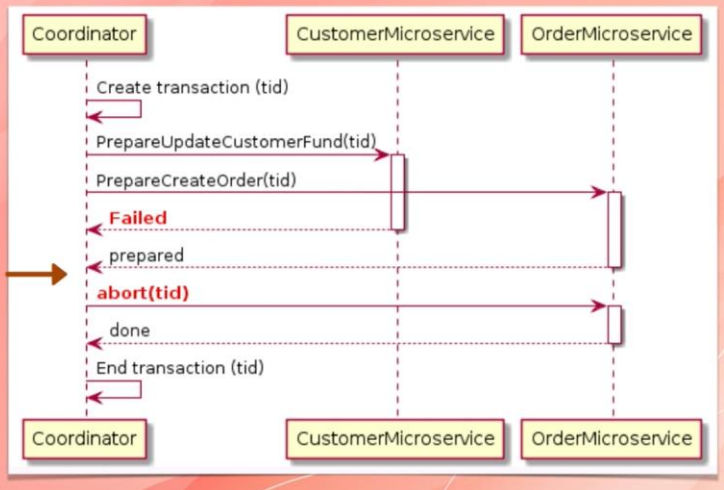
Failed Two-Phase Commit



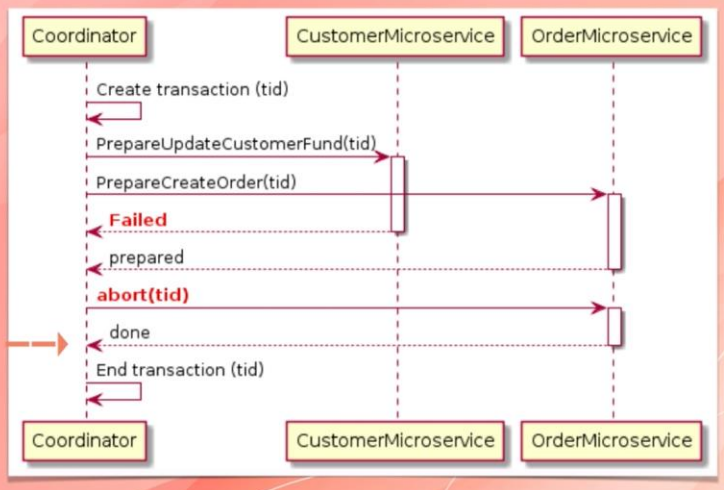
Failed Two-Phase Commit



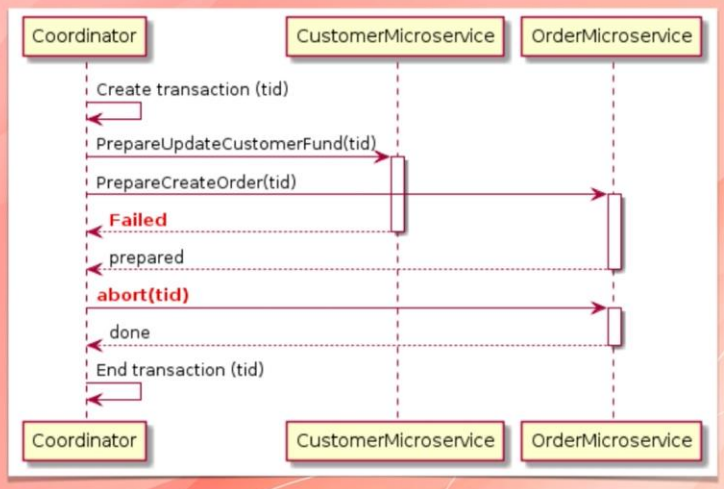
Failed Two-Phase Commit



Failed Two-Phase Commit



Failed Two-Phase Commit



Saga Pattern

For Distributed Transactions

- Asynchronous

2PC

Saga Pattern

For Distributed Transactions

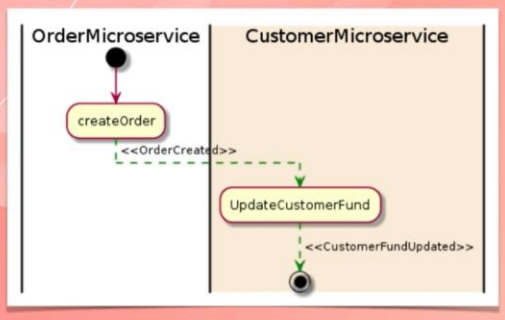
- Asynchronous
- Each micro service handles its own transaction
- There are two sub-patterns
  - Choreography patterns

    If something goes wrong, micro services have to tell the up-stream (previous) micro services to Roll Back the transaction

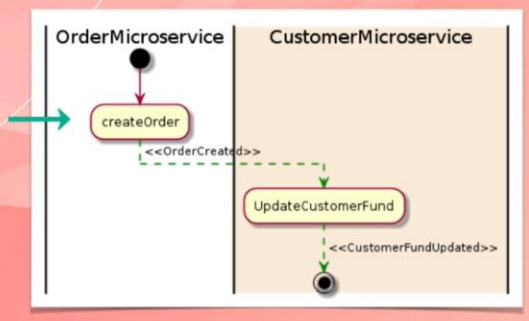
  - Orchestration patterns

    A central micro service delivers the messages for roll-backs

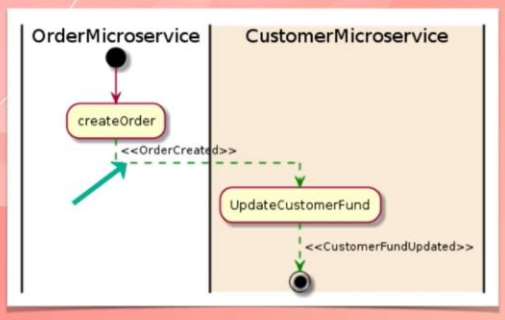
Choreography Pattern



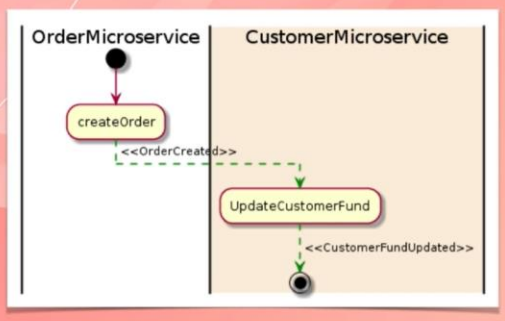
Choreography Pattern

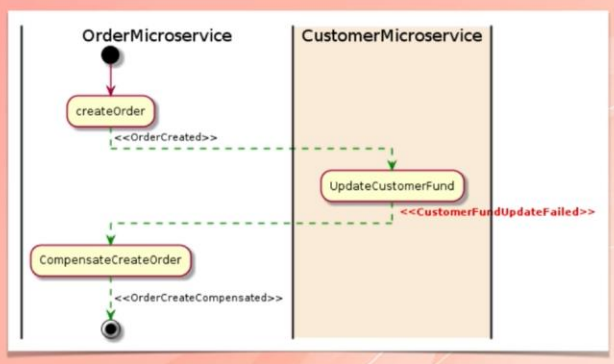


Choreography Pattern

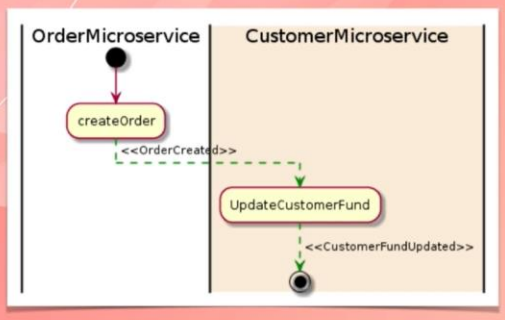


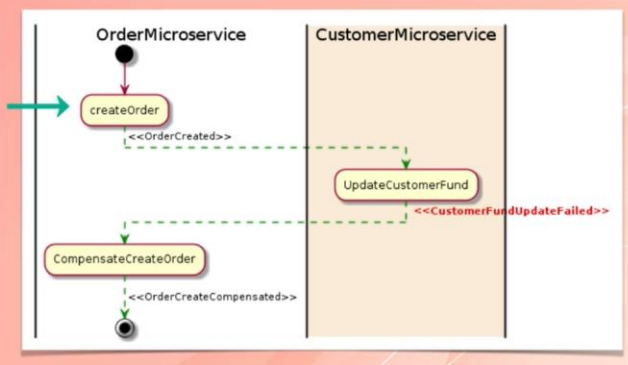
Choreography Pattern



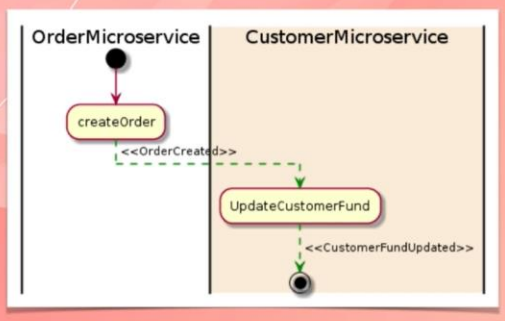


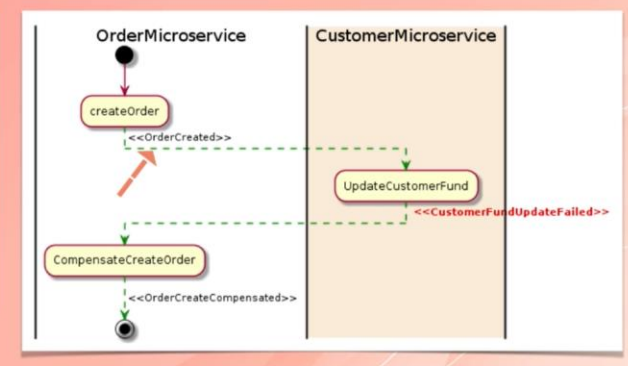
Choreography Pattern



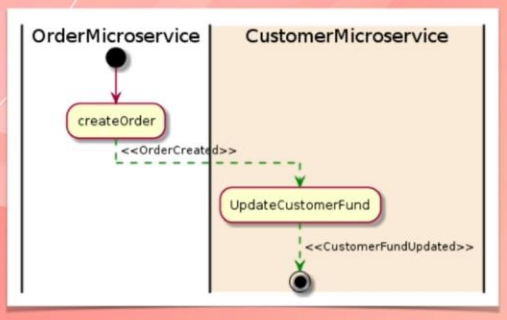


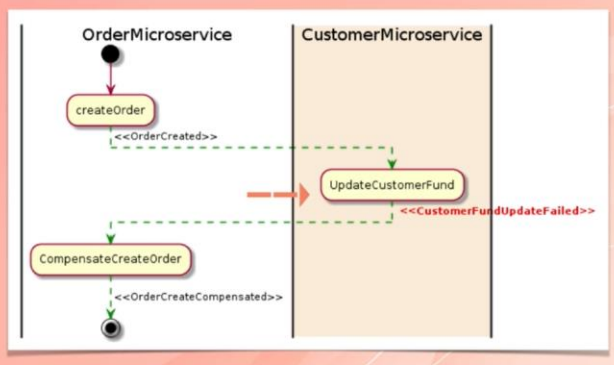
Choreography Pattern



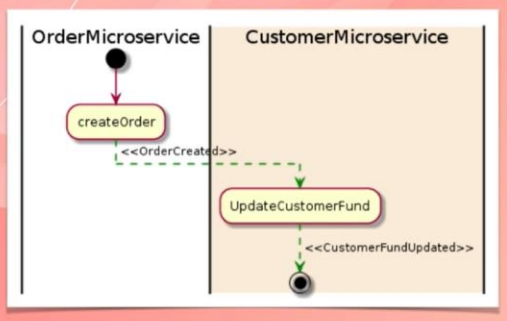


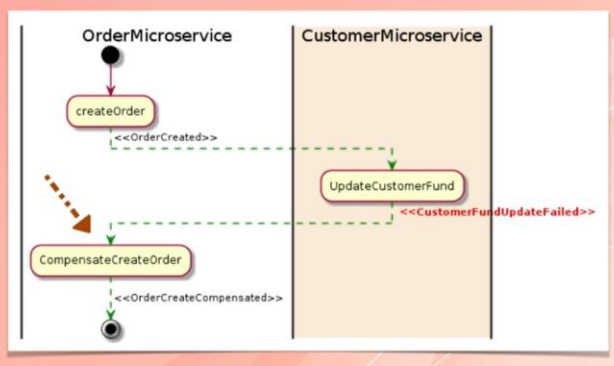
Choreography Pattern



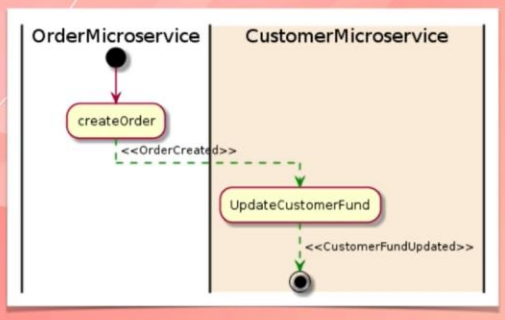


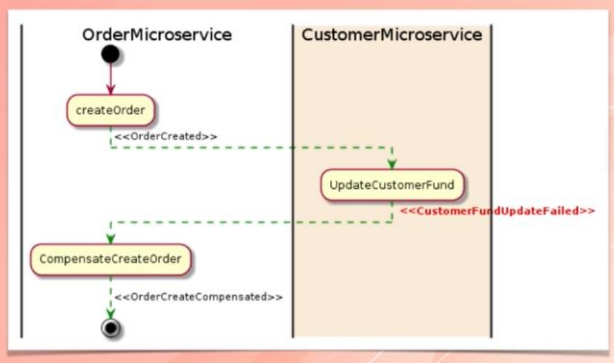
Choreography Pattern



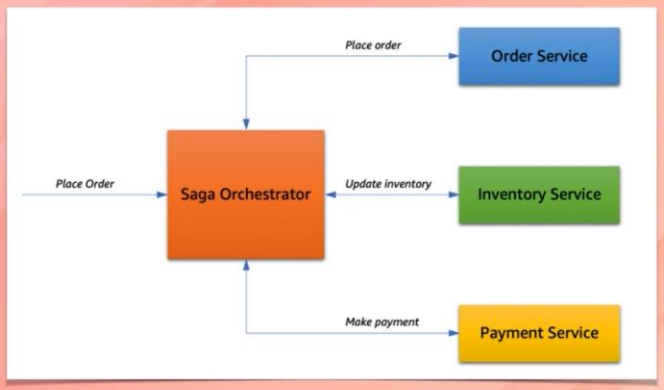


Choreography Pattern





Orchestration Pattern



What are the differences between Monolithic, SOA and Microservices

Monolith	SOA	Microservice
Application is made of a single code base	Tries to break down the monolith app to smaller components	Microservices are more fine-grained and smaller compared to SOA
Application runs as a single process and entire code shares the same memory	Services communicate via Synchronous API calls (SOAP or REST)	Microservices communicate via Rest APIs, Events and Messages
All functions in the code share the same database	In SOA often services share the same database although not a best practice	Each service has its dedicated database

