Ways to communicate between Microservices

- We have seen Synchronous communications through -
  - Rest APIs
  - GraphQl
  - Feign using Eureka discoveries
  - GRPC (10 times faster than REST APIs) - developed by Google as substitute of REST with many more features.

- A synchronous call means that a service waits for the response after performing a request.

- Today we will look at ways to do asynchronous communication in java. This communication usually involves some kind of messaging system like
  - Active Mqs
  - Rabbit MQs
  - Kafka

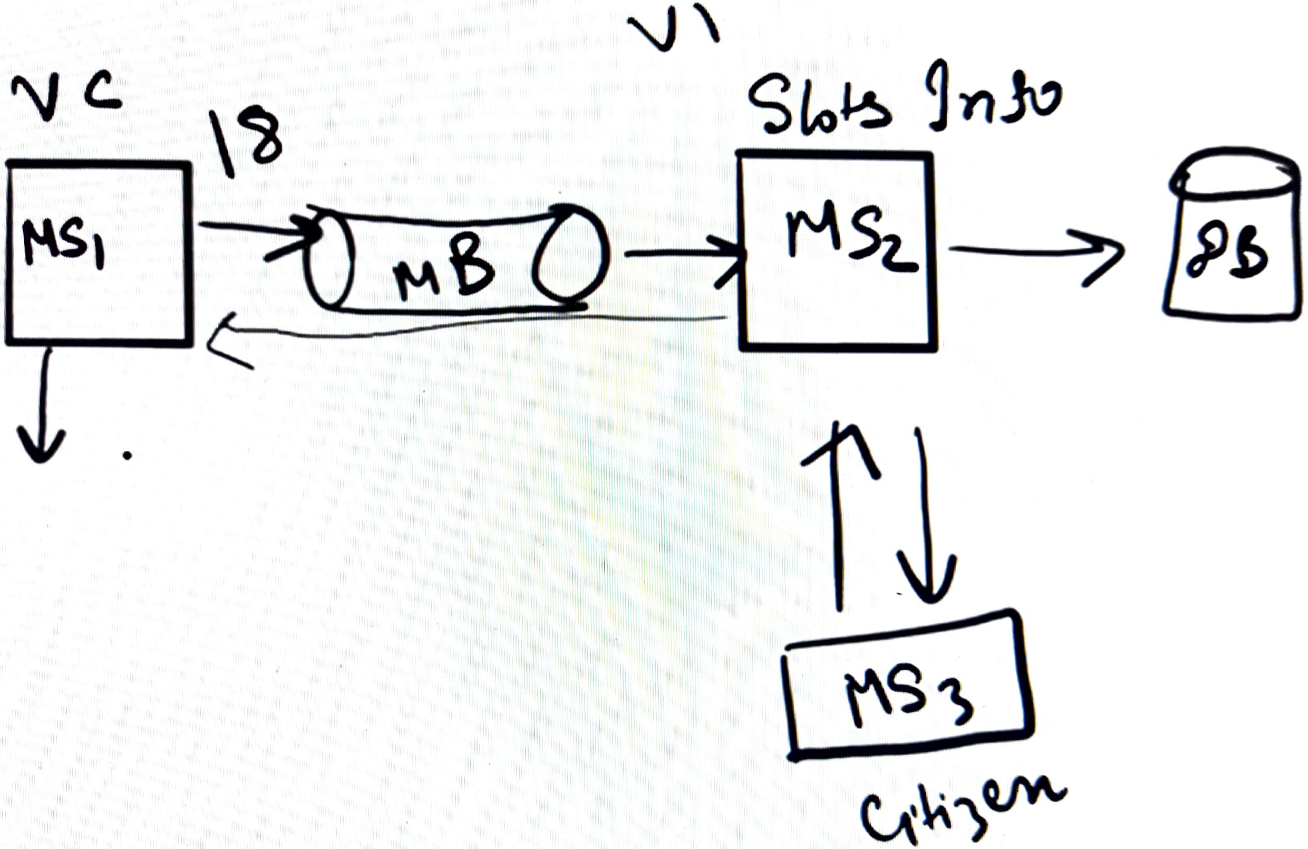
What is Async communication

- In Async communication , To initiate such type of communication, a Microservice who wants to send some data to another Microservice publishes a message to a separate component known as a message broker. It is responsible for handling the message sent by the producer service and it will guarantee message delivery.

