MICRO SERVICES.

**1. Monolithic Architecture**

**Monolithic Architecture** refers to a traditional, unified approach where all components of the software application are combined and packaged into a single unit or codebase.

All modules are deployed in the same instince.

When one module changed whole application must be deployed.

Microservices architecture.

**Microservices Architecture** is a modern software design approach where the application is broken down into small, loosely coupled, independently deployable services, each of which focuses on a specific business function. Each service is typically independent and communicates with other services over a network.

All modules are deployed I individual instances.

When one module changed each one can be deployed individually.

Practical micro services:



Step-1:

Project-1:

1.build department and project entities.

2.each has.

Department:

Getbyprojectcode.

Savedeprtment.

3.Project methods.

Getbyprojectcode.

Savedepartment.

Project-2:

Build only department entity.

2.ch has.

Department:

Getbyprojectcode.

Savedeprtment.

Project-3:

Build only project entity.

Project methods.

Getbyprojectcode.

Savedepartment.

\*\*in project where it has only department entity there we need project table we connect it using micro services.

Pom.xml needed.

1.add cloud dependency in dependency management.

2.add its Greenwich version along with java version.

3.add the feign dependency.

POM.xml for department entity:

1.springboot version>3.4

2.in properties add cloud version below java 7.(it should be 2024.0.0)

3.add import open feign version and make sure version is 4.2.0.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.4.0</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.example</groupId>  
 <artifactId>mvcdatabase</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>mvcdatabase</name>  
 <description>Demo project for Spring Boot</description>  
 <url/>  
  
 <properties>  
 <java.version>17</java.version>  
 <spring.cloud.version>2024.0.0</spring.cloud.version>  
 </properties>  
  
 <dependencies>  
 <!-- Spring Boot Dependencies -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.postgresql</groupId>  
 <artifactId>postgresql</artifactId>  
 <version>42.6.0</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-mail</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-validation</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>1.18.34</version>  
 <scope>provided</scope>  
 </dependency>  
 <dependency>  
 <!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-openfeign -->  
  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-openfeign</artifactId>  
 <version>4.2.0</version>  
  
  
  
 </dependency>  
  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>

Next step:

@SpringBootApplication  
@EnableFeignClients  
@ImportAutoConfiguration(FeignAutoConfiguration.class)  
public class DepartmentMicroservicesApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(DepartmentMicroservicesApplication.class, args);  
 }  
  
}

Step-2:



Nextstep -3:

Create a project interface in project package and redirect it into project usin feign client and url:

@FeignClient(url = "http://localhost:8083",value="projectfeignclient",path="/project")  
public interface Projectfeign {  
  
  
}

Next step -4:

Get feign response making get mapping as interface response:

@GetMapping("/project/{projectCode}")  
 feign.Response getprojectbyid(@PathVariable("projectCode") Long projectcode);  
  
}

Next-step-5:

Go to the service layer.

Before:

We get this

Project project1=pr.findbyProjectcode(department1.getid())

After:

We get this from feign client we created.

Response response = projectfeign.getprojectbyid(department1.getDepartmentcode());

Next step-5:

Create string body

String body=response.body().toString();

Next step-6:

Gson g=new Gson();

Next step-7:

Project project1=g.fromJson(body, Project.class);

Now we can push this entity dto using micro services into department.

Interview questions on micro services?

A,

Certainly! Below is an expanded list of interview questions for each level, covering a wide range of topics in microservices.

Basic Level

1. What is a Microservice Architecture?

Answer: Microservice architecture is a style of architecture where an application is divided into a set of loosely coupled services, each focusing on a specific business capability and communicating over APIs.

2. What are the benefits of using Microservices?

Answer: Benefits include scalability, flexibility, improved fault isolation, faster development cycles, and better maintainability due to smaller and independent services.

3. What is the role of an API Gateway in Microservices?

Answer: An API Gateway serves as a single entry point for all client requests, routing them to the appropriate microservice, handling load balancing, authentication, and rate limiting.

