**MICROSERVICE ARCHITECTURE**

**What is Micro service architecture?**

Micro services is an architectural design for building a distributed application using containers. They get their name because each function of the application operates as an independent service. This architecture allows for each service to scale or update without disrupting other services in the application.



**Why do we need Microservices?**

Microservices are increasingly used in the development world as developers work to create larger, more complex applications that are better developed and managed as a combination of smaller services that work cohesively together for larger, application-wide functionality. Microservices enable IT organizations to be more agile. And build more scalable applications. Microservices translate into faster development and change cycles. As Figure 2 suggests, breaking down a large application into smaller services makes the development workflow faster

**What is monolithic architecture?**

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.

**What is SOA (Service-Oriented Architecture)?**

Service-Oriented Architecture (SOA) is a stage in the evolution of application development and/or integration. It defines a way to make software components reusable using the interfaces.

Formally, SOA is an architectural approach in which applications make use of services available in the network. In this architecture, services are provided to form applications, through a network call over the internet. It uses common communication standards to speed up and streamline the service integrations in applications. Each service in SOA is a complete business function in itself. The services are published in such a way that it makes it easy for the developers to assemble their apps using those services. Note that SOA is different from microservice architecture.

* SOA allows users to combine a large number of facilities from existing services to form applications.
* SOA encompasses a set of design principles that structure system development and provide means for integrating components into a coherent and decentralized system.
* SOA-based computing packages functionalities into a set of interoperable services, which can be integrated into different software systems belonging to separate business domains.

**What are the problems with monolith and SOA?**

1. Single Technology
2. Inflexible Deployment
3. Inefficient compute resources
4. Large and complex
5. Complicated and Expensive ESB
6. Lack of Tooling

**Characteristics of a Microservice Architecture**

1. Componentization via Services. ...
2. Organized around Business Capabilities. ...
3. Products not Projects. ...
4. Smart endpoints and dumb pipes. ...
5. Decentralized Governance. ...
6. Decentralized Data Management. ...
7. Infrastructure Automation. ...
8. Design for failure

**Defining Communication Patterns?**

1. 1-to-1 Synchronous
2. 1-to-1 Asynchronous
3. Pub-Sub/Event Driven

**What is synchronous in microservices?**

A synchronous microservice is one in which data moves to and from an endpoint of a service in a blocking interaction. A typical example of a synchronous data exchange is an HTTP request/response interaction, seen in Figure 1 below. When a request is made to an endpoint under HTTP, the caller is locked in the interaction until a response is received. The caller might receive the response in a mere millisecond or in a few seconds. Regardless of the application latency, the caller cannot move forward to the next task until the response is received. REST is a good example of a synchronous microservice.



**What is Asynchronous in microservices?**

An asynchronous microservice is one in which a request to a service and the subsequent response occur independently from each other. The general practice for implementing an asynchronous microservice is to use a message broker technology, such as Kafka or RabbitMQ, to act as a go-between for services, as seen in Figure 2 below. One service will publish a message to another service using the message broker. The intended service receives the message in its own time. The sending service is not locked to the broker. It simply fires and forgets.



**Testing Microservice?**

1. Unit Tests
2. Integrating Tests
3. End to End Tests

**Unit Testing**

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. This testing methodology is done during the development process by the software developers and sometimes QA staff.

**Integrating Testing**

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements.

**End to End Testing**

End-to-end testing is a methodology used in the software development lifecycle (SDLC) to test the functionality and performance of an application under product-like circumstances and data to replicate live settings. The goal is to simulate what a real user scenario looks like from start to finish.

**What Is Service Mesh Architecture?**

A service mesh provides a consistent, decentralized mechanism for managing communication between multiple services within a system. It can be used to implement features such as encryption, logging, tracing and load balancing, thereby improving security, reliability and observability.

Service mesh examples:-

Incoming traffic (called ingress), outgoing traffic (called egress), and traffic between services (mesh traffic). Each microservice instance (container or VM) has a dedicated Envoy proxy. The control plane is the management layer for the Envoy proxies.

**What is Logging and Monitoring?**

We will discuss one of the most important aspects of the Microservices system, which is the logging and monitoring. Without proper logging and monitoring, we are going to have a lot of problems with our system. It might lead to its failure too.

As I said earlier, this is extremely important in Microservices projects even more than monolith. The reason for that is that in Microservices, the flow goes through multiple processes. In this case it’s difficult to get holistic view of the system or entire flow. Most of the time in traditional monolith, we can examine logs and see what went wrong or what is happening. In this case with Microservices, it is difficult to stitch things together as it involve many processes.

We can look at a specific service and understand if there are any problems with it. It’s very difficult to figure out what is going on with all the services working together. All these problems are handled by well-designed logging and monitoring.

**Logging**

1. Recording the system’s activity. We document what the system did and what the users did and how did the system behave etc.. This is useful for analysing the system’s behaviour and making sure everything is good.
2. Auditing -We can see user’s behaviour and see what they did.
3. Documenting errors -Logging is the best way to write everything that has to do with errors happening in the system.

**Monitoring**

1. Based on system’s metrics -Monitoring tools look at the various metrics of the system, from infrastructure related metrics such as CPU, RAM, and Disk etc... To application related metrics such as requests per minute, orders per day etc... These metrics then available for users via sophisticated dashboards.
2. Alerting -We can define alerts that will be raised when a specific metric goes outside the normal range. For example, when CUP goes high, and Memory usage is high, we can configure monitoring tools to send alerts to a predefined group of people, so they can handle.

As you can see both should implemented in order to make sure our Microservices system is reliable and stable.

**Implementing Logging**

As I said, logging should provide holistic view on the system. It should allow tracing end to end flow. End to end means from the service where the flow originated to the service that ended and with all the services in between included. Logging should contain as much information as possible. So make sure log whatever information you can think of, even things that are first look might seem useless.

For example we definitely want to look at least the timestamp, severity model, message, machine id and may be ip address etc. Then you should be able to filter based on these values.

Logging in Microservices architecture should work quite differently from the traditional logging.

In a traditional logging, if we have, let’s say two services, then each one of them has its own logging infra. The service is using its own logging library and it logs to service’s own repository, may be in to file or database. This is easiest way and when our application is composed of a single process, then there is no problem with such implementation.

However, with Microservices, we have some problems implementing the logging like that. As we want to look at the logs and trace an end to end flow, that would be a problem. Also using different logging libraries means we might utilize different logging formats. For example, we may have JSON in our file and plain text in another file. In addition, we cannot aggregate easily to see how many error per day, how many DB access per day etc.

**Implementing Monitoring**

Monitoring looks at metrics and detects anomalies. It also provides simplified view of the system status usually using dashboards that presents the important metrics in an easy to grasp format.

In addition, we can set alerts to trigger when there is a problem and relevant people can notify. There are two types of monitoring,

Here we monitors the underlines servers. This means, we usually monitor things such as CPU, RAM, Disk, Network and any other metrics that are infrastructure related. Then we can get alerts when a problem is detected in the infrastructure such as high CPU or high disk IO etc.

**When should you not use a microservice?**

1. Loose coupling.
2. Surgically scalable.
3. Enhanced development productivity.
4. Alignment with domain-driven design (DDD)
5. Extensibility.
6. More agile enterprise.