What are the differences between Monolithic, SOA and Microservices

Monolith	SOA	Microservice
Application is made of a single code base	Tries to break down the monolith app to smaller components	Microservices are more fine-grained and smaller compared to SOA
Application runs as a single process and entire code shares the same memory	Services communicate via Synchronous API calls (SOAP or REST)	Microservices communicate via Rest APIs, Events and Messages
All functions in the code share the same database	In SOA often services share the same database although not a best practice	Each service has its dedicated database

Udemy

What are the differences between Monolithic, SOA and Microservices

Monolith	SOA	Microservice
Application is made of a single code base	Tries to break down the monolith app to smaller components	Microservices are more fine-grained and smaller compared to SOA
Application runs as a single process and entire code shares the same memory	Services communicate via Synchronous API calls (SOAP or REST)	Microservices communicate via Rest APIs, Events and Messages
All functions in the code share the same database	In SOA often services share the same database although not a best practice	Each service has its dedicated database

What are the differences between Monolithic, SOA and Microservices

Monolith	SOA	Microservice
Application is made of a single code base	Tries to break down the monolith app to smaller components	Microservices are more fine-grained and smaller compared to SOA
Application runs as a single process and entire code shares the same memory	Services communicate via Synchronous API calls (SOAP or REST)	Microservices communicate via Rest APIs, Events and Messages
All functions in the code share the same database	In SOA often services share the same database although not a best practice	Each service has its dedicated database

What are the differences between Monolithic, SOA and Microservices

Monolith	SOA	Microservice
Application is made of a single code base	Tries to break down the monolith app to smaller components	Microservices are more fine-grained and smaller compared to SOA
Application runs as a single process and entire code shares the same memory	Services communicate via Synchronous API calls (SOAP or REST)	Microservices communicate via Rest APIs, Events and Messages
All functions in the code share the same database	In SOA often services share the same database although not a best practice	Each service has its dedicated database

What are the differences between Monolithic, SOA and Microservices

Monolith	SOA	Microservice
Application is made of a single code base	Tries to break down the monolith app to smaller components	Microservices are more fine-grained and smaller compared to SOA
Application runs as a single process and entire code shares the same memory	Services communicate via Synchronous API calls (SOAP or REST)	Microservices communicate via Rest APIs, Events and Messages
All functions in the code share the same database	In SOA often services share the same database although not a best practice	Each service has its dedicated database

What are the differences between Monolithic, SOA and Microservices

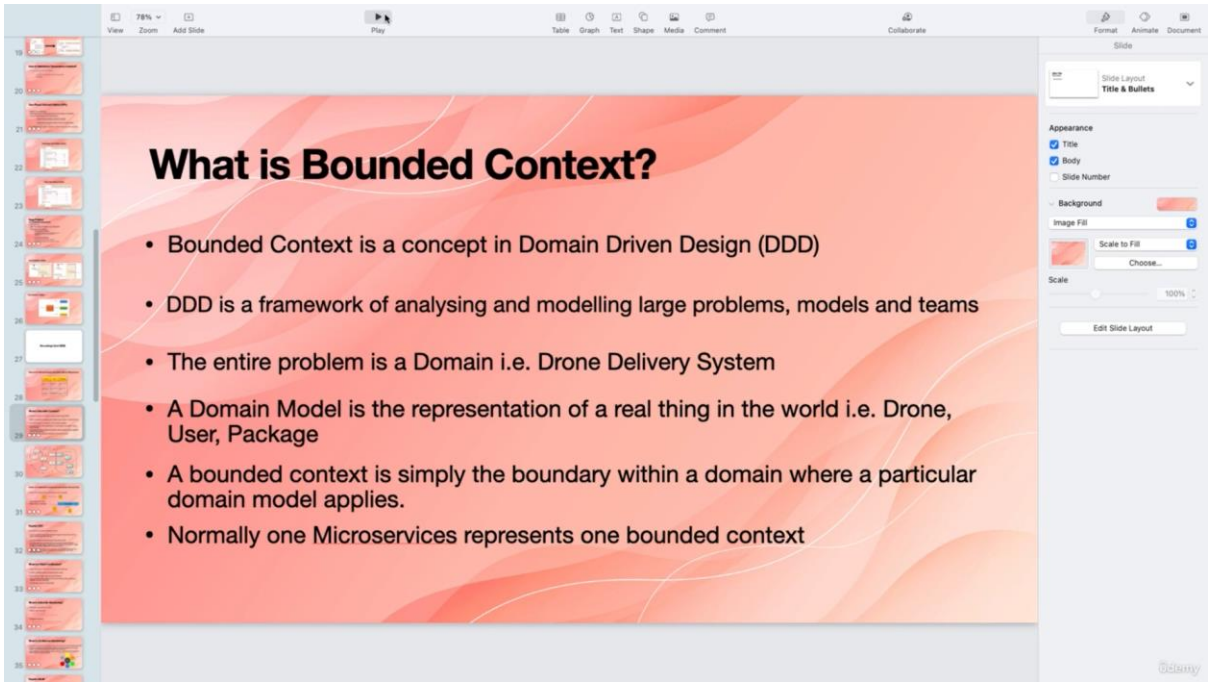
Monolith	SOA	Microservice
Application is made of a single code base	Tries to break down the monolith app to smaller components	Microservices are more fine-grained and smaller compared to SOA
Application runs as a single process and entire code shares the same memory	Services communicate via Synchronous API calls (SOAP or REST)	Microservices communicate via Rest APIs, Events and Messages
All functions in the code share the same database	In SOA often services share the same database although not a best practice	Each service has its dedicated database

What are the differences between Monolithic, SOA and Microservices

Monolith	SOA	Microservice
Application is made of a single code base	Tries to break down the monolith app to smaller components	Microservices are more fine-grained and smaller compared to SOA
Application runs as a single process and entire code shares the same memory	Services communicate via Synchronous API calls (SOAP or REST)	Microservices communicate via Rest APIs, Events and Messages
All functions in the code share the same database	In SOA often services share the same database although not a best practice	Each service has its dedicated database

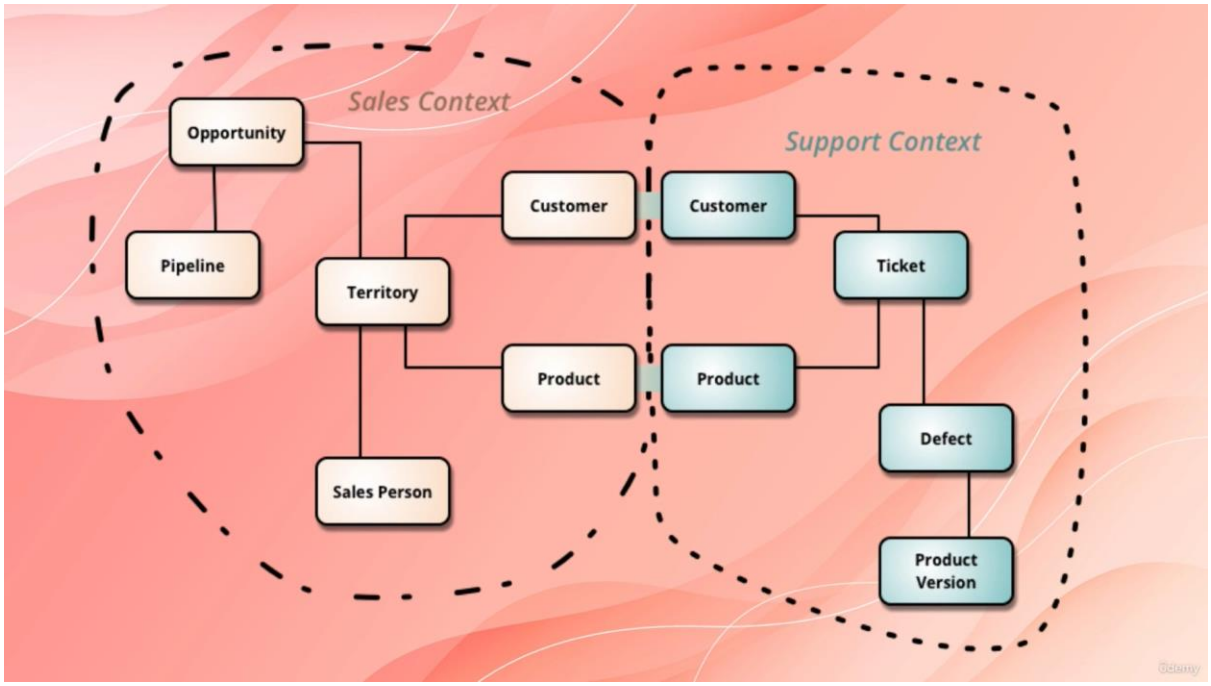
What are the differences between Monolithic, SOA and Microservices

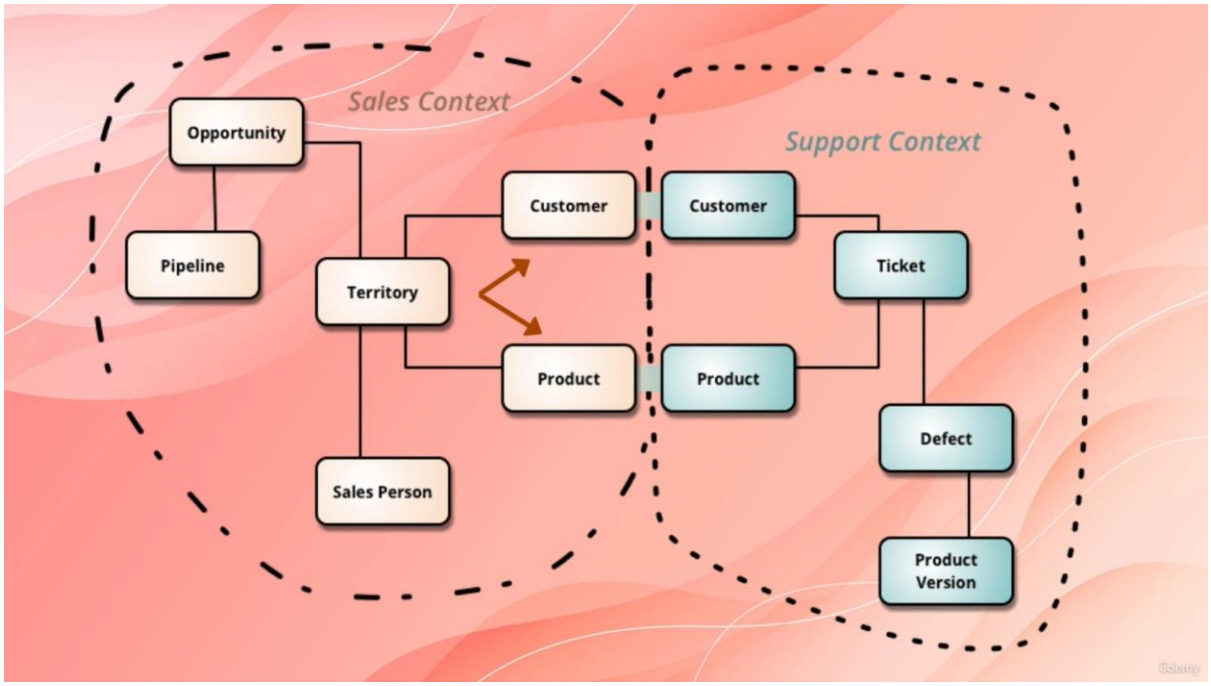
Monolith	SOA	Microservice
Application is made of a single code base	Tries to break down the monolith app to smaller components	Microservices are more fine-grained and smaller compared to SOA
Application runs as a single process and entire code shares the same memory	Services communicate via Synchronous API calls (SOAP or REST)	Microservices communicate via Rest APIs, Events and Messages
All functions in the code share the same database	In SOA often services share the same database although not a best practice	Each service has its dedicated database

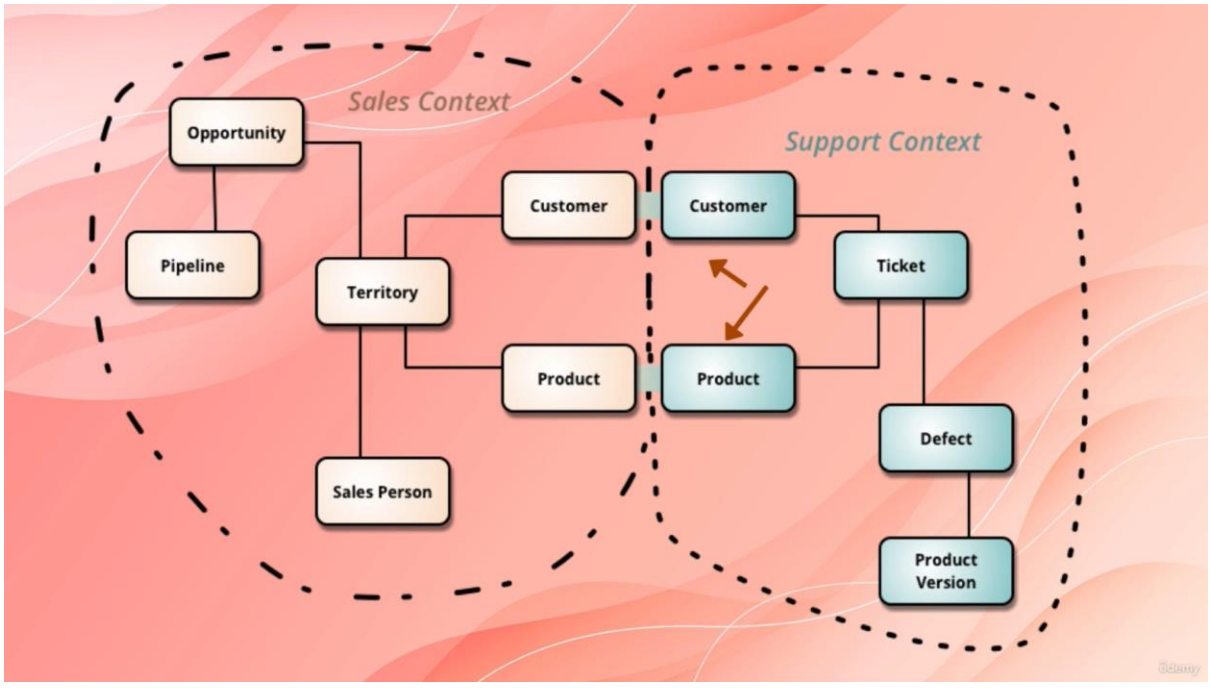


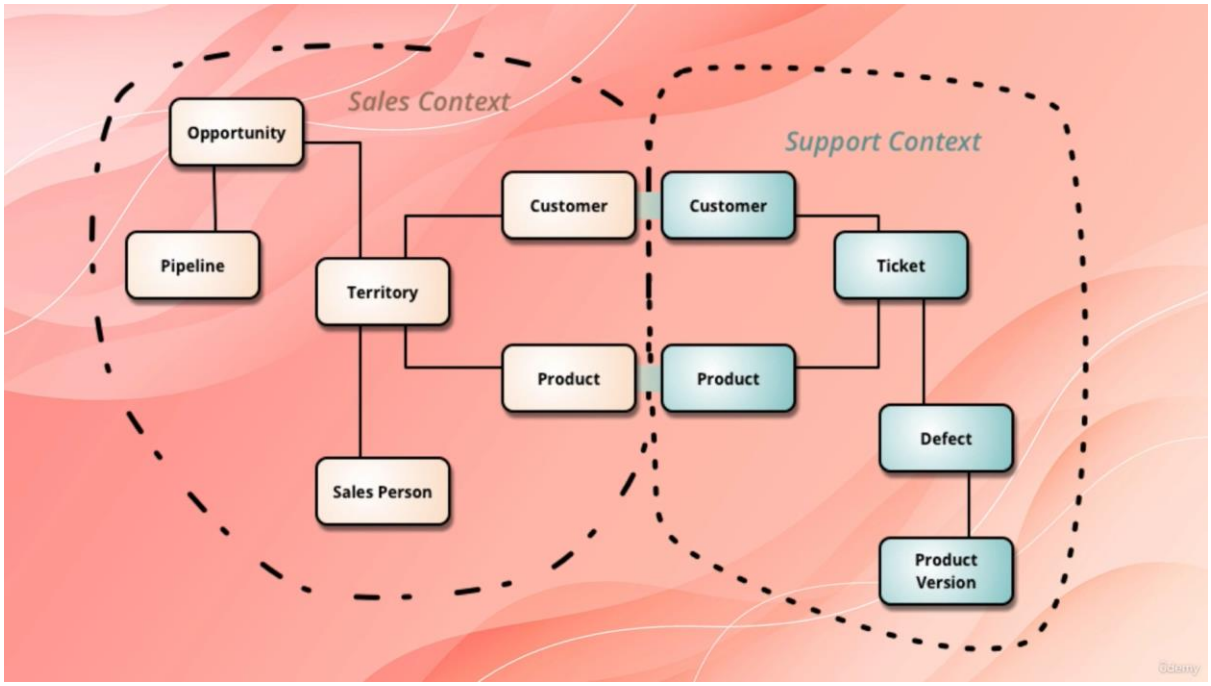
What is Bounded Context?