4. What is Service Discovery in Microservices?

Answer: Service Discovery allows microservices to dynamically locate other services in a system. It is typically implemented through a registry where services register and look for each other.

5. What is the difference between Monolithic and Microservices architectures?

Answer: A monolithic architecture integrates all components into one application, while microservices break the application into small, independent services.

6. What are the basic components of a Microservice-based architecture?

Answer: Key components include services, APIs, databases, API gateways, and service discovery mechanisms.

7. What is the purpose of a Service Registry?

Answer: A Service Registry helps maintain a list of available services, allowing services to discover and communicate with each other dynamically.

8. What is the role of a Database in a Microservices architecture?

Answer: Each microservice usually has its own database to maintain loose coupling between services and avoid sharing data directly.

9. What are some common communication methods used in Microservices?

Answer: REST, HTTP, gRPC, and messaging queues (like RabbitMQ, Kafka) are common methods for service communication.

10. What is a Container and how is it related to Microservices?

Answer: Containers (e.g., Docker) package microservices along with their dependencies, allowing them to be deployed and run consistently across different environments.

Intermediate Level

1. How do Microservices communicate with each other?

Answer: Microservices communicate through REST APIs, HTTP, gRPC, or message brokers (e.g., Kafka). Communication can be synchronous or asynchronous depending on the use case.

2. What is the Circuit Breaker pattern in Microservices?

Answer: The Circuit Breaker pattern prevents a failing service from causing cascading failures by monitoring service calls and “opening” the circuit when failures exceed a threshold, allowing the system to recover.

3. What are some common challenges with Microservices?

Answer: Challenges include service coordination, managing distributed data, inter-service communication, testing, monitoring, and ensuring consistency across services.

4. What is the Event-Driven Architecture in Microservices?

Answer: Event-Driven Architecture enables services to communicate via events instead of direct API calls, allowing for decoupling and more scalable, reactive systems.

5. What is the role of API Gateway in managing traffic for Microservices?

Answer: The API Gateway routes requests to the correct service, handles load balancing, enforces security policies, and can perform tasks like authentication, rate limiting, and logging.

6. What is the Database per Service pattern?

Answer: In this pattern, each microservice has its own database, allowing each service to maintain its own data and ensure that services are decoupled from each other.

7. How do you implement Authentication and Authorization in Microservices?

Answer: OAuth2 and JWT (JSON Web Tokens) are commonly used for authentication and authorization. The API Gateway or a dedicated Authentication Service handles user authentication, and JWT tokens are used for secure communication.

8. What is the concept of “Loose Coupling” in Microservices?

Answer: Loose coupling means that services are independent and changes to one service don’t directly impact others. Each service is responsible for its own domain logic and data.

9. What is the difference between Synchronous and Asynchronous communication in Microservices?

Answer: Synchronous communication happens when services wait for a response after making a request (e.g., REST API), while asynchronous communication allows services to send requests and continue without waiting for a response (e.g., messaging queues).

10. How do you handle versioning in Microservices APIs?

Answer: API versioning can be handled via URL paths (e.g., /v1/resource), request headers, or through the content type (e.g., using application/vnd.myapi.v1+json).

11. How do you monitor and log Microservices?

Answer: Centralized logging tools like ELK (Elasticsearch, Logstash, Kibana) stack and monitoring tools like Prometheus and Grafana are used for monitoring and logging. Each microservice sends its logs and metrics to these centralized tools.

12. What is the role of a message broker in Microservices?

Answer: A message broker (e.g., Kafka, RabbitMQ) allows microservices to communicate asynchronously by sending and receiving messages or events, which helps in decoupling services and handling large volumes of data.

Advanced Level

1. What is the Saga Pattern in Microservices?

Answer: The Saga pattern manages long-running transactions across multiple services by breaking them into smaller, isolated transactions. If any step fails, compensating actions are triggered to ensure consistency across services.

2. What is Domain-Driven Design (DDD) in Microservices?

Answer: DDD is a methodology where the software is designed around the business domain. Microservices are aligned with business domains, and each service handles a specific bounded context.

