1.What is Microservices?

Microservices is a software architecture pattern where an application is built as a collection of small, independent, and loosely coupled services. Each service is responsible for a specific task or functionality, communicates through lightweight protocols like HTTP/REST, and can be deployed, scaled, and maintained independently.

2.Write main components of Microservices?

The main components of microservices architecture are:

Services: small, independent, and decoupled components that perform a specific business capability.

API Gateway: an entry point that manages the access to the microservices, routing the requests to the corresponding service and handling authentication, rate-limiting, and other cross-cutting concerns.

Service Registry: a centralized registry that keeps track of the location and health status of the microservices, allowing them to discover and communicate with each other.

Configuration Server: a centralized server that stores the configuration information for the microservices, allowing them to be easily updated without redeploying the entire application.

Load Balancer: a component that distributes the incoming requests across multiple instances of the same service to improve performance and availability.

3.How is Microservice architecture different from Monolithic architecture?

Microservice architecture is different from monolithic architecture in several ways:

Monolithic architecture consists of a single, large, and tightly coupled application, while microservices architecture is made up of several small and independent services.

In a monolithic architecture, all the code is packaged and deployed as a single unit, while in microservices, each service can be developed, tested, and deployed independently.

Monolithic architecture typically uses a single database for all the components, while microservices may use different databases for each service.

Monolithic architecture can be easier to develop, but it can become more complex and difficult to scale and maintain as the application grows. Microservices architecture is more flexible and scalable, but requires more upfront planning and design.

4. What are the benefits(pros) and drawbacks(cons) of Microservices?

The benefits of microservices include:

Scalability: Microservices can be independently scaled up or down based on demand, improving resource utilization and reducing costs.

Flexibility: Microservices allow for greater flexibility in technology selection, development, and deployment, as each service can be developed, tested, and deployed independently.

Resilience: Microservices can be designed to be resilient to failure, as failures in one service do not affect the other services.

Agility: Microservices can enable faster development and deployment cycles, as changes to one service can be made without affecting the entire application.

The drawbacks of microservices include:

Complexity: Microservices can introduce additional complexity in terms of communication, deployment, and monitoring, which can require additional effort and resources.

Testing: Testing and integration of microservices can be more complex than monolithic architecture.

Increased Infrastructure: Each microservice will require its own infrastructure, which can increase operational costs.

5.Explain the working of Microservice Architecture

The working of microservice architecture typically involves the following steps:

Each service is developed, tested, and deployed independently, using its own technology stack and database if needed.

The API Gateway handles incoming requests, routing them to the corresponding service based on the request URI or other criteria.

The Service Registry keeps track of the location and health status of the microservices, allowing them to discover and communicate with each other.

The Configuration Server stores the configuration information for the microservices, allowing them to be easily updated without redeploying the entire application.

The Load Balancer distributes the incoming requests across multiple instances of the same service to improve performance and availability.

Monitoring and logging tools are used to monitor the health and performance of the microservices, as well as to identify and troubleshoot any issues.

6.Write difference between Monolithic and Microservices Architecture

Differences between Monolithic and Microservices Architecture:

Monolithic architecture consists of a single, large and tightly coupled application, while microservices architecture is made up of several small and independent services.

In a monolithic architecture, all the code is packaged and deployed as a single unit, while in microservices, each service can be developed, tested, and deployed independently.

Monolithic architecture typically uses a single database for all the components, while microservices may use different databases for each service.

Monolithic architecture can be easier to develop, but it can become more complex and difficult to scale and maintain as the application grows. Microservices architecture is more flexible and scalable, but requires more upfront planning and design.

7.Explain spring cloud and spring boot

Spring Boot is an open-source Java-based framework used to create standalone and production-grade Spring-based applications with minimal configuration. It provides a comprehensive set of tools for building modern, cloud-native applications. Spring Cloud is a set of tools

and frameworks built on top of Spring Boot, designed to simplify the development of distributed systems and microservices-based applications. It provides features like service discovery, circuit breaker, load balancing, distributed tracing, and configuration management, making it easier to develop and manage large-scale applications.