- Bounded Context is a concept in Domain Driven Design (DDD)
- DDD is a framework of analysing and modelling large problems, models and teams
- The entire problem is a Domain i.e. Drone Delivery System
- A Domain Model is the representation of a real thing in the world i.e. Drone, User, Package
- A bounded context is simply the boundary within a domain where a particular domain model applies.
- Normally one Microservices represents one bounded context



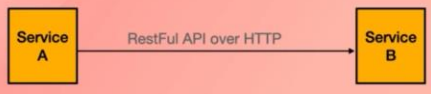

Sales Context
Support Context
Opportunity
Pipeline
Territory
Customer
Product
Sales Person
Customer
Ticket
Product
Defect
Product Version


Sales Context
Support Context
Opportunity
Pipeline
Territory
Customer
Product
Sales Person
Customer
Ticket
Product
Defect
Product Version



Explain how independent microservices communicate with each other

- Point-to-Point & Synchronous API calls (not recommended)



- Responding to an Event
- Responding to a Message

Explain CDC

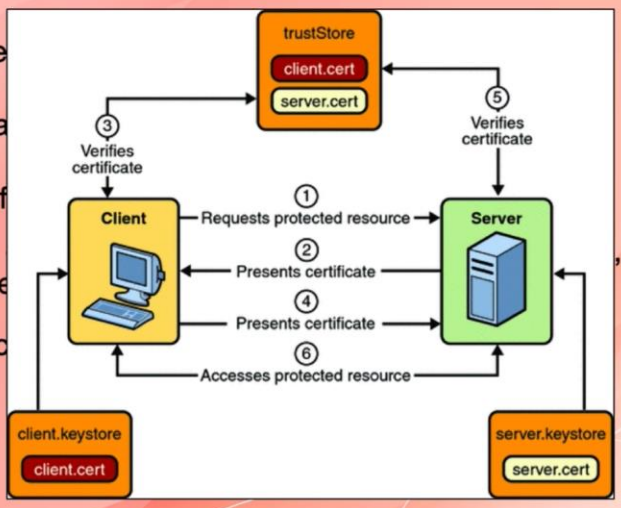
- CDC stands for Consumer Defined Contract
- It's a kind of test to ensure that constant changes in micro service will not break the dependent Microservices
- We have Provider (API) and Consumer (that uses the API)
- The consumer has to write the test for the contract between it and the Provider. In the pre-stage of deployment, Consumer will run the tests. Failing to pass the contract, deployment is not taken further
- The Provider also runs the consumer test on its side before deploying it to make sure that its changes won't affect the consumers to whom it is providing the service.

What are Client Certificates?

- Client Certificate is a method for authenticating the API user
- A Client Certificate File (public & private key pair) is used
- No encryption of data is done by Client Certificate
- When API caller (client) makes an API call (connects to server), the server validates the Client Certificate
- This process is called TLS-hand shake

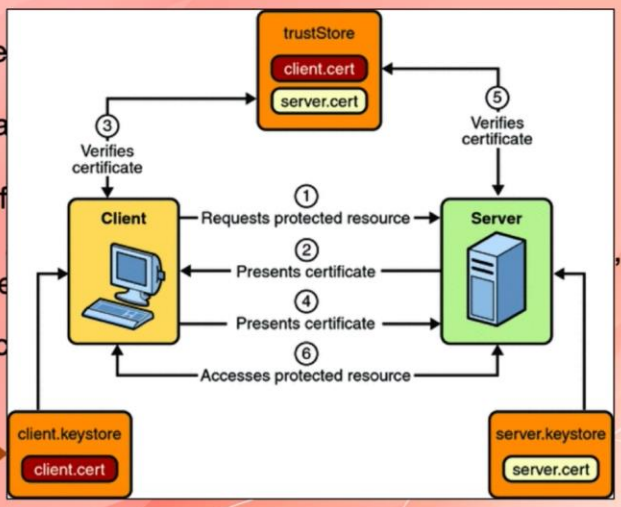
What are Client Certificates?

- Client Certificate
- A Client Certifica
- No encryption of
- When API caller , the server validates the Clie
- This process is c



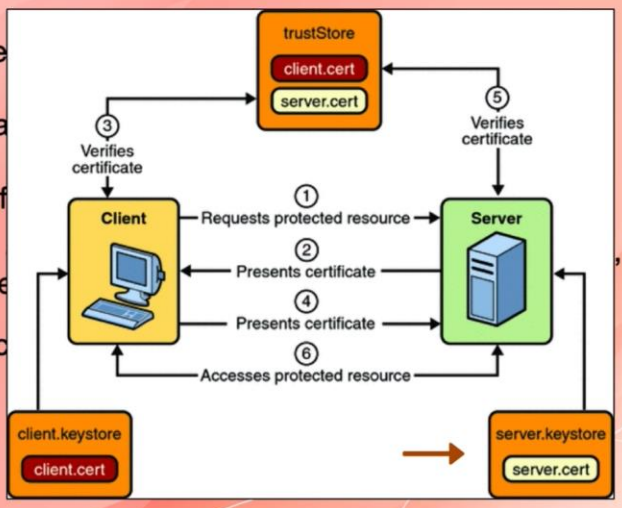
What are Client Certificates?

- Client Certificate
- A Client Certifica
- No encryption of
- When API caller , the server validates the Clie
- This process is c



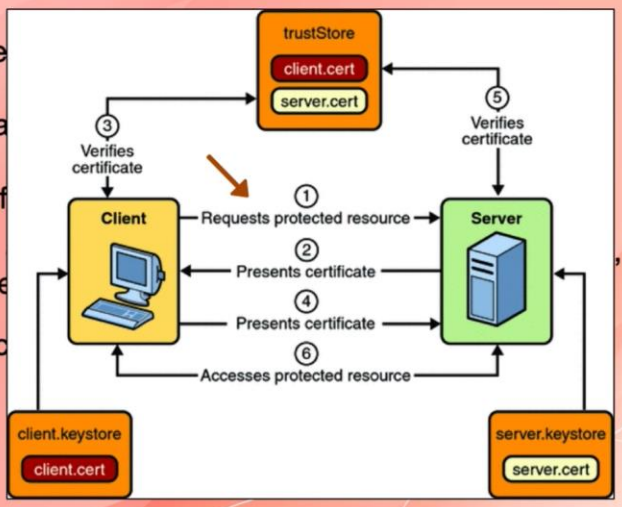
What are Client Certificates?

- Client Certificate
- A Client Certifica
- No encryption of
- When API caller , the server validates the Clie
- This process is c



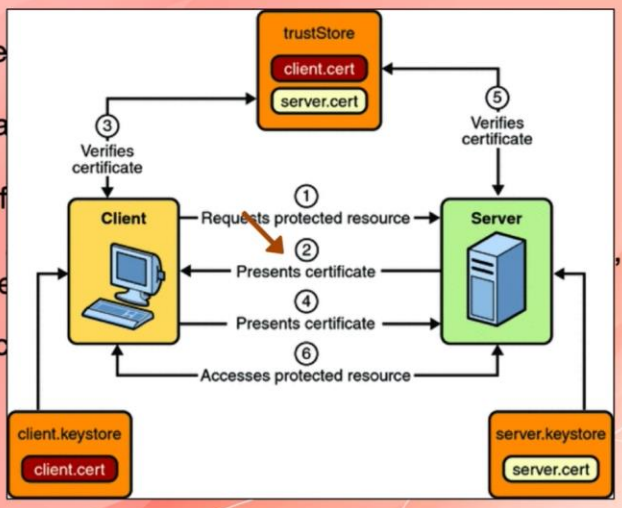
What are Client Certificates?

- Client Certificate
- A Client Certifica
- No encryption of
- When API caller , the server validates the Clie
- This process is c



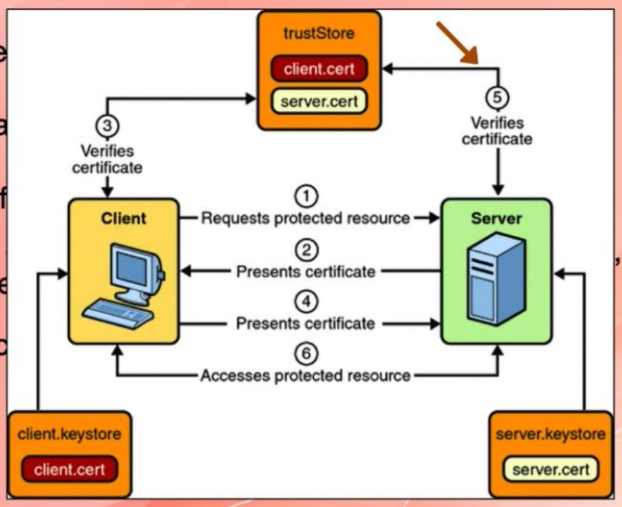
What are Client Certificates?

- Client Certificate
- A Client Certifica
- No encryption of
- When API caller , the server validates the Clie
- This process is c



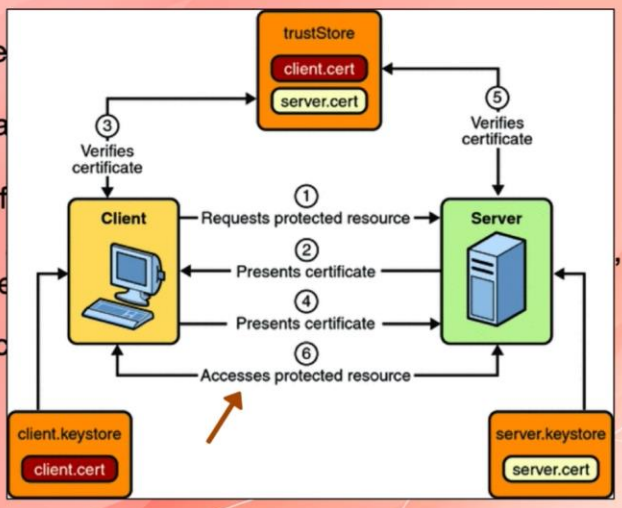
What are Client Certificates?

- Client Certificate
- A Client Certifica
- No encryption of
- When API caller , the server validates the Clie
- This process is c



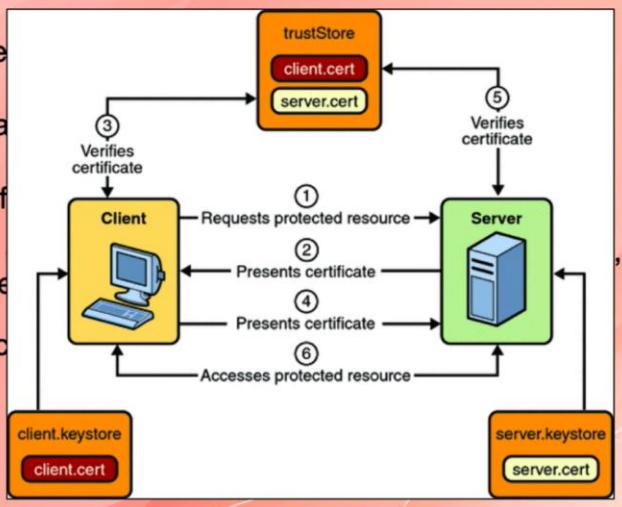
What are Client Certificates?

- Client Certificate
- A Client Certifica
- No encryption of
- When API caller , the server validates the Clie
- This process is c



What are Client Certificates?

- Client Certificate
- A Client Certifica
- No encryption of
- When API caller , the server validates the Clie
- This process is c



What is Semantic Monitoring?

- Monitoring microservices is crucial
- Service layer monitoring

  Is my micro service working as expected?

  Achieved via Health Endpoints (/health), Telemetry (i.e. Prometheus & Grafana) etc

- Semantic monitoring

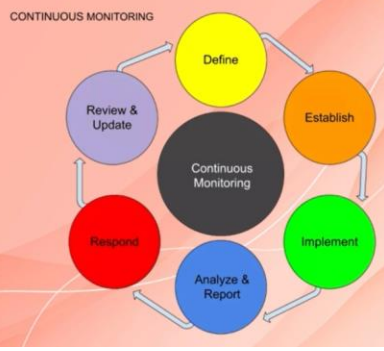
  Approaches micro service monitoring from the business transaction, or semantic, perspective.

  How well the transaction performs for the business and the customers who use it

  Achieved via Functional Testing

What is Continuous Monitoring?

- Continuous Monitoring (CM) is also known as Continuous Control Monitoring (CCM)
- CM is an automated process that allows engineers to detect compliance and security threats in their software development lifecycle and infrastructure.
- Unlike traditional manual & periodical checks, CM helps to identify and track key risks in real time because it uses automation.



Explain OAuth

- OAuth stands for Open Authentication
- Users and Credentials are stored in an authentication server i.e. Google
- Credentials are made of Scope and Claim
    - Scope: What information can be accessed by the user? i.e. Full Name
    - Claim: A set of key-value pairs

Why OAuth suits Micro Services?

- User authenticates against an auth server ONCE
- Using the same Authentication Token all Microservices that use the same Auth Server can be accessed (if they allow)

Explain Idempotence and its usage

- Idempotence is a concept: produce same output for same input
- It is used to make Microservices more resilient
- Some Microservices are triggered by receiving messages from a message broker i.e. ActiveMQ
- Message brokers may send a message more than once
- Microservices must be programmed to handle duplicate messages
- Handiling duplicate messages is called Idempotence
- This is done by
    - Every message must have a unique identifier
    - Microservice stores the message in a local database

What is Service Discovery

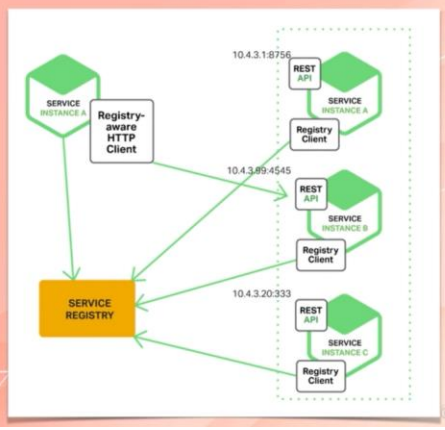
And how is it implemented?