3. How do you ensure Security in Microservices?

Answer: Security in microservices is achieved using techniques like OAuth2, JWT, Mutual TLS (mTLS), API Gateway for enforcing security policies, encryption of sensitive data, and rate limiting to prevent DDoS attacks.

4. How do you handle Distributed Tracing in Microservices?

Answer: Distributed tracing is used to track requests across multiple services. Tools like Jaeger, Zipkin, or OpenTelemetry help visualize the request flow and identify performance bottlenecks or failures in a microservice system.

5. What is Continuous Integration/Continuous Delivery (CI/CD) in the context of Microservices?

Answer: CI/CD ensures that microservices are continuously built, tested, and deployed. CI automatically integrates code changes and runs tests, while CD automates the deployment process to production environments.

6. What is the role of Kubernetes in Microservices?

Answer: Kubernetes is used for container orchestration. It automates the deployment, scaling, and management of microservices, ensuring that each service is running in the desired state and can be easily scaled or updated.

7. What is the CQRS pattern, and how does it apply to Microservices?

Answer: Command Query Responsibility Segregation (CQRS) separates the write (command) operations from the read (query) operations. This helps optimize performance and scalability in systems where read and write workloads are different.

8. What is the Strangler Fig pattern in Microservices?

Answer: The Strangler Fig pattern allows the gradual migration of a monolithic system to a microservice architecture by incrementally replacing parts of the monolith with microservices.

9. How do you handle Data Consistency in Microservices?

Answer: In microservices, eventual consistency is often preferred over strong consistency. Techniques like event sourcing, CQRS, and the Saga pattern are used to ensure data consistency across distributed services.

10. What is the difference between Orchestration and Choreography in Microservices?

Answer: Orchestration uses a central controller (e.g., a service) to coordinate and manage the flow of services, while choreography allows services to interact autonomously, with each service knowing how to handle its part of the process.

11. What are the best practices for testing Microservices?

Answer: Best practices include unit testing each service, using mocks or stubs for external services, integration testing to ensure services work together, and contract testing to ensure that API agreements are respected between services.

12. What is the API Composition pattern in Microservices?

Answer: API Composition is a pattern where a single API Gateway or a dedicated service aggregates responses from multiple microservices to return a unified result to the client.

Summary

• Basic: Covers the fundamentals of microservices, including architecture, benefits, and communication methods.

• Intermediate: Delves into patterns, common challenges, and service communication techniques.

• Advanced: Explores complex design patterns, integration, security, and advanced management techniques for microservices.

This expanded format should provide you with a deeper understanding of microservices, helping you tackle a broader range of interview questions across various levels.

Interview questions by backend developer ?

A,  
**Q1. What are microservices?**

Microservices are an architectural style that structures an application as a collection of small, autonomous services modeled around a business domain. Each service is self-contained and implements a single business capability.

**Q2. What are features of microservices?**

* **Decentralized Data Management**: Each microservice manages its own database.
* **Componentization**: Services are treated as independent components.
* **Business Capabilities**: Services are organized around business capabilities.
* **Autonomy**: Each service can be developed, deployed, and scaled independently.
* **Resilience**: Failure of one service does not affect the entire system.
* **Technology Diversity**: Different services can use different technologies.

**Q3. What are differences between Monolith Vs Microservices?**

* **Architecture**: Monolith is a single unified unit, while microservices are distributed and loosely coupled.
* **Scalability**: Monoliths scale as a whole, microservices scale independently.
* **Deployment**: Monoliths are deployed as a single unit, microservices are deployed independently.
* **Development**: Monoliths are developed in a single codebase, microservices have separate codebases.
* **Fault Isolation**: Failure in a monolith affects the whole system, in microservices, it affects only the failed service.

**Q4. What is Inter Process Communication?**

Inter Process Communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. In microservices, IPC can be achieved using protocols like HTTP/HTTPS, AMQP, or gRPC.

**Q5. How microservices communicate with each other?**

Microservices communicate using lightweight protocols such as HTTP/HTTPS for synchronous communication, and messaging queues like RabbitMQ or Kafka for asynchronous communication.

