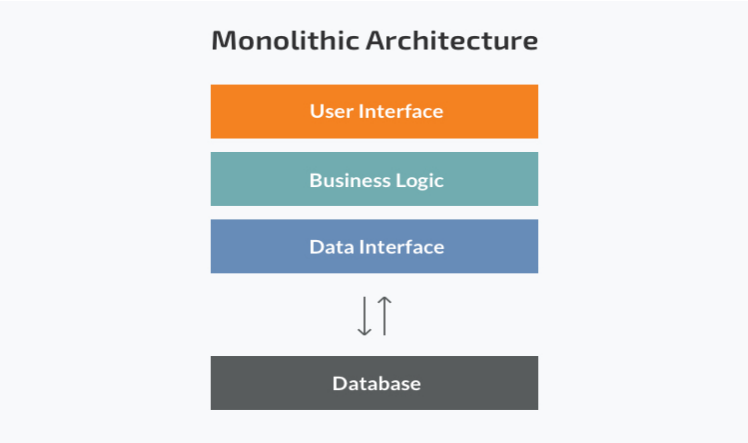
MONOLITHIC ARCHITECTURE:

The monolithic architecture is considered to be a traditional way of building applications.

A monolithic application is built as a single, unified and all the functions such as **client-side user interface, a server side-application, and a database** are managed and served in one place.

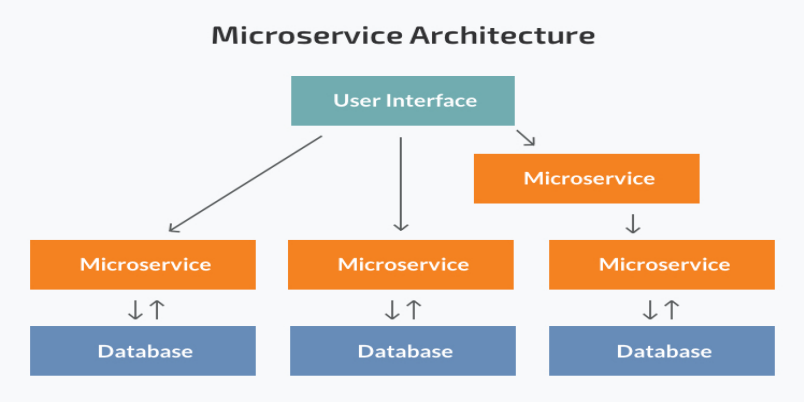
If developers want to update or change something, they access the same code base. So, they make changes in the whole stack at once.