- After the message is received by the broker, it's now its job to pass the message to the target service. If the recipient is down at the moment, the broker might be configured to retry as long as necessary for successful delivery.

- These messages can be persisted if required or stored only in memory. In the latter case, they will be lost when the broker is restarted and they are not yet sent to the consumer.

- Since the broker is responsible for delivering the message, it's no longer necessary for both services to be up for successful communication. Thus async messaging mitigates the biggest problem of synchronous communication - coupling.


V1
VC
18
Slots Into
MS1
MB
MS2
DB
MS3
Citizen

What is Async communication

- A relevant point here is that there, the sender doesn't need to wait for the response. It might be sent back from the receiver later as another asynchronous message.
- The intended service receives the message in its own time. The sending service is not locked to the broker. It simply fires and forgets.

Types of Async communication

- Commonly, there are two options in message-based communication:
  - Point to Point
  - Publisher-Subscriber

What is Publisher-Subscriber Async communication

- In publisher-subscriber messaging-based communication, the topic in the message broker will be used to store the message sent by the publisher and then subscribers that subscribe to that topic will consume that message

- Unlike point to point pattern, the message will be ready to consume for all subscribers and the topic can have one or more subscribers. The message remains persistent in a topic until we delete it.

- In messaging-based communication, the services that consume messages, either from queue or topic, must know the common message structure that is produced or published by producer or publisher.

- examples are Kafka, Amazon SNS etc

What is Event Based Async communication

- Unlike messaging-based communication, in event-based communication, especially in event-driven pattern , the services that consume the message do not need to know the details of the message.

- In event-driven pattern, the services just push the event to the topic in the message broker and then the services that subscribe to that topic will react for each occurrence event in that topic Each event in the topic will be related to a specific business logic execution.

What did u use in your project and why? Pros and cons of each communication method

- Both have potential pros and cons, but the method you choose depends on an application's purpose.

- Pros of Synchronous Communication
  - Synchronous communication is simpler in design

- Cons -
  - carries the risk of spreading failures across services

- Way to mitigate failures in Synchronous communication -
  - The architect must implement sophisticated service discovery and application load balancing among microservices.

What are Microservices?

- Microservices is an architecture where the application is exposed as loosely coupled services that can be independently developed, deployed, and maintained. Each service exposed is referred to as Microservice. Each service performs a unique function.

- Speciality of this architecture is that polyglot architecture is supported. For example, if a team is working on one of the microservice using Java, Spring Boot, and MySQL, another team can work on another microservice using Python, Node JS, and NoSQL.

- Different microservices can use a different version of the same programming language.
- Different microservices can use different programming languages.
- Different microservices can use different architectures as well.

Why Microservices?

In the case of monolith applications, there are several problems like

- Same code base for presentation, business layer, and data access layer. Application is deployed as a single unit.
- Complex to maintain and scalability is an issue.

Microservice solves the above problems.
Microservices are ideal when a monolith or a legacy application needs to be modernized.

For new software development, if the key business drivers are to reduce time to market, scalable better software, lower costs, faster development, or cloud-native development, microservices are ideal.

Each service is independent and gives the flexibility to choose the programming language, database, and/or architecture.

Distinct services can be developed, deployed, and maintained independently.