**Q6. How many microservices are there in your application?**

This answer depends on the specific application and its requirements. Typically, a microservices-based application can have anywhere from a few to hundreds of microservices.

**Q7. How to design and implement a microservice?**

* **Identify Business Capabilities**: Break down the application into business capabilities.
* **Define Service Boundaries**: Define clear boundaries for each service.
* **Choose Technology Stack**: Select appropriate technologies for each service.
* **Implement APIs**: Develop RESTful or gRPC APIs for communication.
* **Deploy Independently**: Ensure each service can be deployed independently.

**Q8. In which scenarios you would choose microservices over monolith systems?**

* **Scalability**: When you need to scale different parts of the application independently.
* **Agility**: When you need to deploy new features quickly and independently.
* **Resilience**: When you need fault isolation to ensure the failure of one service does not bring down the entire system.
* **Technology Diversity**: When you want to use different technologies for different parts of the application.

**Q9. How can you make sure a Microservices based application can handle more users as application becomes more popular?**

* **Load Balancing**: Distribute traffic across multiple instances of services.
* **Auto-Scaling**: Automatically scale services based on demand.
* **Caching**: Use caching to reduce load on services.
* **Database Sharding**: Distribute data across multiple databases.
* **Monitoring and Logging**: Continuously monitor and log performance metrics.

**Q10. How do you handle data consistency in a microservices application?**

* **Eventual Consistency**: Accept that data will be consistent over time.
* **Sagas**: Use a series of compensating transactions to maintain consistency.
* **Distributed Transactions**: Use protocols like 2PC (Two-Phase Commit) for distributed transactions.

**Q11. What is difference between Saga and 2PC?**

* **Saga**: A sequence of local transactions where each transaction updates the database and publishes an event or message.
* **2PC**: A protocol that ensures all participants in a distributed transaction agree to commit or rollback the transaction.

**Q12. How would you track the entire journey of a request across different services?**

* **Distributed Tracing**: Use tools like Zipkin or Jaeger to trace requests across services.
* **Correlation IDs**: Pass a unique identifier with each request to trace its path.

**Q13. How will you troubleshoot a failed API request that is spread across multiple services?**

* **Logs**: Check logs of all involved services.
* **Distributed Tracing**: Use tracing tools to follow the request path.
* **Monitoring Tools**: Use monitoring tools to identify where the failure occurred.

**Q14. What is JWT token and how does it look like?**

* **JWT (JSON Web Token)**: A compact, URL-safe token used for securely transmitting information between parties.
* **Structure**: Consists of three parts - Header, Payload, and Signature, separated by dots (e.g., header.payload.signature).

**Q15. What are use cases of JWT token?**

* **Authentication**: Verify user identity.
* **Authorization**: Grant access to resources.
* **Information Exchange**: Securely transmit information between parties.

**Q16. Why Basic Authentication is not suitable in Microservices Context?**

* **Security**: Basic Authentication sends credentials in every request, which can be insecure.
* **Scalability**: It does not scale well with microservices architecture.
* **Token Management**: JWT or OAuth2 tokens are more secure and manageable.

**Q17. Why should we use OAuth2 for microservices?**

* **Security**: Provides secure access to resources.
* **Scalability**: Supports token-based authentication, which scales well.
* **Flexibility**: Allows fine-grained access control.

**Q18. What is Eureka Server?**

* **Eureka Server**: A service registry for resilient load balancing and failover of middle-tier servers and clients.

**Q19. What rest clients have you used?**

* **RestTemplate**: A synchronous client for making HTTP requests.
* **WebClient**: A non-blocking, reactive client for making HTTP requests.

**Q20. Implementation of rest clients?**

* **RestTemplate Example**:

RestTemplate restTemplate = new RestTemplate();

String result = restTemplate.getForObject("http://example.com/api", String.class);

* **WebClient Example**:

WebClient webClient = WebClient.create();

String result = webClient.get()

.uri("http://example.com/api")

.retrieve()

.bodyToMono(String.class)

.block();

**Q21. How to slowly move users from older version of application to newer version?**

* **Canary Releases**: Gradually roll out the new version to a small subset of users.
* **Blue-Green Deployments**: Run both versions in parallel and switch traffic gradually.