8.Explain how you can override the default properties of Spring boot projects

Spring Boot allows for overriding default properties in a variety of ways, including:

Through the use of application.properties or application.yml files, where properties can be added, changed or removed from the default configuration.

Through the use of external configuration files, which can be specified through the command line or environment variables.

Through the use of system properties or environment variables, which can be used to override any property in the Spring Boot application.

9.What issues are generally solved by spring clouds?

Spring Cloud helps to solve several common issues in developing microservices-based applications, including:

Service Discovery: Spring Cloud provides service discovery mechanisms that allow services to find and communicate with each other without hardcoding network locations.

Configuration Management: Spring Cloud provides a centralized configuration server that can manage configurations for multiple services and allows for easy updates and version control.

Load Balancing: Spring Cloud provides load balancing features that can distribute traffic across multiple instances of a service to improve performance and reliability.

Circuit Breaker: Spring Cloud provides circuit breaker patterns to prevent cascading failures in a distributed environment.

Distributed Tracing: Spring Cloud provides distributed tracing capabilities that allow developers to trace requests as they pass through multiple services.

10.What do you mean by Cohesion and Coupling?

Cohesion and Coupling are two important concepts in software engineering:

Cohesion refers to the degree to which the elements within a module or component work together to achieve a common goal. High cohesion means that elements within a module are closely related and work together to achieve a common purpose, while low cohesion means that elements within a module have little relationship to each other and perform unrelated tasks.

Coupling refers to the degree to which one module or component depends on another. Tight coupling means that modules are highly dependent on each other, while loose coupling means that modules have minimal or no dependency on each other. High coupling can make it difficult to change or modify a module without affecting other modules, while low coupling can make the system more modular and easier to maintain.

11.Write the fundamental characteristics of Microservice Design

Fundamental characteristics of Microservice Design include:

Small and independent: Microservices are small and independently deployable services that perform a specific business function.

Autonomous: Each microservice is self-contained and operates independently, without depending on other services.

Highly cohesive and loosely coupled: Each microservice has a well-defined and focused responsibility and communicates with other services through well-defined APIs, reducing the dependency between services.

Resilient: Microservices are designed to be resilient in the face of failures, using techniques like fault-tolerance, circuit breakers, and retries to handle failures.

Scalable: Microservices can be scaled horizontally, allowing the system to handle high levels of traffic and load.

Continuous Delivery: Microservices are developed and deployed independently, allowing for continuous delivery of new features and updates.

12.What are the challenges that one has to face while using Microservices?

Some challenges that one may face while using Microservices include:

Increased complexity: Microservices architecture can be more complex and difficult to design and manage compared to monolithic architecture.

Distributed systems: Microservices are distributed systems, and this can make it challenging to ensure consistent data and maintain transactional integrity.

Testing: Testing microservices can be challenging as it requires testing each service independently and as part of the larger system.

Service discovery and communication: Microservices need to discover and communicate with each other, and this can be a challenge in a dynamic and constantly changing environment.

Monitoring: Monitoring and debugging microservices can be challenging, as the services are distributed and may use different technologies.

Some common challenges faced while using Microservices include:

Managing inter-service communication and APIs.

Ensuring data consistency across services.

Ensuring the availability and reliability of services.

Ensuring the security and access control of services.

Managing multiple deployment environments.

13.Explain how independent microservices communicate with each other

Independent microservices communicate with each other through well-defined APIs. Typically, microservices use a combination of synchronous and asynchronous communication mechanisms. Synchronous communication involves the request-response model, where the service sends a request to another service and waits for a response. Asynchronous communication involves the use of message brokers, where a service sends a message to a message broker, and the message broker delivers the message to the intended recipient.

14.Explain the term Eureka in Microservices

Eureka is a service discovery and registration tool in Microservices architecture. It provides a registry of all the services in the system and allows services to register and discover each other. Eureka works in

combination with other tools in the Spring Cloud ecosystem to provide a comprehensive set of tools for developing and managing Microservices-based applications.

