**MICROSERVICE ARCHITECTURE**

Microservice Architecture is a special design pattern of Service-oriented Architecture. It is an open-source methodology.

All the processes will communicate with each other with the smallest granularity to implement a big system or service.

Microservice is a service-based application development methodology. In this methodology, big applications will be divided into smallest independent service units.

**APPLICATIONS:**

• Independent − Each microservice should be independently deployable.

• Coupling − All microservices should be loosely coupled with one another such that changes in one will not affect the other.

• Business Goal − Each service unit of the entire application should be the smallest and capable of delivering one specific business goal.

**Advantages**

• Small in size – it is small in size and easy to maintain than any other monolithic application.

• Focused − one microservice should be full stack in nature and should be committed to delivering only one business property.

• Autonomous − Each microservice should be an autonomous business unit of the entire application. Hence, the application becomes more loosely coupled, which helps to reduce the maintenance cost.

• Technology heterogeneity − Microservice supports different technologies to communicate with each other in one business unit, implementing a heterogeneous system, one can obtain maximum security, speed and a scalable system.

• Resilience − Resilience is a property of isolating a software unit. Resilience is another property which implements highly scalable and less coupled system.

• Ease of deployment − As the entire application is sub-divided into small piece of units, every component should be full stack in nature. All of them can be deployed in any environment very easily with less time complexity unlike other monolithic applications of the same kind.

**Disadvantages**

• Distributed system − Due to technical heterogeneity, different technologies will be used to develop different parts of a microservice. A huge set of skilled professionals are required to support this big heterogeneous distributed software.

• Cost − Microservice is costly, as you must maintain different server space for different business tasks.

• Enterprise readiness − Microservice architecture can be considered as a conglomerate of different technologies, as technology is evolving day-by-day.

**SCALING**

Scaling is a process of breaking down a software in different units. Scaling also defines in terms of scalability. It helps to improve security, durability, and maintainability of the application. We have three types of scaling procedures that is followed in the industries.

**X-Axis Scaling**

X-axis scaling is also called as horizontal scaling. In this procedure, the entire application is sub-divided into different horizontal parts. Normally, any web server application can have this type of scaling.

**Y-Axis Scaling**

Y-axis scaling is also called as a vertical scaling that includes any resource level scaling. Any DBaaS or Hadoop system can be considered to be Y-axis scaled. In this type of scaling, the users request is redirected and restricted by implementing some logic.

**Z-Axis Scaling**

X- and Y-axis scaling is pretty much easier to understand. However, one application can also be scaled at the business level, which is called as Z-axis scaling. Following is an example of scaling a cab service application in the different verticals of business units.

**BLUEPRINT**

Microservice implements SOA internally. In a broader sense, we can consider it as a subset of one SOA application.

**Rule & Workflow**

• High Cohesion − All the business models need to be sub-divided into the smallest business part as much as possible. Each service should be focused to perform only one business task.

• Independent − All the services should be full stack in nature and independent of each other.

• Business Domain Centric − Software will modularize according to the business unit and is not tier based.

• Automation − Testing deployment will be automated. Try to introduce minimal human interaction.

• Observable − Each service will be full stack in nature and they should be independently deployable and observable like an enterprise application.

**Team Management**

“Two Pizza Rule” is a kind of rule that restricts the number of attendees in a microservice development team. According to this rule, number of the team members of one application should be so small such that they can be fed by two pizza. Generally, the number should not be more than 8. As microservice is full stack in nature, the team is also full stack in nature. To increase the productivity, we need to build one team of maximum 8 members with all kinds of expertise required for that service.

**Task Management**

Task is an important role in software development life cycle. Developing a large-scale application can be broken down into several small units of task. Let us consider we need to develop one application such as Facebook. Then, “Log in” functionality can be considered as a task of the entire build process. Progress for each of these tasks need to be monitored properly under highly skilled professionals. Agile is the well-known process structure followed in the industries to keep up with good task management.

**CATEGORIES OF SERVICES:**

**Platform as a Service [PaaS]** − In this service-oriented architecture, the platform is given as a tool which can be customized according to the business needs. PaaS plays an important role in mobile application development. PaaS originally develops to provide a built-in architecture or infrastructure to developers. It reduces the higher level programming complexity in dramatically reduced time.