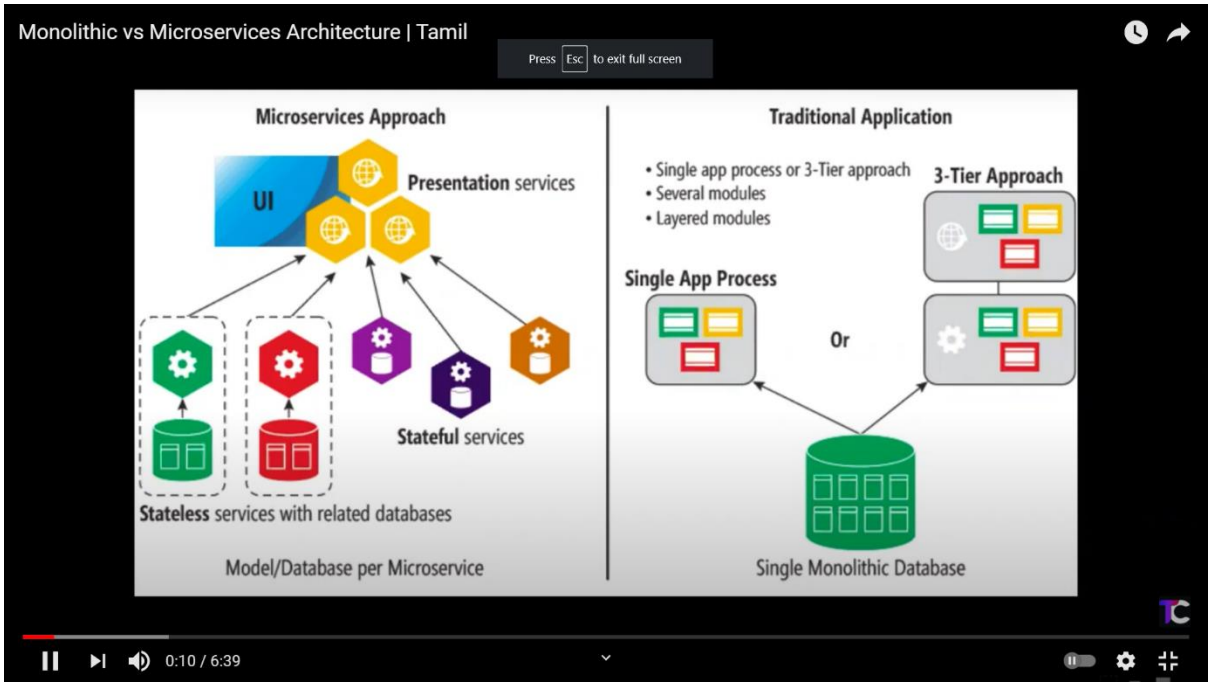
MICROSERVICES ARCHITECTURE:

While a monolithic application is a single unified unit, a microservices architecture breaks it down into a collection of **smaller independent units**. These units carry out every application process as a **separate service**.

So all the services have their own logic and the database as well as perform the specific functions.



Difference Between both Architecture:



Advantage of Monolithic architecture:

Easier debugging and testing -> You can run end-to-end testing much faster.

Simple to Deploy -> You do not have to handle many deployments , just one file or directory is enough.

Simple to Develop -> Any engineering team has the right knowledge and capabilities to develop a monolithic application.

Disadvantage of the Monolithic Architecture

Understanding. -> It becomes too complicated to understand the third person**.**

Making changes. - > Any code change may affects the whole system so it has to be thoroughly coordinated. This makes the overall development process much longer.

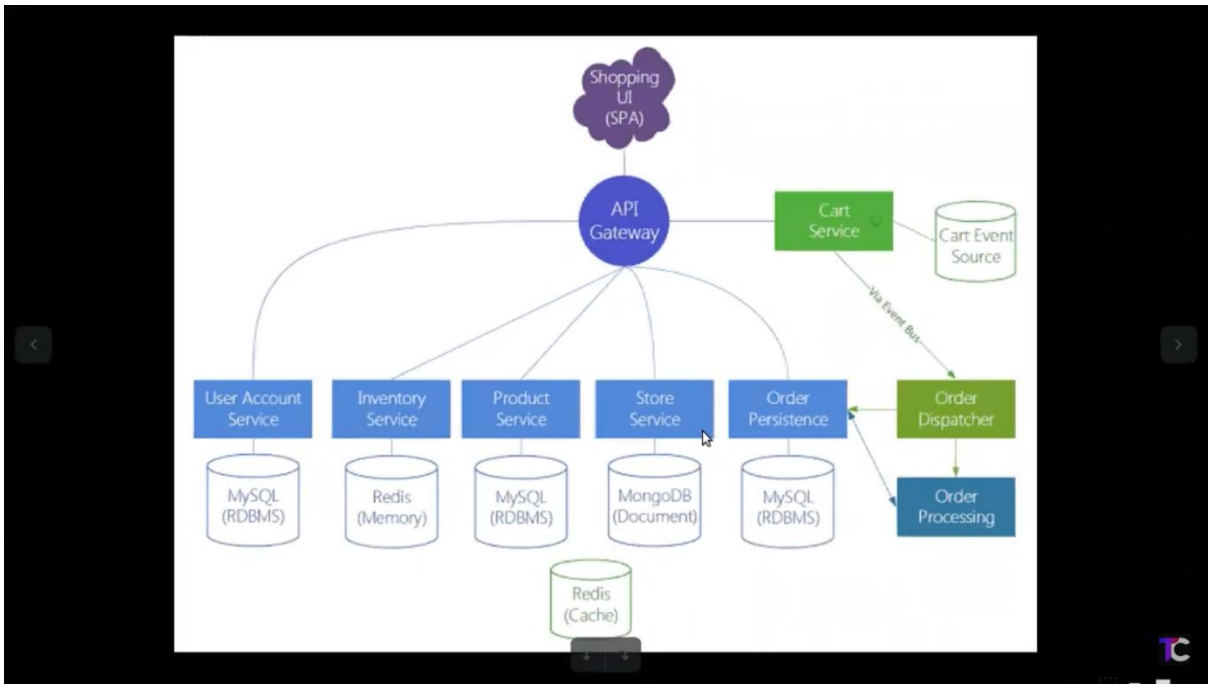
Advantages of Microservice Architecture:

