03

Microservices

WHAT YOU WILL LEARN

* What is microservices
* Challenges of Microservices Architecture
* Difference between Monolith and Microservice
* Advantages of microservice

# What is microservices

* Microservices are the small services that work together.
* The microservice architectural style is an approach to develop a single application as a suite of small services. Each microservice runs its process and communicates with lightweight mechanisms.
* These are the services which are exposed by REST.
* The services must be cloud-enabled.

# Challenges of Microservices Architecture

Top challenges that an organization face:

* **Dynamic Scale up and Scale Down:** The loads on the different microservices may be at a different instance of the type. As well as auto-scaling up your microservice should auto-scale down. It reduces the cost of the microservices. We can distribute the load dynamically.
* **Monitoring:** The traditional way of monitoring will not align well with microservices because we have multiple services making up the same functionality previously supported by a single application. When an error arises in the application, finding the root cause can be challenging.
* **Fault Tolerance:** Fault tolerance is the individual service that does not bring down the overall system. The application can operate at a certain degree of satisfaction when the failure occurs. Without fault tolerance, a single failure in the system may cause a total breakdown. The circuit breaker can achieve fault tolerance.
* **DevOps Culture:** Microservices fits perfectly into the DevOps. It provides faster delivery service, visibility across data, and cost-effective data. It can extend their use of containerization switch from Service-Oriented-Architecture (SOA) to Microservice Architecture (MSA).

# Difference between Monolith and Microservice

|  |  |
| --- | --- |
| Monolith | Microservice |
| * Monolithic architecture is built as one large system and is usually one code-base | * Microservices architecture is built as small independent module based on business functionality |
| * It is not easy to scale based on demand | * It is easy to scale based on demand. |
| * It has shared database | * Each project and module has their own database |
| * Large code base makes IDE slow and build time gets increase. | * Each project is independent and small in size. So overall build and development time gets decrease. |
| * It extremely difficult to change technology or language or framework because everything is tightly coupled and depend on each other | * Easy to change technology or framework because every module and project is independent |

# **Principles of microservices**

**Single responsibility:**

Each microservice must have a single responsibility and provide a single functionality.

The database is also decentralized and generally, each microservice has its own database.

**Built around business capabilities:**

A microservice shall never restrict itself from adopting appropriate technology stack or backend database storage which is most suitable for solving the business purpose i.e. each microservice can use different technology based on business requirements.

**Design for failure:**

Microservices must be designed with failure cases in mind.

his architecture and going down of one microservice should not affect the whole system, other functionalities must remain accessible to the user.

# Advantages of microservice

* It is easy to manage as it is relatively smaller in size.
* If there’s any update in one of the microservices, then we need to redeploy only that microservice.
* Each microservice can use different technology based on the business requirements.
* If a particular microservice goes down due to some bug, then it doesn’t affect other microservices and the whole system remains intact, continues providing other functionalities to the users.

# **Disadvantages of microservices**

* Its complexity increases with the increase in number of microservices.
* Microservices are less secure relative to monolithic applications due to the inter-services communication over the network.
* Debugging is difficult as the control flows over many microservices and to point out why and where exactly the error occurred is a difficult task.

# Service Discovery

The service discovery is the automatic detection of devices and services over the network. In other words, service discovery is how an application and microservices connect in the distributed environment. Service discovery implementations include both:

* The **central server** that maintains a global view of the address.
* The **clients** that connect to the central server can update and retrieve the address.

There are **two** discovery patterns: **Client-side discovery** and **Server-side discovery**.

* **Client-side discovery:** In the Client-side discovery, client is responsible for determining the network location of available services. The client uses a **load-balancing algorithm** to select one of the available services and make a request. **Netflix OSS** is an example of a client-side discovery pattern.
* **Server-side discovery:** In the server-side discovery, the client makes an HTTP request to a service through a load balancer. The load balancer contacts to service registry and route each request to an available service instance. Similar to client-side discovery, service instances are registered and deregistered with the service registry. The **AWS ELB** (Elastic Load Balancer) is an example of server-side discovery. ELB balances the external traffic from the internet.

Diagram

Description automatically generated

## Microservice intercommunication

interact one service with another service because one microservice should not directly access the other Microservices database as per the Microservice’s recommendation.

There are two types of inter-service communication in Microservices:

* **Synchronous communication -** In synchronous communication, one service will communicate with another service through the rest endpoint using HTTP or https protocol. In this approach, calling service will wait until the caller service responds.