**Software as a Service [SaaS]** − Software as a Service is a software licensing business, where the software is centrally hosted and licensed on a subscription basis. SaaS can be accessed mainly through the browser and it is a very common architecture pattern in many business verticals such as Human Resource Management (HRM), Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), etc.

**Infrastructure as a Service [IaaS]** − Infrastructure plays a good role in IT industries. Using cloud computing, some of the organizations provide virtual infrastructure as their services. IaaS is very helpful for bringing agility, cost-effectiveness, security, performance, productivity, etc. in software development. Amazon EC2 and Microsoft Azure are the biggest examples of IaaS.

**Data as a Service [DaaS]** − Information technology deals with data and some of the top industry leaders believe that data will be the new sustenance of the society. DaaS is a type of service where data is shared with business conglomerates for research and analysis. DaaS brings simplicity, agility, and security in the data access layer. Following is an example of Oracle Data cloud, which can be accessed or licensed for your own business needs.

**Back End as a Service [BaaS]** − BaaS is also known as MBaaS, which means mobile back-end as a service. In this type of service, backend of the application will be provided to business units for their own business ventures. All push notifications, social networking services fall under this type of services. Facebook and Twitter are examples of well-known BaaS service provider.

**Security**

Security issue is associated with all kinds of services available in the market. Whatever the cloud you are using - private, public, hybrid, etc., security should be maintained at all levels. Entire security issue can be broadly sub-divided

• Security issue faced by service providers − This type of security issue is faced by the service providers such as Google, Amazon, etc. To ensure security protection, background check of the client is necessary especially of those who have direct access to the core part of the cloud.

• Security issue faced by consumers − Cloud is cost friendly, hence it is widely used across industries. Some organizations store the user details in third party data centers, and pull the data whenever required. Hence, it is mandatory to maintain security levels such that any private data of one customer should not be visible to any other users.

To prevent the above-mentioned security problems, following are some of the defensive mechanisms used by organizations.

• Deterrent Control threat to reduce cyber-attack.

• Preventive Control − Maintain high level authentication policy to access your cloud.

• Detective Control − Monitor users and detect any potential risk.

• Corrective Control − Work closely with different teams and fix the issues.

**COMPOSITION PATTERNS:**

Functional decomposition plays an important role in building your microservices. It provides agility, flexibility, and scalability to your application.

**Aggregator Pattern**

Aggregator pattern is the simplest web pattern that can be implemented while developing a micro-service. In this composition pattern, a simple web module will act as a load balancer, which means it will call different services as per requirements.

An aggregator can be again exposed as another service to the outer world, which can be consumed by others whenever required. While developing aggregator pattern web service, we need to keep in mind that each of our services A, B and C should have its own caching layers and it should be full stack in nature.

**Proxy Pattern**

Proxy microservice pattern is a variation of the aggregator model. In this model we will use proxy module instead of the aggregation module. Proxy service may call different services individually.

In Proxy pattern, we can build one level of extra security by providing a dump proxy layer. This layer acts like the interface.

**Chained Pattern**

As the name suggests, this type of composition pattern will follow the chain structure. Here, we will not be using anything in between the client and service layer. Instead, we will allow the client to communicate directly with the services and all the services will be chained up in a such a manner that the output of one service will be the input of the next service. Following image shows a typical chained pattern microservice.

One major drawback of this architecture is the client will be blocked until the entire process is complete. Thus, it is highly recommendable to keep the length of the chain as short as possible.

**Branch Microservice Pattern**

Branch microservice is the extended version of aggregator pattern and chain pattern. In this design pattern, the client can directly communicate with the service. Also, one service can communicate with more than one services at a time.

Branch microservice pattern allows the developer to configure service calls dynamically. All service calls will happen in a concurrent manner, which means service A can call Service B and C simultaneously.

**Shared Resource Pattern**

Shared resource pattern is a conglomerate of all types of patterns mentioned earlier. In this pattern, the client or the load balancer will directly communicate with each service whenever necessary. This is the most effective designing pattern followed widely in most organizations.