**Q22. How to achieve zero-downtime during the deployments?**

* **Blue-Green Deployments**: Deploy the new version alongside the old one and switch traffic.
* **Rolling Updates**: Gradually update instances of the application.

**Q23. A microservice in your system must perform a time-consuming task, How do you ensure that other services don’t experience delays while waiting for the task to complete?**

* **Asynchronous Processing**: Use message queues to handle time-consuming tasks asynchronously.
* **Background Jobs**: Offload the task to background workers.

**Q24. How does OAuth2 Works?**

* **Authorization Grant**: The client obtains an authorization grant from the resource owner.
* **Access Token**: The client exchanges the authorization grant for an access token.
* **Resource Access**: The client uses the access token to access protected resources.

**Q25. How will you monitor fleet of microservices in production?**

* **Monitoring Tools**: Use tools like Prometheus, Grafana, or ELK stack.
* **Health Checks**: Implement health checks for each service.

**Q26. What is difference between Orchestration and Choreography in microservices context?**

* **Orchestration**: A central controller coordinates the interactions between services.
* **Choreography**: Services interact with each other through events without a central controller.

**Q27. What shall be preferred communication style in microservices: synchronous or asynchronous?**

* **Asynchronous**: Preferred for decoupling services and improving resilience.
* **Synchronous**: Used when immediate response is required.

**Q28. How big a single microservice should be?**

* **Single Responsibility**: A microservice should handle a single business capability.
* **Size**: Small enough to be developed and deployed independently, but large enough to avoid excessive communication overhead.

**Q29. How to partition a large application into microservices architecture?**

* **Domain-Driven Design**: Identify bounded contexts and business capabilities.
* **Decompose**: Break down the application into smaller, independent services.

**Q30. What is Bounded Context?**

* **Bounded Context**: A boundary within which a particular domain model is defined and applicable.

**Q31. How can we perform end-to-end testing for a system with hundreds of microservices? Is it necessary to deploy all services before test execution?**

* **Service Virtualization**: Simulate the behavior of dependent services.
* **Selective Deployment**: Deploy only the necessary services for the test.

**Q32. How will you write an end-to-end test for microservices architecture?**

* **Test Scenarios**: Define end-to-end test scenarios.
* **Test Data**: Prepare test data for each service.
* **Automation**: Use tools like Selenium, JUnit, or Cucumber for automation.

**Q33. What is Contract-Driven Tests?**

* **Contract-Driven Tests**: Tests that ensure the interactions between services adhere to predefined contracts.

**Sure, let's continue with the remaining questions:**

**Q34. How will you implement service discovery in microservices architecture?**

* **Service Registry: Use a service registry like Eureka, Consul, or Zookeeper where services register themselves and discover other services.**
* **Client-Side Discovery: Clients query the service registry to find the instances of a service.**
* **Server-Side Discovery: A load balancer queries the service registry and routes requests to the appropriate service instances.**

**Q35. How to achieve zero downtime deployment (blue/green) when there is a database change?**

* **Database Versioning: Use database versioning tools like Flyway or Liquibase to manage schema changes.**
* **Backward Compatibility: Ensure the new version of the application is backward compatible with the old database schema.**
* **Data Migration: Gradually migrate data to the new schema while both versions are running.**

**Q36. How do you divide your monolithic application to microservice? What will be your approach?**

* **Identify Boundaries: Identify logical boundaries within the monolith based on business capabilities.**
* **Decompose: Break down the monolith into smaller, independent services.**
* **Incremental Migration: Gradually migrate functionality from the monolith to microservices.**
* **Refactor: Refactor the monolith to remove dependencies on the extracted services.**

**Q37. How do you incrementally migrate from monolithic to microservices? Can you think of any pattern that can be applied?**

* **Strangler Fig Pattern: Gradually replace parts of the monolith with microservices until the monolith is completely replaced.**
* **API Gateway: Use an API Gateway to route requests to either the monolith or the new microservices.**
* **Feature Toggles: Use feature toggles to switch between the monolith and microservices.**