15.What is eureka server?

Eureka Server is a service discovery tool in Microservices architecture. It provides a registry of all the services in the system and allows services to register and discover each other. Eureka Server acts as a central point of contact for all the services in the system.

16.What is eureka client?

Eureka Client is a library that enables services to register themselves with the Eureka Server and discover other services registered with the Eureka Server. Eureka Client libraries are available for various programming languages and frameworks.

17.How to you change server port number for microservice?

To change the server port number for a microservice, you can specify the desired port number in the application.properties or application.yml file of the microservice. For example, to change the port number to 8081, you can add the following line to the application.properties file:

server.port=8081

18.What are some common Microservices design principles?

Some common Microservices design principles include:

Single Responsibility Principle (SRP): Each microservice should have a single responsibility or business function.

Loose Coupling: Microservices should be loosely coupled and communicate with each other through well-defined APIs.

High Cohesion: Each microservice should have a well-defined and focused responsibility.

Autonomy: Each microservice should be self-contained and operate independently, without depending on other services.

Resiliency: Microservices should be designed to be resilient in the face of failures.

Agility: Microservices should be developed and deployed independently, allowing for continuous delivery of new features and updates.

19.Explain the way to implement service discovery in microservices architecture

There are several ways to implement service discovery in Microservices architecture, including:

Client-side discovery: In client-side discovery, each client is responsible for discovering the location of the service it needs to communicate with. This can be implemented using libraries like Eureka Client or Consul.

Server-side discovery: In server-side discovery, a load balancer or gateway service is responsible for discovering the location of the services and routing requests to the appropriate service. This can be implemented using tools like Netflix Zuul or Spring Cloud Gateway.

Hybrid approach: In a hybrid approach, a combination of client-side and server-side discovery is used to achieve the benefits of both approaches.

20.Explain the importance of reports and dashboards in microservices

Reports and dashboards are important in Microservices architecture as they help monitor the health, performance, and usage of the individual microservices and the system as a whole. Reports and dashboards provide real-time insights into the behavior of the microservices, allowing developers and operators to quickly identify and resolve issues, optimize performance, and make data-driven decisions.

21.What are the advantages of using Spring Cloud?

Some advantages of using Spring Cloud include:

Simplified Microservices Development: Spring Cloud provides a suite of tools and libraries that simplify the development of Microservices, including service registration and discovery, load balancing, and distributed tracing.

Consistency: Spring Cloud provides a consistent set of abstractions for developing Microservices, making it easier to maintain and evolve the system over time.

Integration: Spring Cloud integrates with a wide range of technologies, including Docker, Kubernetes, and Apache Kafka, making it easy to deploy and operate Microservices in any environment.

Open Source: Spring Cloud is open-source software, which means it is free to use and can be customized to meet specific needs.

22.What does one mean by Service Registration and Discovery? How is it implemented in Spring Cloud?

Service Registration and Discovery is the process of registering and discovering the location of Microservices in a distributed system. In Spring Cloud, service registration and discovery are implemented using the Eureka Server and Eureka Client libraries. The Eureka Server acts as a registry of all the services in the system, while the Eureka Client library allows services to register themselves with the Eureka Server and discover other services registered with the Eureka Server.

23.What is service registry and used for?

Service registry is a component of Microservices architecture that acts as a directory of all the services in the system. It provides a centralized place where services can register themselves and discover other services in the system. Service registry is used to enable Service Registration and Discovery, which is a key requirement for Microservices architecture.

24.What does one mean by Load Balancing? How is it implemented in Spring Cloud?

Load Balancing is the process of distributing incoming network traffic across multiple servers or nodes. Load balancing is used in Microservices architecture to distribute traffic across multiple instances of a service, ensuring that each instance is used efficiently and that the system can handle high traffic loads. In Spring Cloud, load balancing is implemented using the Ribbon library, which provides client-side load balancing for HTTP and TCP-based services. Ribbon allows services to discover and load balance across multiple instances of a service, improving availability and scalability.