- Service Discovery is used to find where a service is (IP & DNS name)
- This is needed because in the cloud i.e. AWS services have dynamic IPs or DNS name
- Service Instances (i.e. a virtual machine or container) register or write their IP or DNS name in a service registry
- Service Registry is a Key-Value pair storage. Service Name, IP

What is Service Discovery

Client-Side Service Discovery

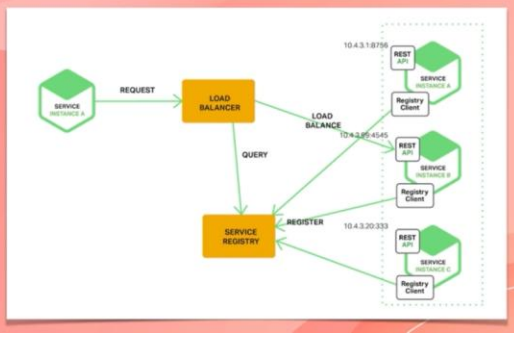
- The client service is responsible for discovering the network location of the services and load balance across them.
- Services register their IP when they start up
- IP of a service is removed using a Heart Beat mechanism
- No Load Balancer will be required

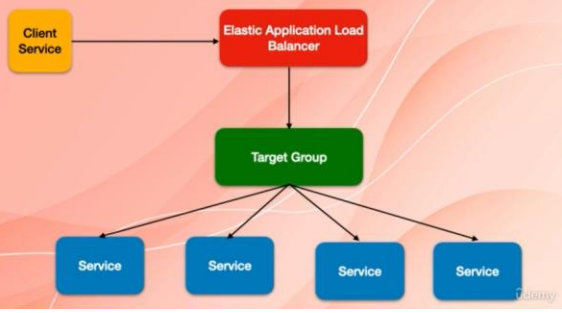


What is Service Discovery

Server-Side Service Discovery

- The client connects to Service Registry via a Load Balancer
- The load balancer queries the Service Registry
- The load balancer routes the traffic to target micro service





What is Service Discovery

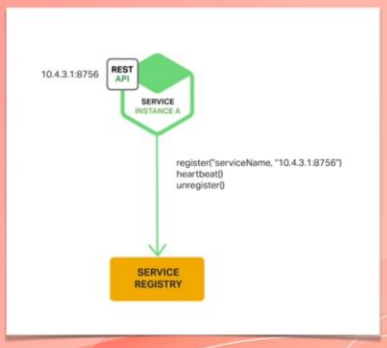
Examples of Service Registry Systems

- etcd: A highly available, distributed service discovery system. Used by Kubernetes
- Hashicorp consul: Offers fast service discovery, load balancing and APIs for registering and de-registering services
- Apache Zookeeper: Used to coordinate distributed systems. It was built for Hadoop. Commonly used alongside Apache Kafka

What is Service Discovery

Service Registration Patterns

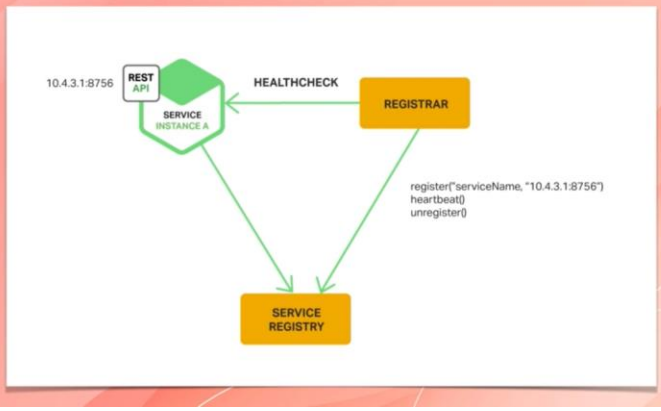
- Self Registration: Services register and de-register themselves



What is Service Discovery

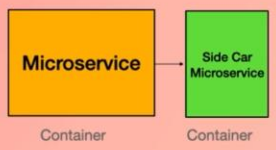
Service Registration Patterns

- Third-Party Registration: Another system or micro service does the registration and de-registration



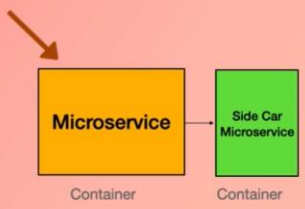
What's Side Car pattern?

- Definition: Deploying components of an application or service into a separate process or container to provide isolation and encapsulation.



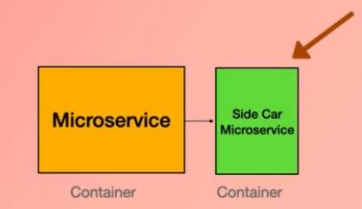
What's Side Car pattern?

- Definition: Deploying components of an application or service into a separate process or container to provide isolation and encapsulation.



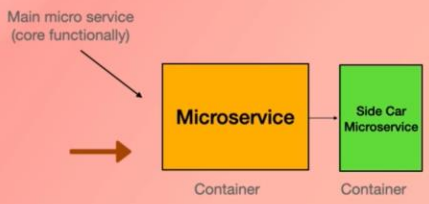
What's Side Car pattern?

- Definition: Deploying components of an application or service into a separate process or container to provide isolation and encapsulation.



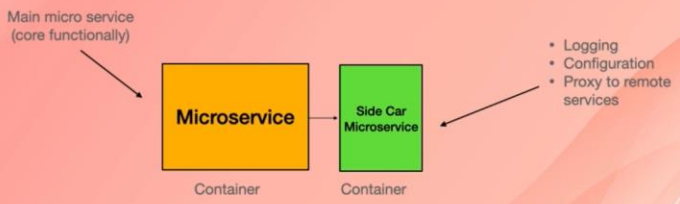
What's Side Car pattern?

- Definition: Deploying components of an application or service into a separate process or container to provide isolation and encapsulation.



What's Side Car pattern?

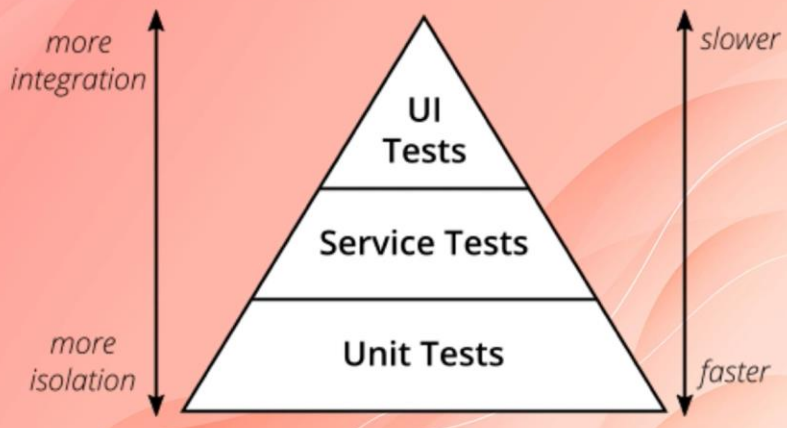
- Definition: Deploying components of an application or service into a separate process or container to provide isolation and encapsulation.



Types of tests in micro services

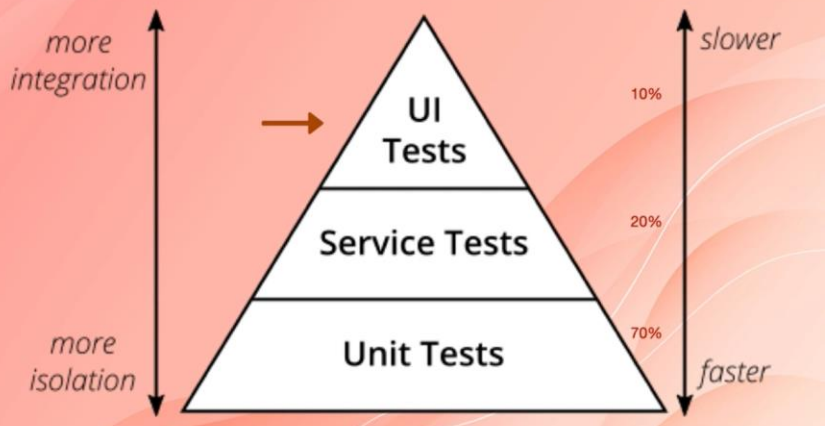
- Functional Testing: Is the overall system working?
- Load Testing: Does the service scale when the load goes up?
- Resilience Testing: How the application reacts to infrastructure failure.

What is Mike Cohn's Test Pyramid?



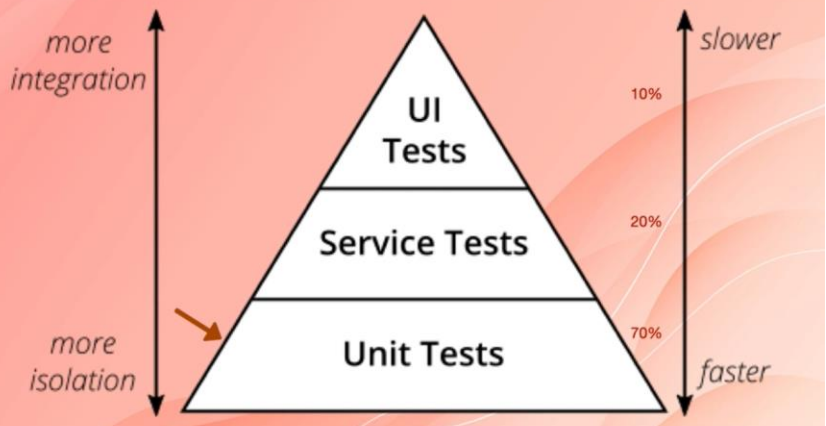
What is Mike Cohn's Test Pyramid?

- Write tests with different granularity.
- The more high-level you get the fewer tests you should have.



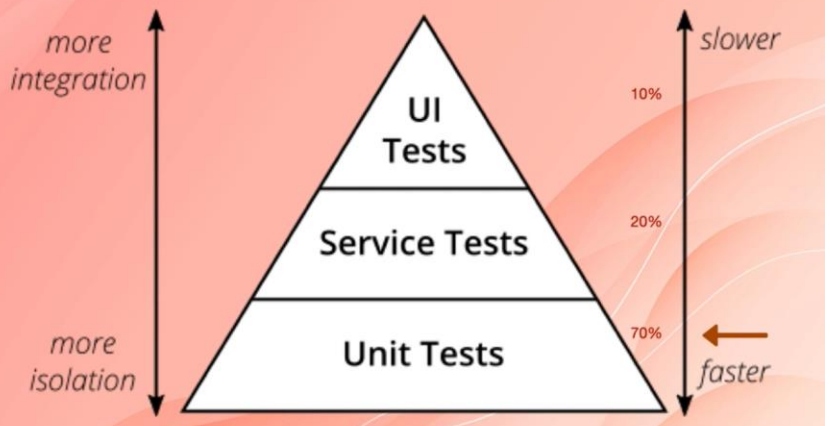
What is Mike Cohn's Test Pyramid?

- Write tests with different granularity.
- The more high-level you get the fewer tests you should have.



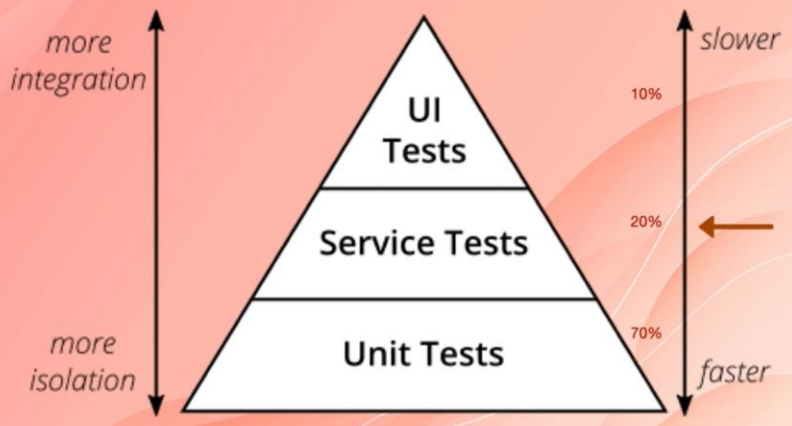
What is Mike Cohn's Test Pyramid?

- Write tests with different granularity.
- The more high-level you get the fewer tests you should have.



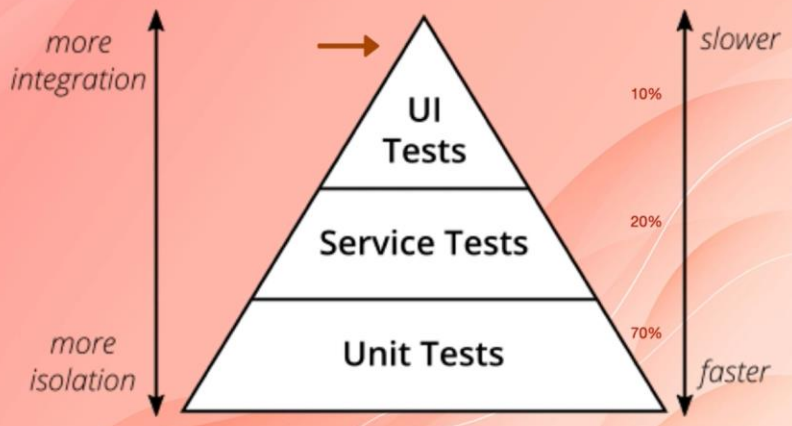
What is Mike Cohn's Test Pyramid?

- Write tests with different granularity.
- The more high-level you get the fewer tests you should have.



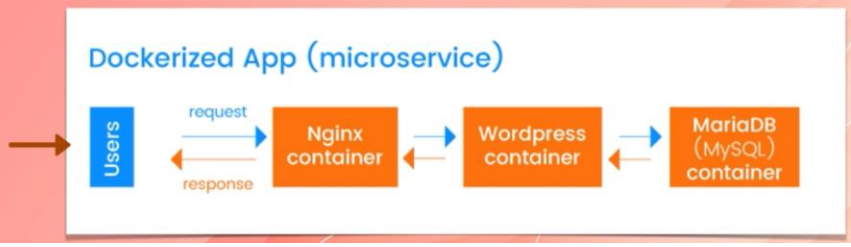
What is Mike Cohn's Test Pyramid?

- Write tests with different granularity.
- The more high-level you get the fewer tests you should have.



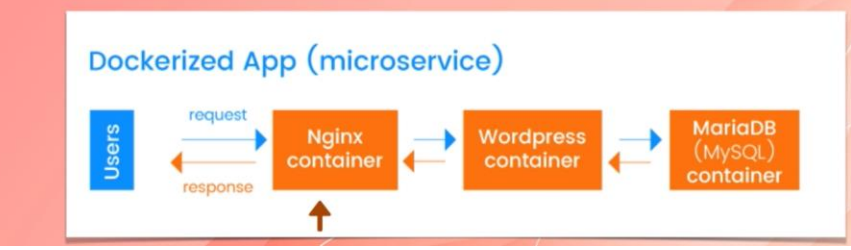
Explain Container in Microservices

- A container is a bundling of an application (i.e. micro service) and all its dependencies as a package, that allows it to be deployed easily and consistently regardless of environment.
- For example a software that only runs on Linux and uses MySQL can be deployed to a Windows server without having to install MySQL.
- Containers use Virtualisation features of the host operating system.
- Containers allow us to deploy Microservices to various environments. Also services can be built with different technologies but run side by side.



