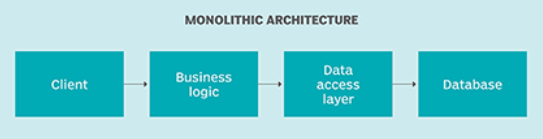
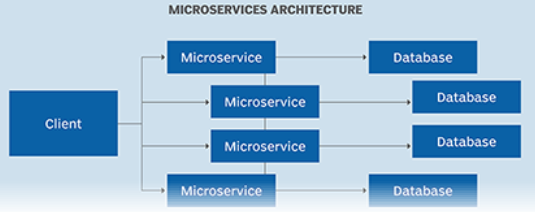
* **Difference Between Monolith and Microservices Architecture.**
* If we decide to develop all the components of our application in a single project then it is called **Monolith Architecture** based project/application.



* Microservices focus on a **single business domain/functionality** that can be implemented as fully independent deployable services and implement them on different technology stacks.



* **Microservice Architecture (MSA) should consist of the following components**



**1. Clients**

The architecture starts with different types of clients, from different devices trying to perform various management capabilities such as search, build, configure etc.

**2. Identity Providers**

These requests from the clients are then passed on to the identity providers who authenticate the requests of clients and communicate the requests to API Gateway. The requests are then communicated to the internal services via well-defined API Gateway.

**3. API Gateway**

Since clients don’t call the services directly, API Gateway acts as an entry point for the clients to forward requests to appropriate microservices.

The advantages of using an API gateway include:

* All the services can be updated without the clients knowing.
* Services can also use messaging protocols that are not web-friendly.
* The API Gateway can perform cross-cutting functions such as providing security, load balancing etc.

After receiving the requests of clients, the internal architecture consists of microservices which communicate with each other through messages to handle client requests.

4. Messaging Formats

There are two types of messages through which they communicate:

* Synchronous Messages: In the situation where clients wait for the responses from a service, Microservices usually tend to use REST (Representational State Transfer) as it relies on a stateless, client-server, and the HTTP protocol. This protocol is used as it is a distributed environment each and every functionality is represented with a resource to carry out operations
* Asynchronous Messages: In the situation where clients do not wait for the responses from a service, Microservices usually tend to use protocols such as AMQP, STOMP, MQTT. These protocols are used in this type of communication since the nature of messages is defined and these messages have to be interoperable between implementations.