Independent components -> All the services can be deployed and updated independently, which gives more flexibility.

Bug -> Bug in one microservice has an impact only on a particular service and does not influence the entire application.

It is much easier to add new features to a microservice application than a monolithic one.

Easier understanding. -> This application is easier to understand and manage. You just concentrate on a specific service that is related to a business goal you have.



Disadvantages of Microservice Architecture

Extra complexity -> Since a microservices architecture is a distributed system, you have to choose and set up the connections between all the modules and databases. Also, as long as such an application includes independent services, all of them have to be deployed independently.

Testing -> A multitude of independently deployable components makes testing a microservices-based solution much harder.

Message Queue:

A message queue is a form of **asynchronous service-to-service** communication used in serverless and microservices architectures. Messages are stored on the queue until they are processed and deleted. Each message is processed only once, by a single consumer.

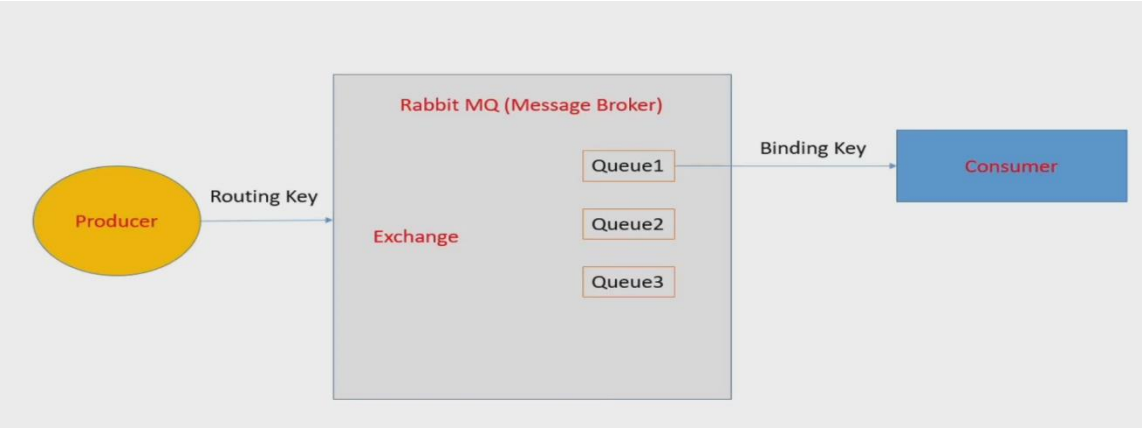
The messages are usually small, and can be things like requests, replies, error messages, or just plain information. To send a message, a component called a producer adds a message to the queue. The message is stored on the queue until another component called a consumer retrieves the message and does something with it.