**Q38. How do you handle the security of microservice application?**

* **Authentication and Authorization: Use OAuth2 or JWT for secure authentication and authorization.**
* **API Gateway: Implement security at the API Gateway level.**
* **Encryption: Encrypt sensitive data in transit and at rest.**
* **Security Audits: Regularly perform security audits and vulnerability assessments.**

**Q39. How does authorization work in microservice application, Do you know the end-to-end flow from request hit to browser and getting response?**

* **User Authentication: The user authenticates and receives an access token (e.g., JWT).**
* **API Gateway: The API Gateway validates the token and forwards the request to the appropriate microservice.**
* **Service Authorization: Each microservice validates the token and checks the user's permissions.**
* **Response: The microservice processes the request and sends the response back through the API Gateway to the user.**

**Q40. How to make microservice API backward compatible?**

* **Versioning: Use versioning in your API endpoints (e.g., /v1/resource).**
* **Deprecation: Gradually deprecate old versions while supporting them for a transition period.**
* **Compatibility: Ensure new changes do not break existing clients.**

**Q41. How do you make sure your microservices interact with each other?**

* **API Contracts: Define clear API contracts and use tools like Swagger/OpenAPI for documentation.**
* **Service Discovery: Use a service registry for dynamic discovery of services.**
* **Message Brokers: Use message brokers like RabbitMQ or Kafka for asynchronous communication.**

**Q42. What can be done if there are communication failures between microservices?**

* **Retries: Implement retry logic with exponential backoff.**
* **Circuit Breaker: Use the circuit breaker pattern to prevent cascading failures.**
* **Fallbacks: Provide fallback mechanisms to handle failures gracefully.**

**Q43. If you had to scale a Spring Boot application what strategies you would use?**

* **Horizontal Scaling: Deploy multiple instances of the application.**
* **Load Balancing: Use a load balancer to distribute traffic across instances.**
* **Auto-Scaling: Configure auto-scaling based on metrics like CPU usage or request rate.**

**Q44. Microservice is running fine for a year, but there is performance difference from 13 ms to 30ms. How do you find the issue and fix it?**

* **Monitoring: Use monitoring tools to identify performance bottlenecks.**
* **Profiling: Profile the application to find slow methods or database queries.**
* **Logs: Check logs for errors or warnings.**
* **Optimization: Optimize the identified bottlenecks, such as improving database queries or refactoring code.**

**Q45. Why do we need to use API Gateway pattern?**

* **Centralized Routing: Route requests to the appropriate microservices.**
* **Security: Implement authentication, authorization, and rate limiting.**
* **Load Balancing: Distribute traffic across multiple instances.**
* **Aggregation: Aggregate responses from multiple services.**

**Q46. What is circuit breaker design pattern?**

* **Circuit Breaker: A pattern that prevents a service from repeatedly trying to execute an operation that is likely to fail, allowing it to fail fast and recover gracefully.**

**Q47. What do you require to implement circuit breaker design pattern?**

* **Library: Use a library like Hystrix or Resilience4j.**
* **Configuration: Configure thresholds for failure rates and timeouts.**
* **Fallbacks: Define fallback methods to handle failures.**

**Q48. Have you worked with Docker and Kubernetes?**

* **Docker: A platform for developing, shipping, and running applications in containers.**
* **Kubernetes: An orchestration platform for managing containerized applications.**

**Q49. Do you know what is Docker? Why we need it?**

* **Docker: A tool that allows you to package an application and its dependencies into a container.**
* **Need: Ensures consistency across different environments, simplifies deployment, and improves scalability.**

**Q50. Explain Docker commands you used most?**

* **docker build: Build an image from a Dockerfile.**
* **docker run: Run a container from an image.**
* **docker ps: List running containers.**
* **docker stop: Stop a running container.**
* **docker rm: Remove a container.**

**Q51. What is Kubernetes?**

* **Kubernetes: An open-source platform for automating the deployment, scaling, and management of containerized applications.**