Explain Container in Microservices

- A container is a bundling of an application (i.e. micro service) and all its dependencies as a package, that allows it to be deployed easily and consistently regardless of environment.
- For example a software that only runs on Linux and uses MySQL can be deployed to a Windows server without having to install MySQL.
- Containers use Virtualisation features of the host operating system.
- Containers allow us to deploy Microservices to various environments. Also services can be built with different technologies but run side by side.



What is the main role of docker in micro services?

- Create Image: Packages the application and all its dependencies to a binary file called Docker Image.
- Run Containers: A running instance of a Docker Image is a Docker Container
- Provides Networking: Containers can connect to each other i.e. Website container connects to MySQL container

How can you deploy containers?

- Deploy to servers that have Docker on them (not recommended)
- Deploy to server-less container services i.e. AWS Fargate
- Deploy to a container orchestration service i.e. Kubernetes. They provide networking, monitoring, logging and access control.

What is Restful API?

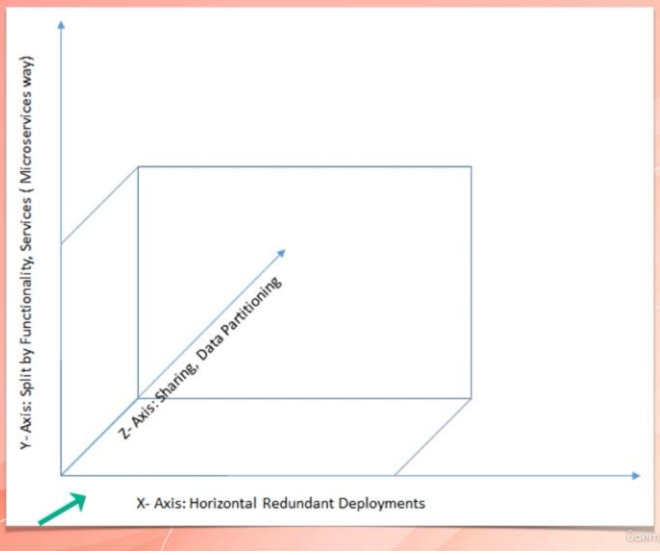
- An API, or *application programming interface*, is a set of rules that define how applications or devices can connect and communicate with each other.
- **Uniform interface**. All API requests for the same resource should look the same, no matter where the request comes from
- **Client-server decoupling**. In REST API design, client and server applications must be completely independent of each other.
- **Statelessness**. REST APIs are stateless, meaning that each request needs to include all the information necessary for processing it
- **Cacheability**. When possible, resources should be cacheable on the client or server side.
- **Layered system architecture**. In REST APIs, the calls and responses go through different layers. Don't assume that the client and server applications connect directly to each other.

What are the ways of testing the security of micro services?

- **DAST**. Dynamic Application Security Testing aka Black Box.
  - Tests the security of the end-to-end application
  - Is done by injecting malicious data or input i.e. SQL Injection
- **SAST**. Static Application Security Testing aka White Box Testing
  - Finds the security issues by scanning and reviewing the source code
- **IAST**. Interactive Application Security Testing
  - Suits more modern applications i.e. Mobile Apps
  - Combines both DAST and SAST approaches
    Places an engine within the application to analyse the app in real time, development and QA
- **RASP**. Run-time Application Security Protocol. It allows the applications to perform continuous security checks on itself and terminate the suspicious sessions.

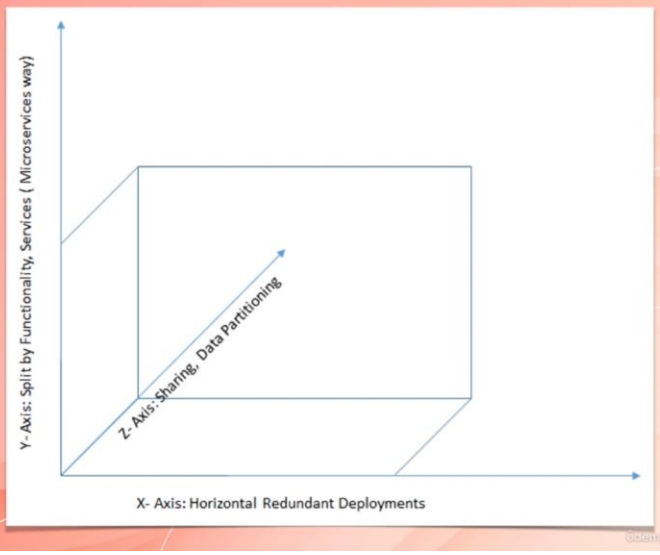
Explain the Scaling Cube

- X-Axis: Add more servers
  - AWS EC2 & Autoscaling Group is an example
- Y-Axis: Split the system by functions: Microservices
- Z-Axis: Partitioning
  - Example: Handle customers of each country by different servers closer to them



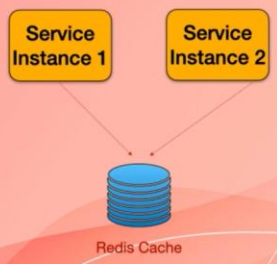
Explain the Scaling Cube

- X-Axis: Add more servers
  - AWS EC2 & Autoscaling Group is an example
- Y-Axis: Split the system by functions: Microservices
- Z-Axis: Partitioning
  - Example: Handle customers of each country by different servers closer to them



What is Data Offloading?

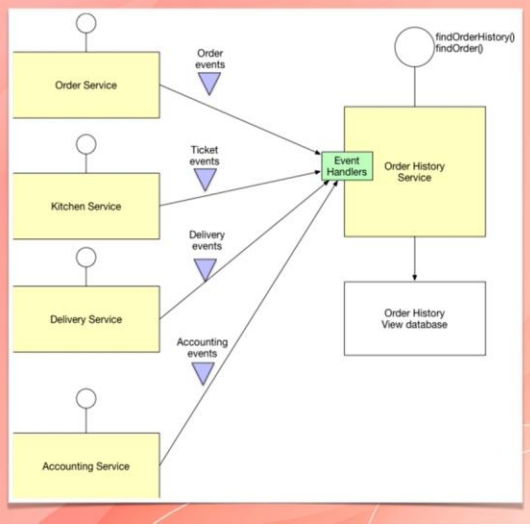
- Microservices normally run in virtual machines i.e. containers
- Microservices may come and go because virtual machines are not stable
- Microservices must not store data locally because if they go the data will go with them
- Microservices must store the state and session data in a shared database such as Redis



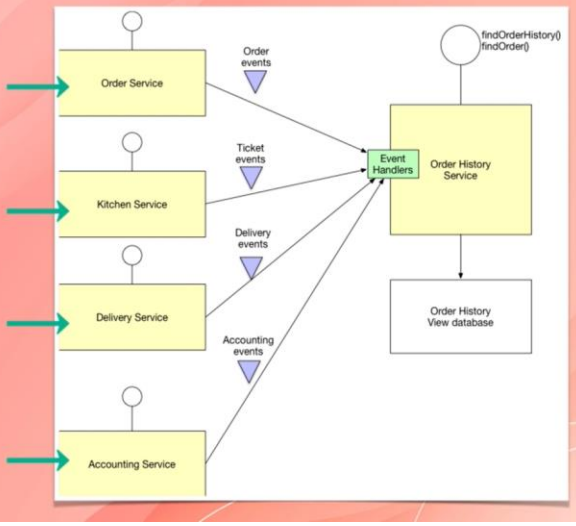
Explain the CQRS pattern

- CQRS stands for Command Query Responsibility Segregation
- Since each micro service has its own database, it is not possible to run SQL Queries across multiple tables in multiple domains
- To support querying a micro service will maintain a view of the data it needs to execute the query
- To keep the view database the micro service subscribes to domain events (Event Streaming)

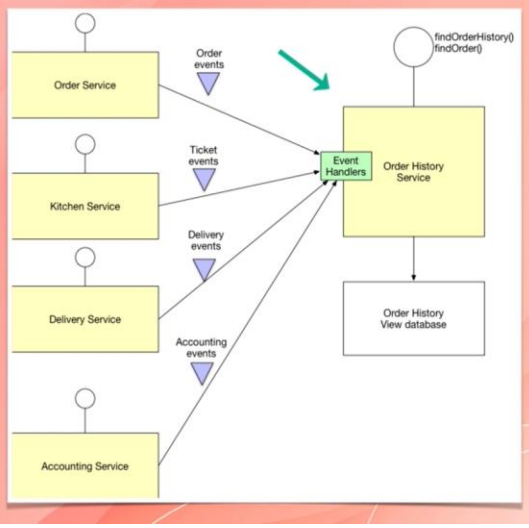
Explain the CQRS pattern



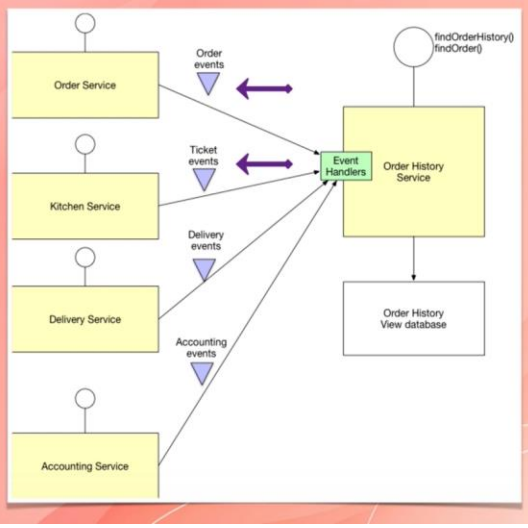
Explain the CQRS pattern



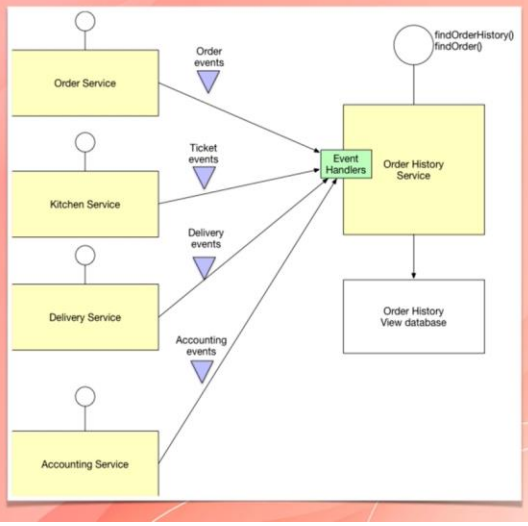
Explain the CQRS pattern



Explain the CQRS pattern

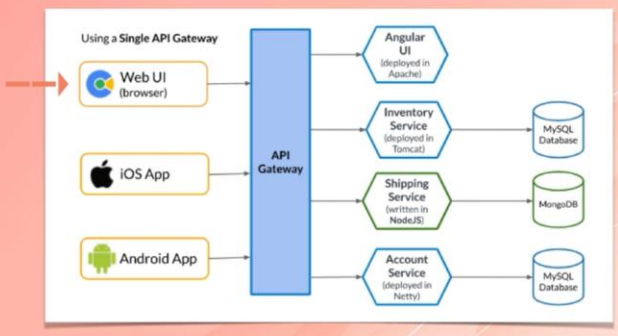


Explain the CQRS pattern



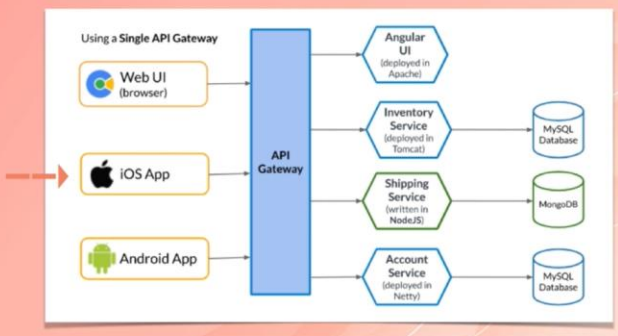
Explain the API Gateway Pattern

- Helps clients i.e. a Mobile App, connect to the micro services



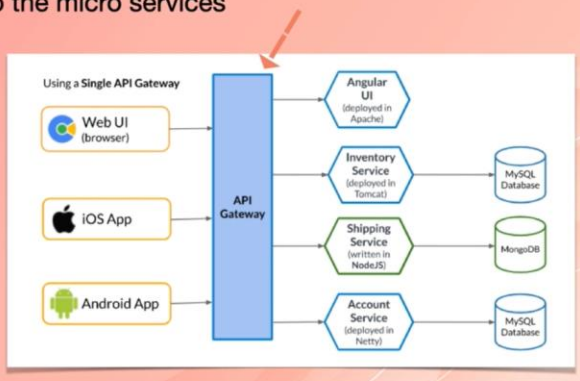
Explain the API Gateway Pattern

- Helps clients i.e. a Mobile App, connect to the micro services



Explain the API Gateway Pattern

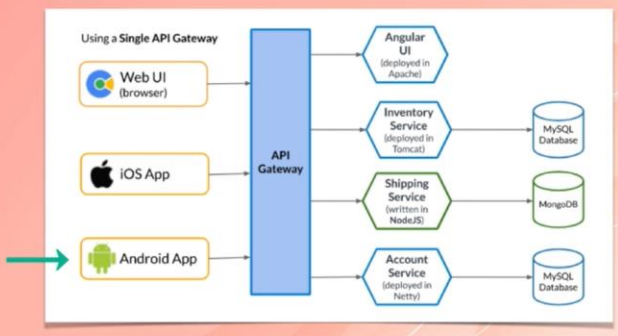
- Helps clients i.e. a Mobile App, connect to the micro services



Explain the API Gateway Pattern

- Helps clients i.e. a Mobile App, connect to the micro services

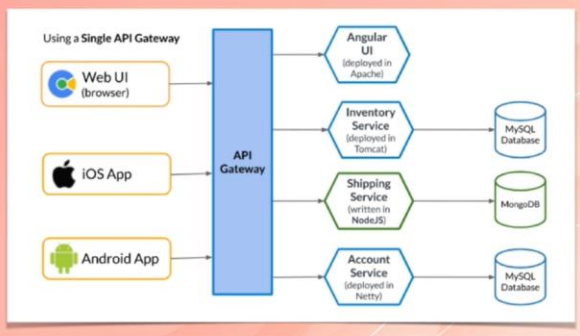
- Without an API Gateway, clients get coupled with the microservice
- API Gateway can aggregate the API calls and reduce network roundtrips