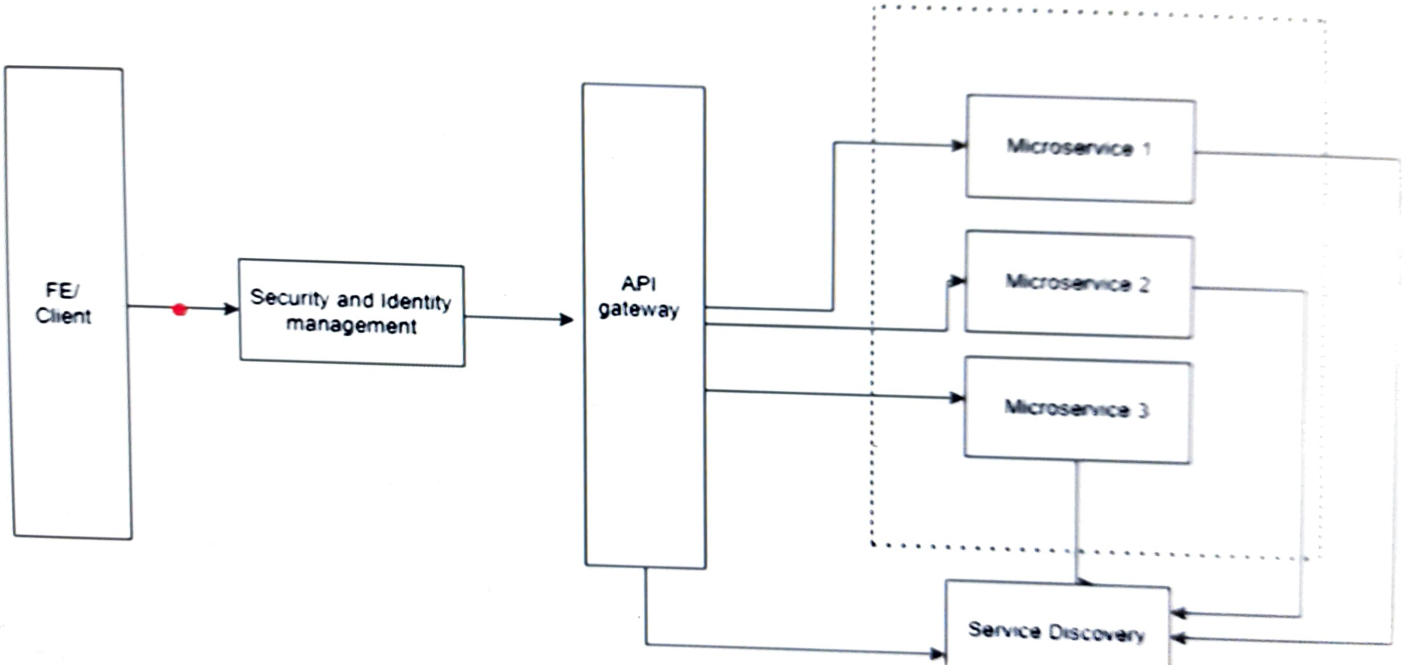
What are the pros and cons of Microservice Architecture?

Pros of Microservice Architecture	Cons of Microservice Architecture
Freedom to use different technologies	Management of a large number of services is difficult.
Each microservices focuses on single capability	Communication between microservices is complex.
Supports individual deployable units	Increased efforts for configuration and other operations
Allow frequent software releases	Difficult to maintain transaction safety and data boundaries
Ensures security of each service	Due to the decentralized nature of microservices, more microservices will mean more resources hence high Investment
Multiple services are parallelly developed and deployed	Debugging of problems is harder unless the right instrumentation is followed during design and development.
	Microservices will need a large team size with the right mix of experience in design, development, automation, deployments, tools, and testing.

When to use microservices?

- Reduce time to market,
- Scalable better software,
- Lower costs,
- Faster development,
- Cloud-native development
- It makes sense to adopt a microservices architecture, if the team size is big enough as each service will require its team to develop, deploy and manage.
- Timeframe and skills of team members are a constraint.
- If fast results are required,
- choose microservices architecture only if the team also has experience in microservices.
- Do not use this architecture for simple application which can be managed by monolithic application .
- So you use ask yourself first do we really need this microservice architecture to decouple the services as it comes with a cost

How does Microservice Architecture work?



What are the main features of Microservices?

- Microservices architecture breaks an application into smaller services, and it is possible to develop, deploy each service independently. This makes the introduction of new features in an application very easier.

- **Decentralization**: Microservices architecture leads to distributed systems. The data management is decentralized. There will be a monolithic database containing all data belonging to the application. Each service has the ownership of the data related to the business functionality of that service.

- **Black box:** Every microservice is defined as a black box. The details of the complexity are hidden from other services/components.