Message Broker:

A message broker is an intermediary computer program module that translates a message from the formal messaging protocol of the sender to the formal messaging protocol of the receiver. RabbitMQ is one of the most popular open source message brokers.

RabbitMq:

RabbitMQ is a messaging broker and it is an intermediary for messaging. It gives your applications a common platform to send and receive messages, and your messages a safe place to live until received.



Top Alternatives to RabbitMQ

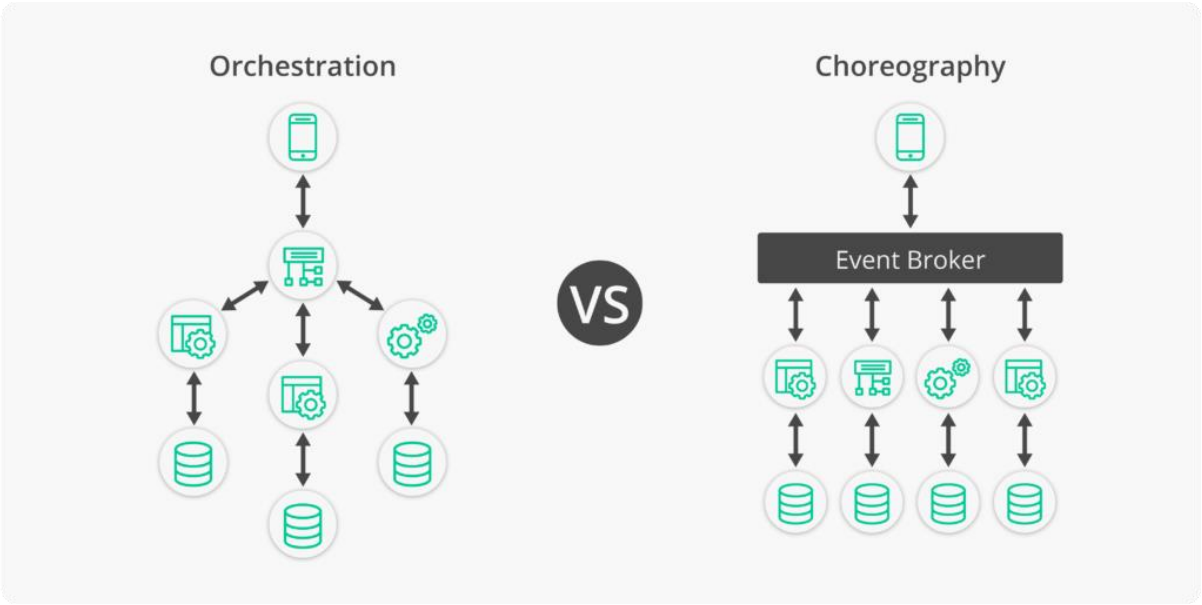
- MuleSoft Anypoint Platform.
- IBM MQ.
- Apache Kafka.
- Google Cloud Pub/Sub.
- Amazon MQ.
- Apache ActiveMQ.
- KubeMQ.
- ZeroMQ.

Orchestration & Choreography:

Orchestration is the automated configuration, management, and coordination of computer systems, applications, and services. Orchestration helps IT to more easily manage complex tasks and workflows.

IT teams must manage many servers and applications, but doing so manually isn't a scalable strategy. The need to combine multiple automated tasks and their configurations across groups of systems or machines increases. That's where orchestration can help.

Orchestration can be used to provision or deploy servers, assign storage capacity, create virtual machines, and manage networking, among other tasks.



In Orchestration, central system to control and call various Microservices to complete a task.

In Choreography, each microservice works like a state machine and reacts based on the inputs from other parts.

To choreograph microservices, you need a way of exchanging messages between microservices whenever something happens – you need an event broker. The moment a given microservice sends a message, they're done. Everything else happens in an asynchronous manner, without waiting for a response or worrying about what happens next. Each service is observing its environment, and any other service that subscribes to that channel of messages will know what to do from there.