**Example –**

1. Rest Template
2. Feign
   1. **Spring Cloud OpenFeign** provides OpenFeign integrations for Spring Boot apps through auto-configuration and binding to the Spring Environment.
   2. The Feign is a declarative web service (HTTP client) developed by **Netflix**. Its aim is to simplify the HTTP API clients.
   3. the **pom.xml** and add the **Feign**dependency. Feign inherits from the **Netflix**.
   4. Once the dependency is added, **enable** the Feign to scan the clients by adding the annotation **@EnableFeignClients**in the**Application.java**file.
   5. Add an annotation **@FeignClient.**Pass the attributes **name** and **URL**.

* **Asynchronous communication** - In asynchronous communication, one service will communicate with another service through the asynchronous messaging. The calling service will not wait to respond by the caller service.

**Example** –

1. Apache Kafka
2. Active MQ.

# API Gateway

The API Gateway is a server. It is a single-entry point into a system.

API Gateway encapsulates the internal system architecture. It provides an API that is tailored to each client.

Diagram

Description automatically generated

**An API Gateway Responsibilities:**

* Security
* Caching
* API composition and processing
* Managing access quotas
* API health monitoring
* Versioning
* Routing

**Advantages of API Gateway**

* The most important advantage of API Gateway is that it encapsulates the internal structure of the application.
* Rather than invoking the specific service, the client directly talks to the API Gateway.
* It reduces the number of round trips between client and application.
* It provides each kind of client with a specific API.

**Disadvantages**

* It requires routing rules.
* There is a possibility of a single point of failure.
* Risk of complexity due to all the API rules are in one place.

# Load Balancer

Load balancing Is the process of distributing load from one server to multiple servers.

**Load Balancing Algorithm**

1. Roud-robbin

a simplified example, assume that an enterprise has a cluster of three servers: Server A, Server B, and Server C.

• The first request is sent to Server A.  
• The second request is sent to Server B.  
• The third request is sent to Server C.

1. Sticky-session

With sticky sessions, **a load balancer assigns an identifying attribute to a user, typically by issuing a cookie or by tracking their IP details**. Then, according to the tracking ID, a load balancer can start routing all of the requests of this user to a specific server for the duration of the session.

1. Ip-hashing

Source IP hash load balancing algorithm that combines source and destination IP addresses of the client and server to generate a unique hash key.

Load balancing can be of two types:

1. Server-side Load Balancing

In Server-side load balancing, the **instances of the service are deployed on multiple servers** and then a **load balancer is put in front** of them. It is generally a hardware load balancer. All the incoming requests traffic firstly comes to this load balancer acting as a middle component. It then decides to which server a particular request must be directed to based on some algorithm.

Diagram

Description automatically generated

**Disadvantages of Server-side load balancing**

1. Server-side load balancer acts as a **single point of failure** as if it fails, all the instances of the microservice becomes inaccessible as only load balancer has the list of servers.
2. Since each microservice will have a separate load balancer, the **overall complexity of the system increases,** and it becomes **hard to manage**.
3. The **network latency increases** as the number of hops for the request increases from one to two with the load balancer, one to the load balancer and then another from load balancer to the microservice.
4. Client-side Load Balancing

The instances of the service are deployed on multiple servers. Load balancer's logic is part of the client itself; it holds the list of servers and decides to which server a particular request must be directed to base on some algorithm. These client-side load balancers are also known as **software load balancers**.

Diagram

Description automatically generated

**Disadvantages of Client-side load balancing**

1. The load balancer's logic is mixed up with the microservice code.

**What is Netflix's Ribbon**

Following are the **features of Ribbon**:

1. **Load balancing:** It provides client side load balancing functionality.
2. **Fault tolerance:** It can be used to determine whether the servers are up or not and can also detect those servers that are down and hence, ignore them for sending the further requests.
3. **Configurable load balancing rules:** By default ribbon uses **RoundRobinRule** for distributing requests among servers. In addition to it, it also provides **AvailabilityFilteringRule** and **WeightedResponseTimeRule**. We can also define our custom rules as per our needs.
4. It **supports multiple protocols** like HTTP, TCP, UDP etc.

Example-

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

# Fault Tolerance

Consider a scenario in which six microservices are communicating with each other. The **microservice-5** becomes down at some point, and all the other microservices are directly or indirectly depend on it, so all other services also go down.

The solution to this problem is to use a **fallback** in case of failure of a microservice. This aspect of a microservice is called **fault tolerance**.

**Fault tolerance** can be achieved with the help of a **circuit breaker**. It is a pattern that wraps requests to external services and detects when they fail. If a failure is detected, the circuit breaker opens. All the subsequent requests immediately return an error instead of making requests to the unhealthy service. It monitors and detects the service which is down and misbehaves with other services. It rejects calls until it becomes healthy again.

**Hystrix**