- **Security**: The Microservice platform itself should provide capabilities for certificate management, different types of credentials, authentication, and authentication based on RBAC (Role-based access model). Security is decoupled from the microservice development team as platform standardization help with it.

What are the main features of Microservices?

- **Polyglot**:

This is one of the significant aspects of microservices architecture.

For example, if a team is working on one of the microservice using Java, Spring Boot, and MySQL, another team can work on another microservice using Python, Node JS, and NoSQL.

Different microservices can use a different version of the same programming language. Eg. Python 2.7 and Python 3.0

Different microservices can use different programming languages.

Different microservices can use different architectures as well.

How do microservices communicate with each other?

In the case of Microservice Architecture, there are 2 different types of inter-service communication between microservices.

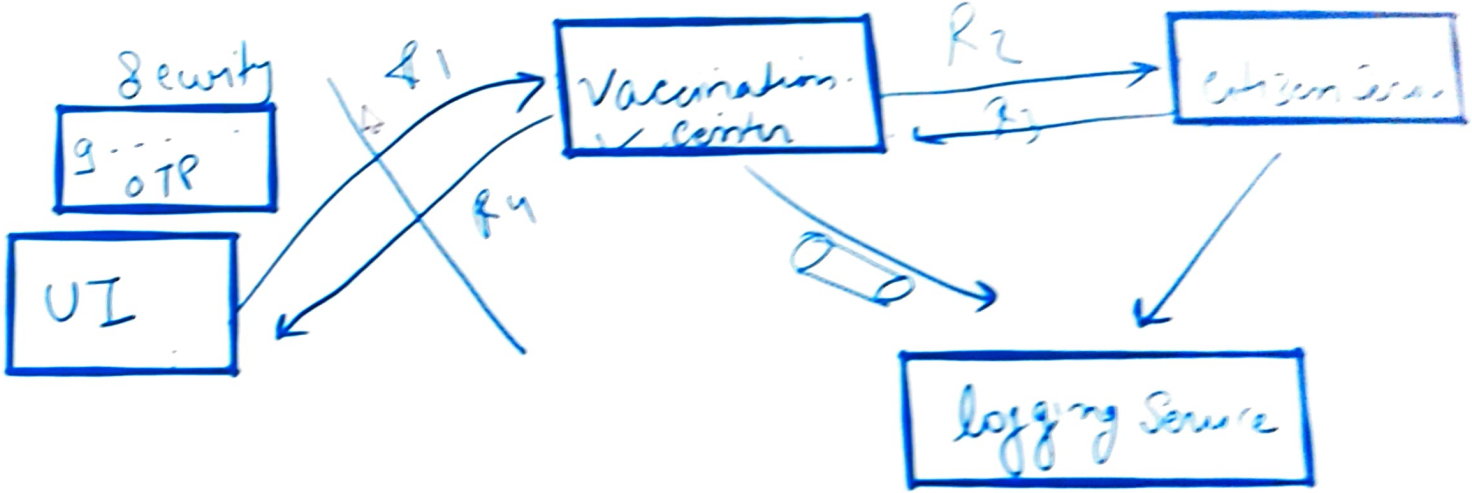
a. Synchronous communication

b. Asynchronous communication

Synchronous communication:

In the case of Synchronous communication between microservices, the client service waits for the response within a time limit. The possible solution is using HTTP Protocol using via REST API for interservice communication.

JACN
Add



What is the difference between Monolithic, SOA and Microservices Architecture?

- Monolithic Architecture is similar to a big container wherein all the software components of an application are assembled together and tightly packaged.
- A Service-Oriented Architecture is a collection of services which communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity.
- Microservice Architecture is an architectural style that structures an application as a collection of small autonomous services, modeled around a business domain.
- Main diff b/w SOA and MS architecture is Based on sharing of data and info. SOA shares and reuses as much as possible while MS focuses on sharing as little as possible

How to design microservices?

Microservices need to be designed by making use of the best practices like

- A separate data store for each of the microservice.
- The application needs to be split into loosely coupled microservices based on business functionality.
- Decentralized framework.
- Polyglot architecture as per the business needs.
- Services to be designed, developed, deployed, and managed separately.
- Domain-driven design
- Real-time monitoring of the application should be possible.
- Deploy in containers