Explain the API Gateway Pattern

- Helps clients i.e. a Mobile App, connect to the micro services

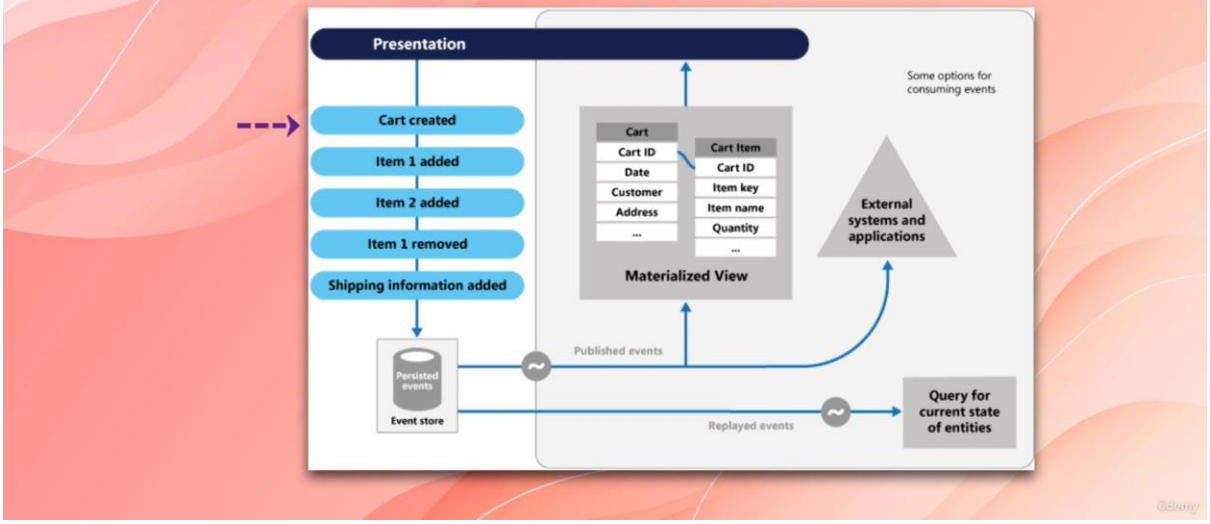
- Without an API Gateway, clients get coupled with the microservice
- API Gateway can aggregate the API calls and reduce network roundtrips
- Security issues if no API Gateway
- API Gateway handles cross-cutting issues i.e. SSL, Auth, Access Log etc



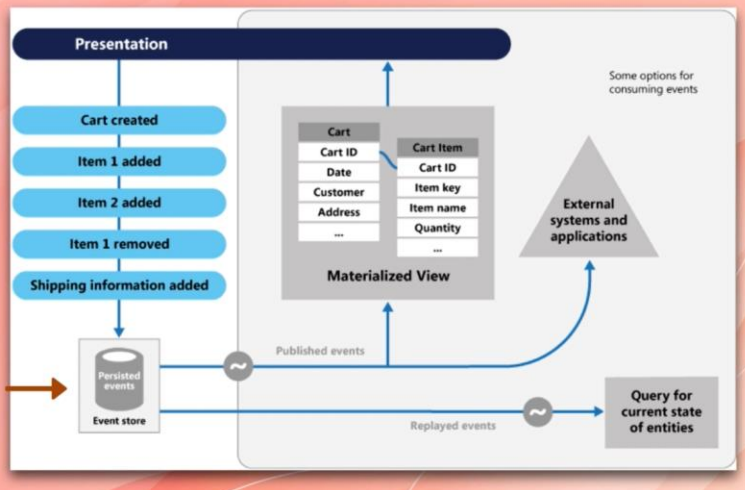
Explain Event Sourcing

- Traditional Create, Read, Update and Delete (CRUD) has issues in micro services architecture
    - CRUD is done on the online database, reducing performance because transactions lock the tables
    - In a micro services architecture multiple services may need to update a model (i.e. an order status) so concurrency is likely
    - It's not possible to know the history of events i.e. Status = 1 , WHY?
- In Event Sourcing:
    - Micro service subscribes to Events that are relevant
    - Stores the events are they are received in an append-only table or database

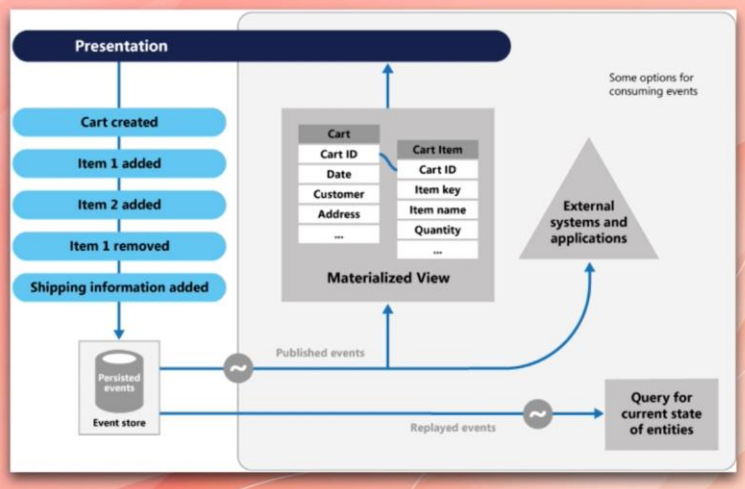
Explain Event Sourcing



Explain Event Sourcing

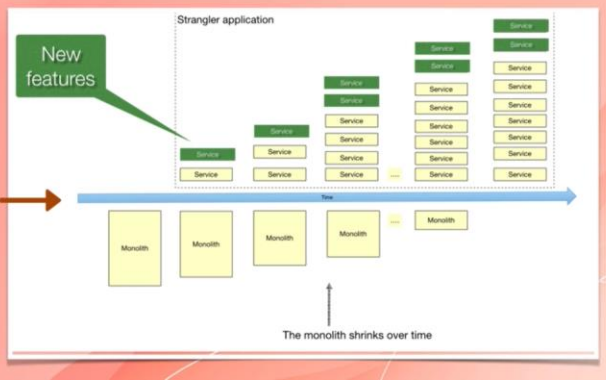


Explain Event Sourcing



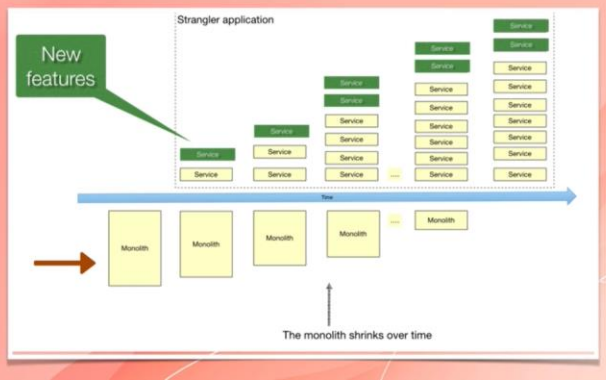
What is Strangler Application

- This is an approach for migrating legacy monolithic applications to Microservices
- Modernize an application by incrementally developing a new (strangler) application around the legacy application. In this scenario, the strangler application has a microservice architecture.



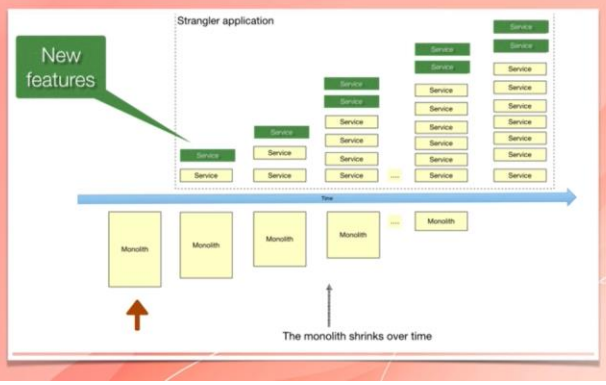
What is Strangler Application

- This is an approach for migrating legacy monolithic applications to Microservices
- Modernize an application by incrementally developing a new (strangler) application around the legacy application. In this scenario, the strangler application has a microservice architecture.



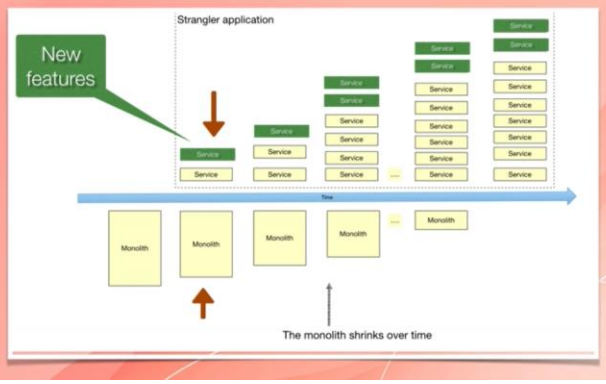
What is Strangler Application

- This is an approach for migrating legacy monolithic applications to Microservices
- Modernize an application by incrementally developing a new (strangler) application around the legacy application. In this scenario, the strangler application has a microservice architecture.



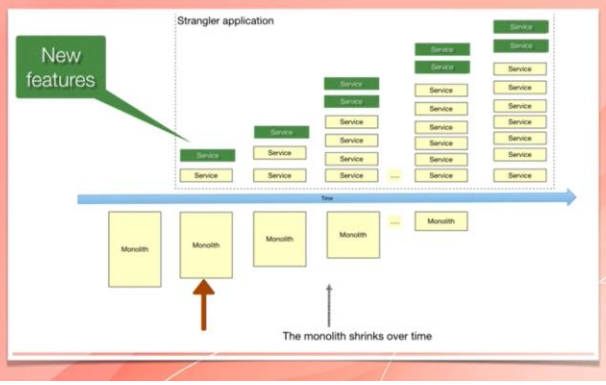
What is Strangler Application

- This is an approach for migrating legacy monolithic applications to Microservices
- Modernize an application by incrementally developing a new (strangler) application around the legacy application. In this scenario, the strangler application has a microservice architecture.



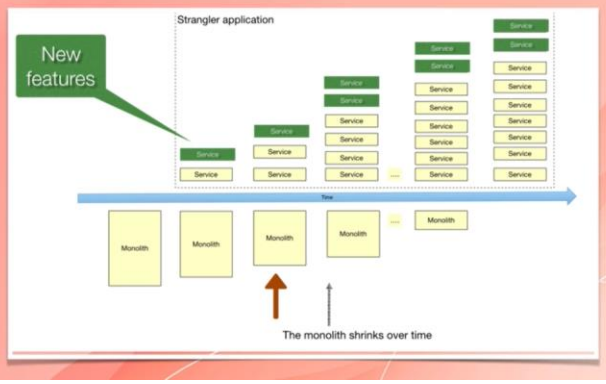
What is Strangler Application

- This is an approach for migrating legacy monolithic applications to Microservices
- Modernize an application by incrementally developing a new (strangler) application around the legacy application. In this scenario, the strangler application has a microservice architecture.



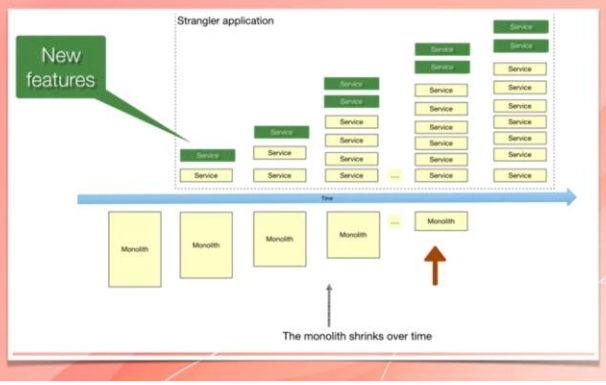
What is Strangler Application

- This is an approach for migrating legacy monolithic applications to Microservices
- Modernize an application by incrementally developing a new (strangler) application around the legacy application. In this scenario, the strangler application has a microservice architecture.



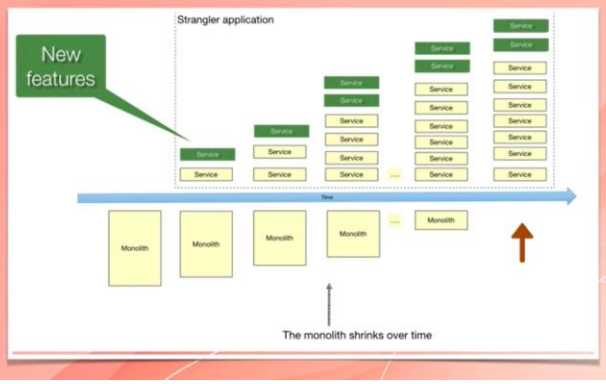
What is Strangler Application

- This is an approach for migrating legacy monolithic applications to Microservices
- Modernize an application by incrementally developing a new (strangler) application around the legacy application. In this scenario, the strangler application has a microservice architecture.



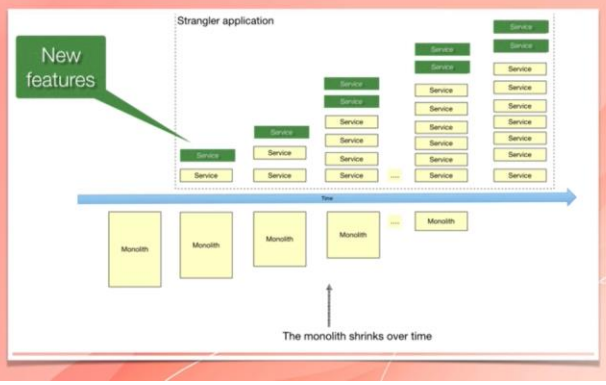
What is Strangler Application

- This is an approach for migrating legacy monolithic applications to Microservices
- Modernize an application by incrementally developing a new (strangler) application around the legacy application. In this scenario, the strangler application has a microservice architecture.



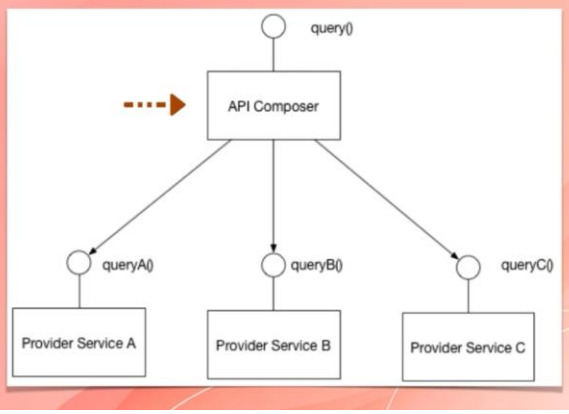
What is Strangler Application

- This is an approach for migrating legacy monolithic applications to Microservices
- Modernize an application by incrementally developing a new (strangler) application around the legacy application. In this scenario, the strangler application has a microservice architecture.



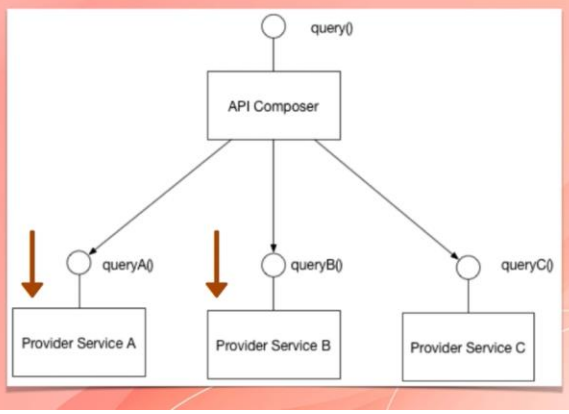
What is API Composition?