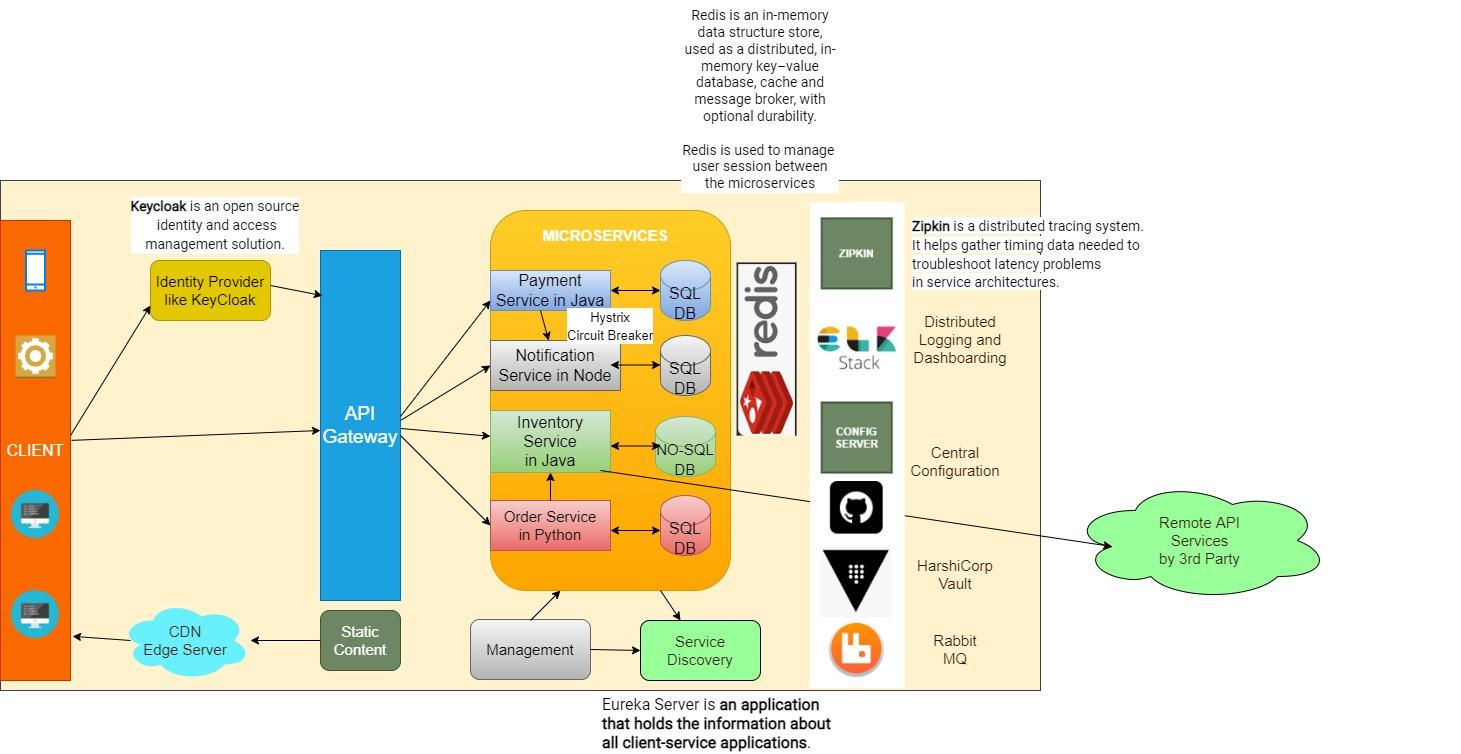
**5. Data Handling**

Well, each Microservice owns a private database to capture their data and implement the respective business functionality.Also, the databases of Microservices are updated through their service API only.

**6. Static Content**

After the Microservices communicate within themselves, they deploy the static content to a cloud-based storage service that can deliver them directly to the clients via Content Delivery Networks (CDNs).

**8. Service Discovery**

Acts as a guide to Microservices to find the route of communication between them as it maintains a list of services on which nodes are located.

* **Advantages VS Disadvantage of Microservices Architecture.**

| **ADVANTAGES** | **DISADVANTAGES** |
| --- | --- |
| Freedom to use different technologies | Increases troubleshooting challenges |
| Each microservice focuses on single business capability | Increases delay due to remote calls |
| Supports individual deployable units | Increased efforts for configuration and other operations |
| Allows frequent software releases | Difficult to maintain transaction safety |
| Ensures security of each service | Tough to track data across various service boundaries |
| Multiple services are parallelly developed and deployed | Difficult to move code between services |

* **Key Projects from SpringBoot for Microservices Architecture.**
* **Spring Cloud Config Server**

It is used to **externalize the configuration of our microservices** into a centralized place.

If there are any changes in the configuration, our applications should be updated without the need to restart them.

There are many options to implement this Centralized Configuration eg: Using a Git Repository, or using HashiCorp Consul etc.

* **Spring Cloud Bus**

This module contains a lightweight message broker implementation, which is mainly **used to broadcast some messages to the other services**.

Whenever any configuration changes in the config server the service bus to broadcast configuration changes in our Config Server.

* **Spring Cloud Netflix Eureka**

This module is used to implement the **Service Registry and Discovery Pattern** in Microservices architecture.

As there can be many independent Microservices, we need a reliable way to scale the services and provide inter-service communication instead of hard-coding the service information.

Other alternatives include Spring Cloud Consul which uses Hashicorp’s Consul or Spring Cloud Zookeeper for Apache Zookeeper

* **Spring Cloud Circuit Breaker**

Inter-service Communication is common in the Microservice architecture, if one of the services is down, the other service which is communicating with it, should be able to handle this failure gracefully.

Spring Cloud Circuit Breaker provides an abstraction over different Circuit Breaker implementations like Reslience4J, Hystrix, Sentinel etc.

* **Spring Cloud Sleuth**

In Microservice Architecture, if there is any error, it’s hard to debug and trace it, Spring Cloud Sleuth provides us with the functionality to trace the inter-service calls.

* **Spring Cloud Stream**

This module mainly allows us to implement asynchronous communication between our microservices using event-driven architecture.

RabbitMQ can be used as a message broker to implement some event driven microservices.

* **Spring Cloud Gateway**

This library helps us to implement the API Gateway pattern, by hiding the complexity of our microservices from the external clients.

If a client wants to connect to one of our services, all the traffic will be going through the API Gateway and the gateway will route the request to the appropriate service.

This library also handles cross-cutting concerns like Security, Monitoring, Rate-limiting and Resiliency.

We also secure our microservices using Keycloak as Authorization Server together with the Spring Cloud Gateway.

**INTER Service Communication using different clients**

*RestTemplate* uses Java Servlet API and is therefore synchronous and blocking. Contrarily, *WebClient* is asynchronous and will not block the executing thread while waiting for the response to come back. Only when the response is ready will the notification be produced.

*RestTemplate* will still be used. In some cases, the non-blocking approach uses much fewer system resources compared to the blocking one. Hence, in those cases, *WebClient* is a preferable choice.

**Feign** is a declarative REST client whose purpose is to make REST calls easier.

Feign provides an HTTP request template. By writing a simple interface and inserting annotations, you can define the parameters, format, address and other information of the HTTP request.

Feign will completely proxy HTTP requests. We only need to call it like a method to complete service requests and related processing.

Feign can be used in combination with Eureka and Ribbon to support load balancing.

### **Distributed Logging using ELK Stack**

Centralized Logging for our microservices can be implemented using ELK Stack (Elasticsearch, Logstash and Kibana)

**Session Management in Microservice Architecture.**