- When each microservice has its own database, querying database with joins across two databases or services is not possible
- API Composition queries multiple services and returns the result after joining them



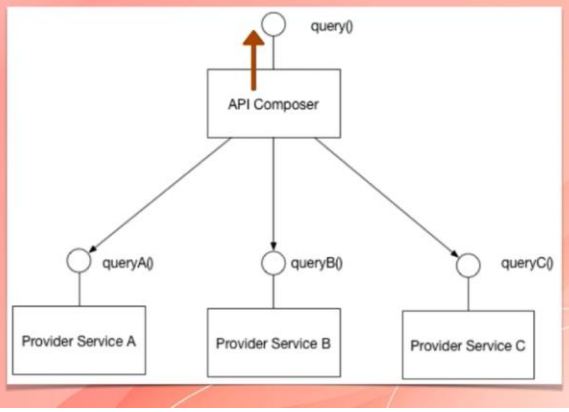
What is API Composition?

- When each microservice has its own database, querying database with joins across two databases or services is not possible
- API Composition queries multiple services and returns the result after joining them



What is API Composition?

- When each microservice has its own database, querying database with joins across two databases or services is not possible
- API Composition queries multiple services and returns the result after joining them



What are patterns of Cross Cutting Concerns?

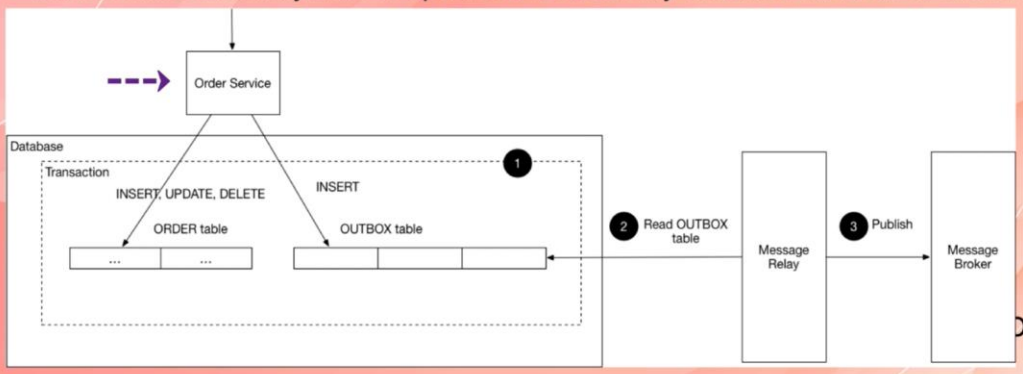
- Microservice Chassis: A framework handles Cross Cutting Concerns (CCC) i.e. Logging
  - Spring boot
  - Spring Cloud
  - DropWizard
  - Gizmo
  - GoKit
- Externalised Configuration: A service must run in multiple environments - dev, test, qa, staging, production - without modification and/or recompilation
    - Store database connection strings, log storage API endpoints etc in an external system in each environment i.e. AWS Config or Hashicorp Consul
- Service Template: Build a microservice template that includes all code for cross-cutting concerns. Use it to create new Microservices.

Transactional Outbox Table

- Microservices normally have to update a record locally and raise an event or send a message
- Sending a message in the middle of a transaction is not reliable
- Microservices add the message to a table while the transaction is in progress
- A Message Relay service reads the committed records from the event or message Outbox table

Transactional Outbox Table

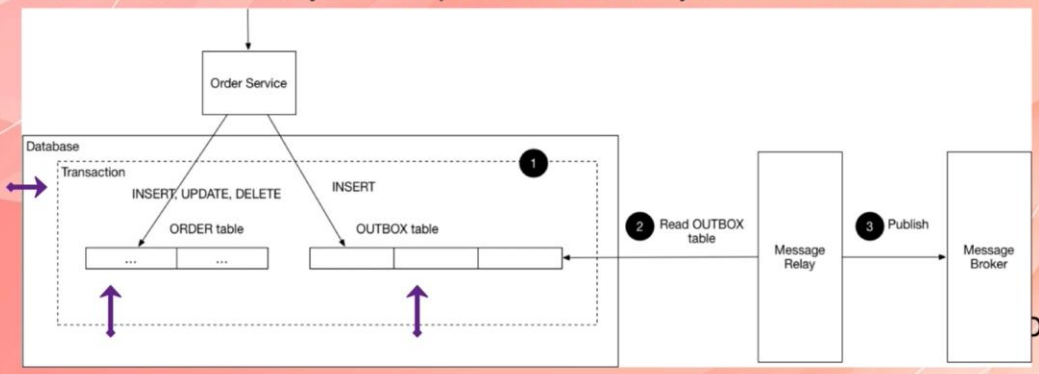
- Microservices normally have to update a record locally and raise an event or send a



Outbox

Transactional Outbox Table

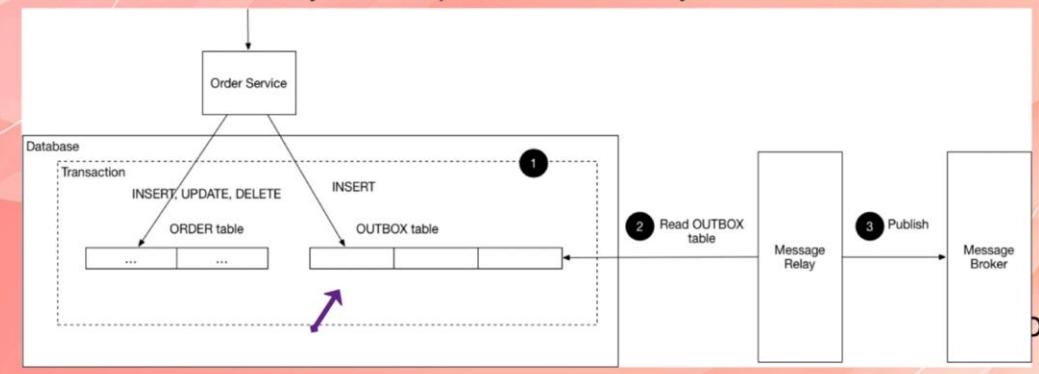
- Microservices normally have to update a record locally and raise an event or send a



Outbox

Transactional Outbox Table

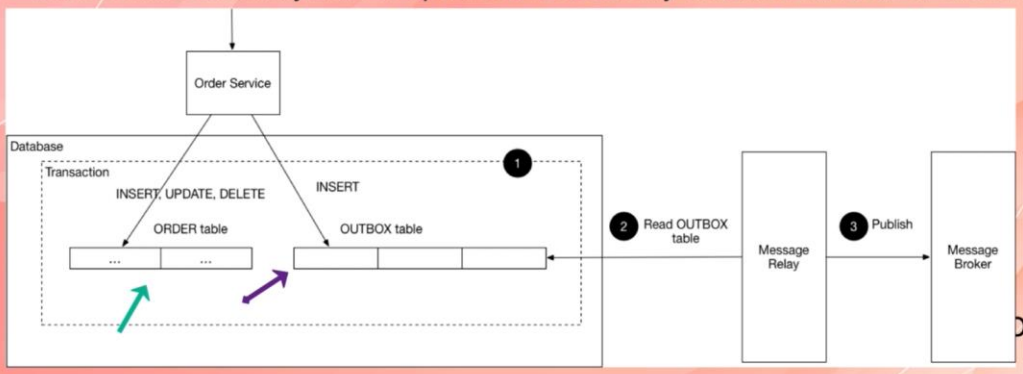
- Microservices normally have to update a record locally and raise an event or send a



Outbox

Transactional Outbox Table

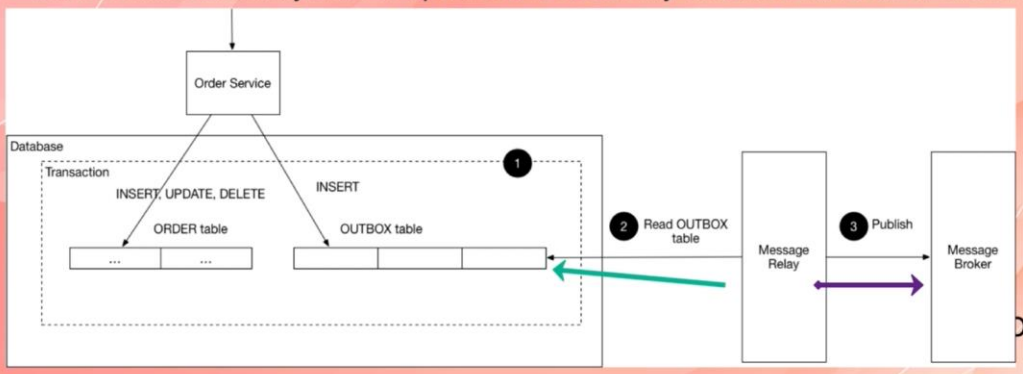
- Microservices normally have to update a record locally and raise an event or send a



Outbox

Transactional Outbox Table

- Microservices normally have to update a record locally and raise an event or send a

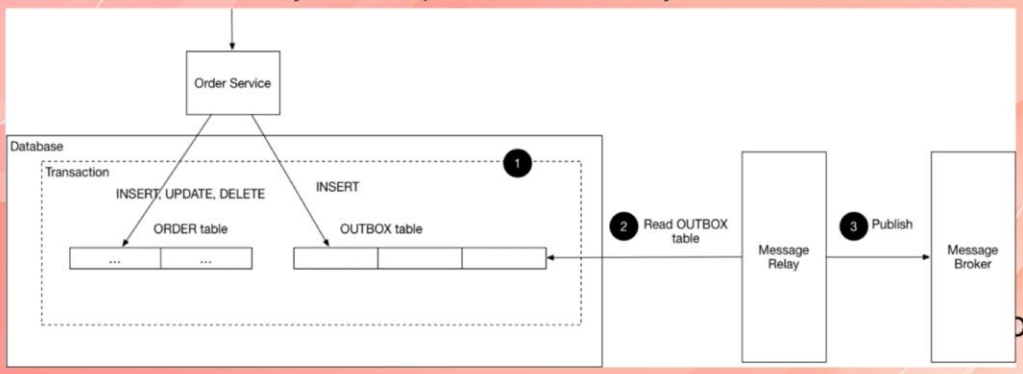


Outbox

47. Explain Transactional Outbox

Transactional Outbox Table

- Microservices normally have to update a record locally and raise an event or send a



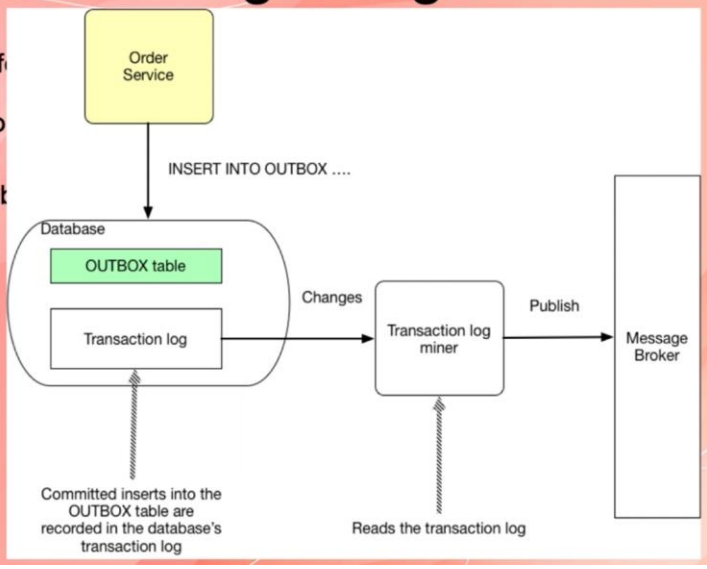
Outbox

Transactional Log Tailing

- It's a pattern for reading events and messages from Outbox Table
- Guaranteed to be accurate
- Requires database-specific solution i.e. specific Sql Server

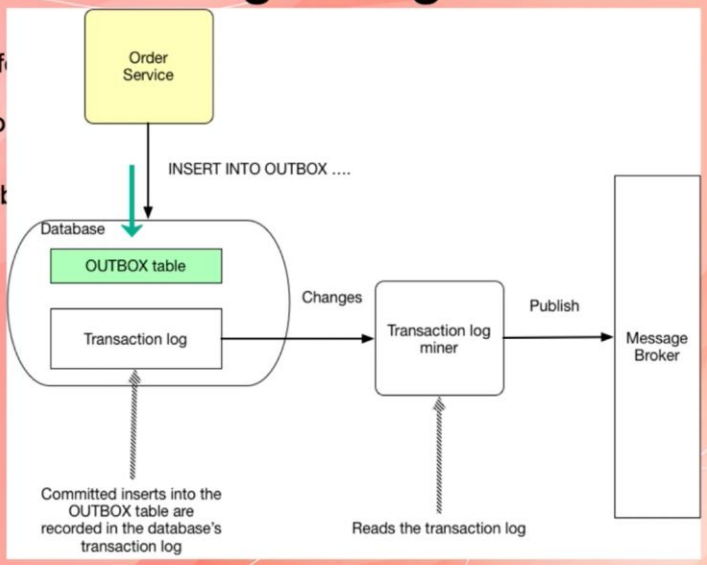
Transactional Log Tailing

- It's a pattern f
- Guaranteed to
- Requires datab



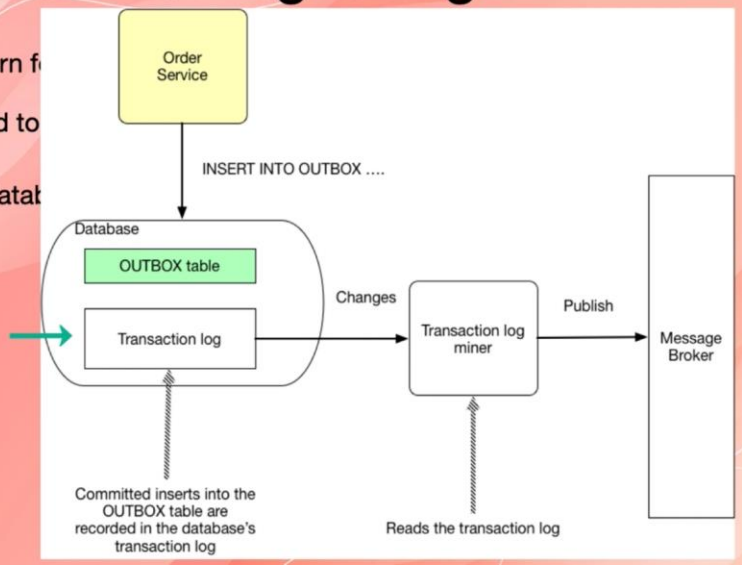
Transactional Log Tailing

- It's a pattern f
- Guaranteed to
- Requires datab



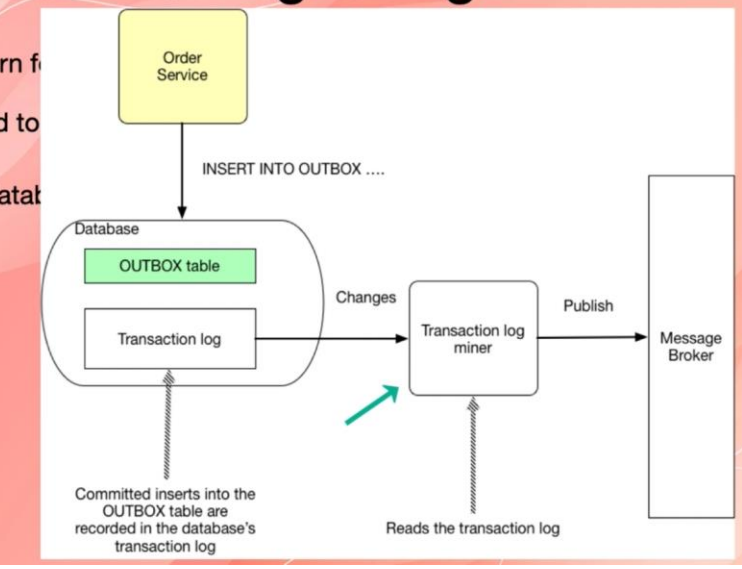
Transactional Log Tailing

- It's a pattern f
- Guaranteed to
- Requires datab



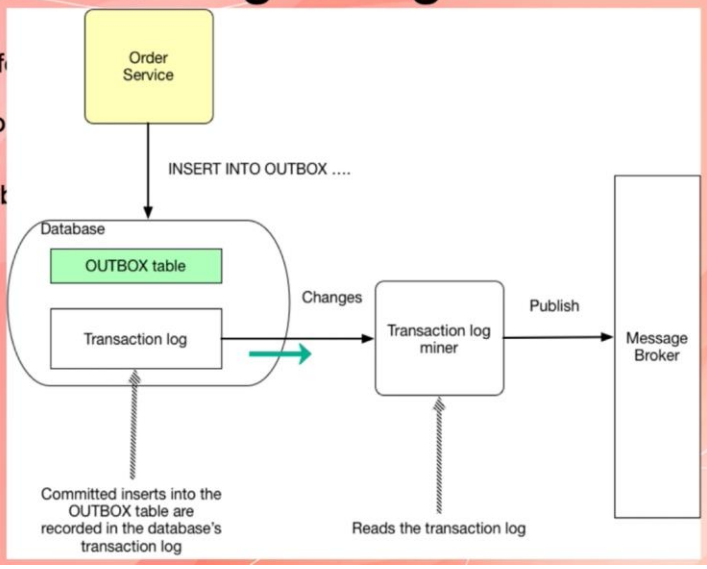
Transactional Log Tailing

- It's a pattern f
- Guaranteed to
- Requires datab



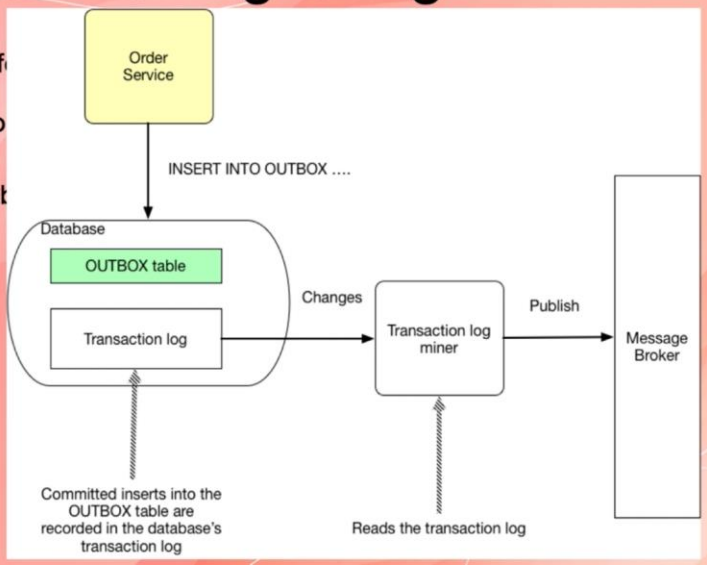
Transactional Log Tailing

- It's a pattern f
- Guaranteed to
- Requires datab



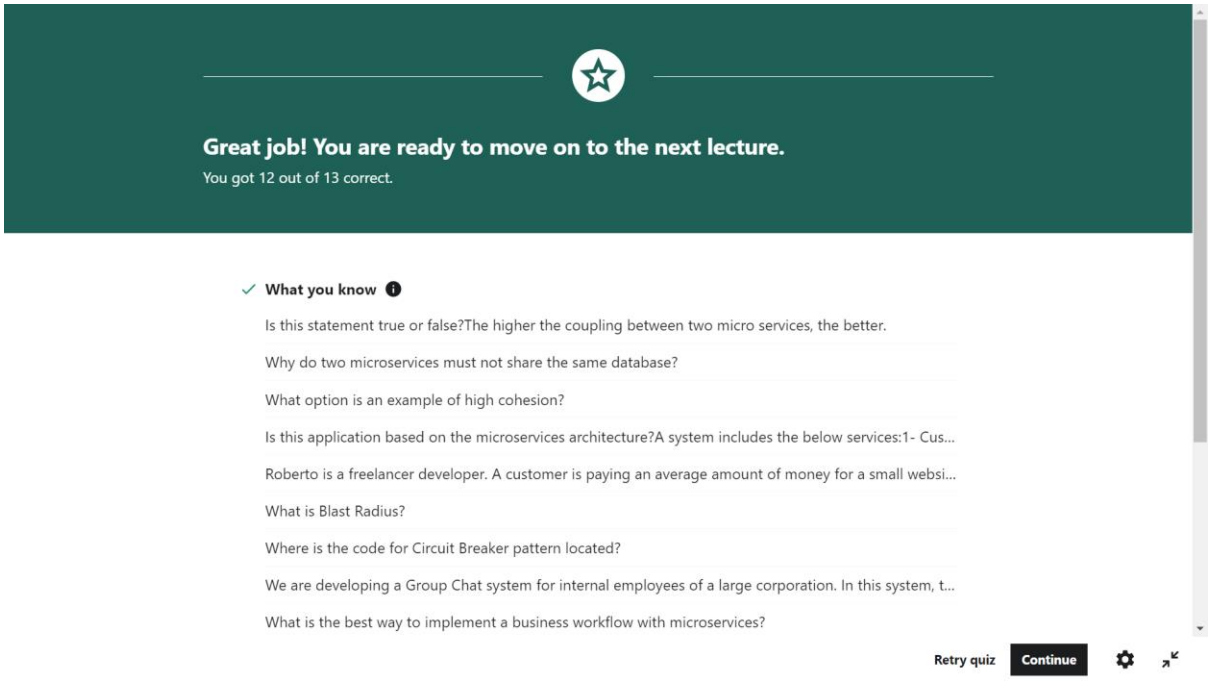
Transactional Log Tailing

- It's a pattern f
- Guaranteed to
- Requires datab



Polling Publisher Pattern

- It's a pattern for reading events and messages from Outbox Table
- Message Relay Services periodically queries the Outbox Table
- If new records found, Message Relay Service sends them to Message Broker, then deletes the messages from the OutBox Table
- Works with any SQL Database



Self Assessment - Microservices

Quiz 1 | 13 questions

The questions are derived from the 50 lectures of the course. Test your knowledge and gain confidence before attending a job interview.

Start quiz **Skip quiz**

Good job!

Question 1:

Is this statement true or false?

The higher the coupling between two micro services, the better.

- ○ True. When coupling goes up the two microservices work together better.
- ◉ **False. We aim to reduce the coupling because the coupling is dependent, and dependency makes the expansion and maintenance of software difficult.**

Good job!
Two microservices might use the same data model in a different way. i.e. Customer in the context of Sales has a different use compared to the domain of Customer Service.

Question 2:

Why do two microservices must not share the same database?

- ○ Because data-level coupling is one of the worst types of couplings, we know that a high degree of coupling is not good.
- ○ Because each of those microservices may do a different thing with data.
- ◉ **Both above options.**

Good job!
Interfaces collect the methods and properties that related to each other in one place. Helper classes may include irrelevant static method hence low cohesion.

Question 3:

What option is an example of high cohesion?

- ○ Global variables in Javascript
- ○ Helper classes in C#
- ◉ **Interfaces in Java and C#**

Good job!
This architecture is SOA. In Micro Services each service has its own database and is totally independent. SOAP is not a common protocol and instead REST is used. Also the services are more granular.

Question 4:

Is this application based on the microservices architecture?

A system includes the below services:

1- Customer

2- Orders

3- Payments

The application has a SQL Server database. The above three services communicate via calling SOAP APIs.

- (●) **No.**
- ( ) Yes.

Good job!
Microservices have a more complex architecture compared to that of monolithic applications. They take longer to develop and test too. If a project is to be done on a low budget and quickly it is better to be done as a monolithic application. The programming language really does not matter. Anything that the developer is comfortable with will work.

This was discussed in Lecture 5: **Benefits and Drawbacks of Monolithic Applications** ›

Question 5:

Roberto is a freelancer developer. A customer is paying an average amount of money for a small website, and he wants the website to be done quickly. What do you recommend to Roberto?

- ( ) Build the website with microservices. Then, the customer can extend it later.
- (●) **Build a monolithic application with ASP.NET Core**

Good job!

Question 6:

What is Blast Radius?

- [x] **How the overall system gets impacted by the failure of a microservice**
- [ ] **It is a kind of test like TDD. We add code until the blast radius becomes zero**

Good job!

Question 7:

Where is the code for Circuit Breaker pattern located?

- [ ] **In the client microservice**
- [ ] **In the microservice that exposes an API (server)**
- [x] **A proxy microservice that sits between the client microservice and the server microservice.**

Good job!
We need to use a message broker as message brokers allow re-ordering messages, and support the concept of Channels which means different users can reveive messgaes in the same Topic but on a different channel. RabbitMQ is a message broker so it is the best option.

Question 8:

We are developing a Group Chat system for internal employees of a large corporation. In this system, the managers have priority over other employees for receiving the company announcements. What is best to be used for building the server-side application? Rabbit MQ or Apache Kafka?

- [x] **RabbitMQ**
- [ ] Apache Kafka

Good job!
As a rule of thumb orchestraton is used inside a bounded context. But between bounded contexts coreography is used. This is a rule of thumb and not always true.

Question 9:

What is the best way to implement a business workflow with microservices?

- [ ] Choreography design pattern
- [ ] Orchestration pattern
- [x] **Orchestration within a bounded context, choreography between two bounded contexts**

Good job!

Question 10:

What is a client certificate?

- ○ It is the Base64 format of username: password. This value is sent to the server upon a Restful API call for authentication.
- ○ It is the Private Key of a Public-Private key pair.
- ◉ **It is the Public Key of a Public-Private key pair.**

Good job!

SQS is likely to show an item in the queue more than once. The application must handle the duplicate messages, which is Idempotence

This was discussed in Lecture 29: **Explain Idempotence and its usage** ›

Question 11:

A system is made of several AWS Lambda microservices. These Lambda microservices receive messages via Amazon Simple Queue Service (SQS). What design pattern must they use?

- ○ Sprout Methods
- ◉ **Idempotence**
- ○ Event Sourcing

Good job!

Question 12:

Which option is best for building microservices that send events from the server to Web clients?

- ○ Redis
- ○ Memcached
- ◉ **Both options**

Good job!
Both patterns are used to query data across microservices

This was discussed in Lecture 47: **Explain Transactional Outbox** ›

Question 13:

Which two design patterns are similar?

- (●) **API Composition and CQRS**
- ( ) **Transactional Outbox and Data Offloading**
- ( ) **SideCar pattern and CDC**

Question 13 of 13

See results ›

Review the course materials to expand your learning.

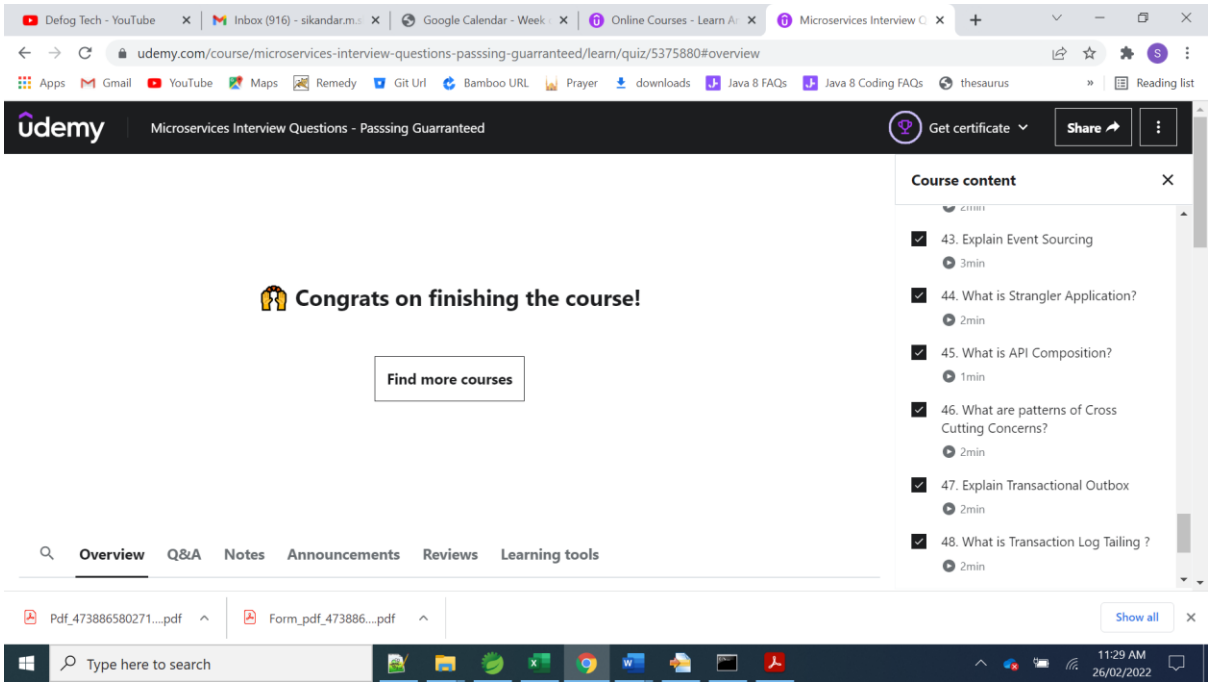
You got 10 out of 13 correct.

✓ What you know

- Is this statement true or false?The higher the coupling between two micro services, the better.
- Why do two microservices must not share the same database?
- What option is an example of high cohesion?
- Is this application based on the microservices architecture?A system includes the below services:1- Cus...
- What is Blast Radius?
- Where is the code for Circuit Breaker pattern located?
- We are developing a Group Chat system for internal employees of a large corporation. In this system, t...
- What is the best way to implement a business workflow with microservices?
- What is a client certificate?
- Which option is best for building microservices that send events from the server to Web clients?

✗ What you should review

Continue **Retry quiz**


Defog Tech - YouTube
Inbox (916) - sikandar.m
Google Calendar - Week
Online Courses - Learn A
Microservices Interview C
udemy.com/course/microservices-interview-questions-passsing-guarranteed/learn/quiz/5375880#overview
Apps
Gmail
YouTube
Maps
Remedy
Git Url
Bamboo URL
Prayer
downloads
Java 8 FAQs
Java 8 Coding FAQs
thesaurus
Reading list
udemy
Microservices Interview Questions - Passsing Guarranteed
Get certificate
Share
Course content
43. Explain Event Sourcing
3min
44. What is Strangler Application?
2min
45. What is API Composition?
1min
46. What are patterns of Cross Cutting Concerns?
2min
47. Explain Transactional Outbox
2min
48. What is Transaction Log Tailing ?
2min
Congrats on finishing the course!
Find more courses
Overview
Q&A
Notes
Announcements
Reviews
Learning tools
Pdf_473886580271....pdf
Form_pdf_473886....pdf
Show all
Type here to search
11:29 AM
26/02/2022