Unit-5

Ques1) what is microservice

Ans:

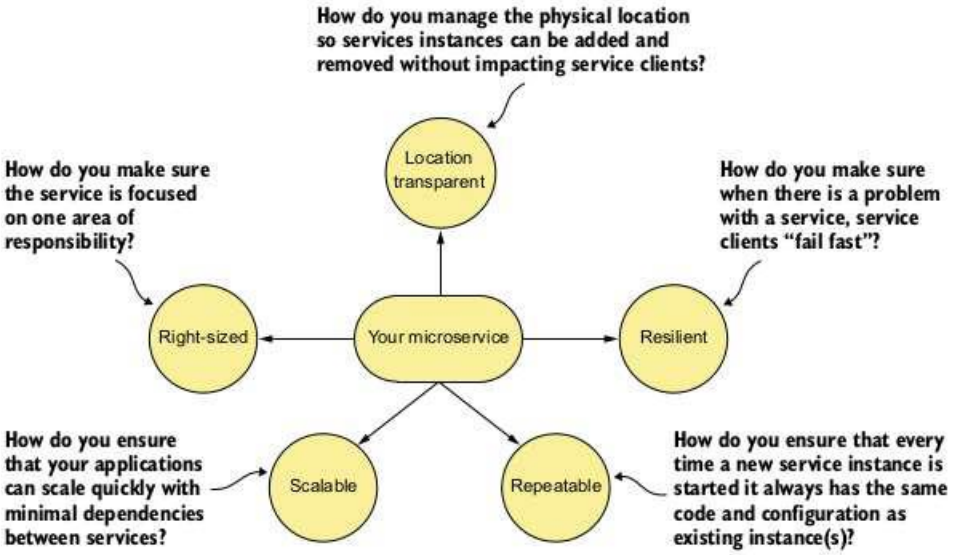
•Microservices are a modern approach to software whereby application code is delivered in small, manageable pieces, independent of others.

•Microservices is an architectural approach to developing software applications as a collection of small, independently deployable services. Each microservice is focused on a specific business capability and can be developed, deployed, and scaled independently.

•Overall, microservices architecture aims to improve the flexibility, scalability, and resilience of software applications, making it easier to develop and maintain complex systems.

Ques2) Key characteristics/features of microservices include:

Ans)



Microservices have several key features that distinguish them from monolithic architectures. Some of the main features include:

•Decomposition: Breaking down an application into smaller, manageable services, each responsible for a specific aspect of the application's functionality.

•Independence: Each microservice can be developed, deployed, and scaled independently of other services, allowing for more flexibility and agility in development and operations.

•Resilience: Microservices are designed to be resilient to failures. If one service fails, it does not necessarily impact the entire application.

•Scalability: Services can be scaled independently based on demand, allowing for better resource utilization and performance optimization.

•Technology diversity: Each microservice can be implemented using different technologies, languages, and frameworks, based on what is most suitable for its specific functionality.

•Communication: Microservices communicate with each other using lightweight protocols such as HTTP/REST or messaging queues, enabling them to work together to fulfill complex business requirements.

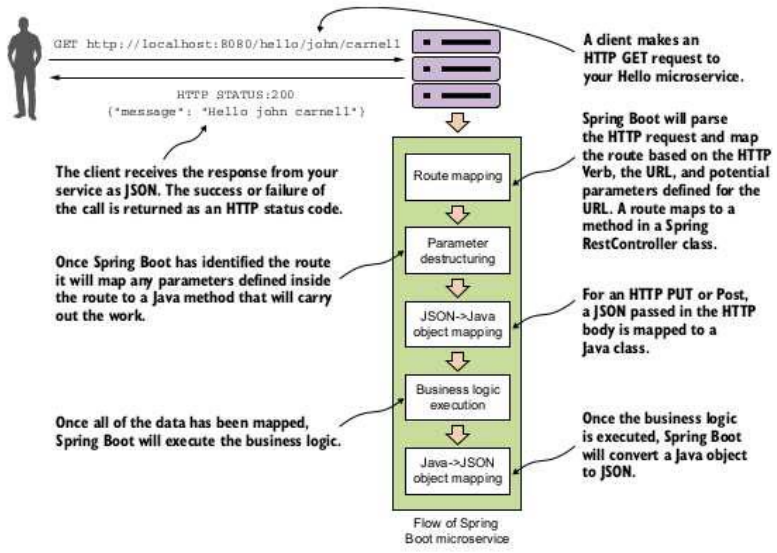
Ques3) What is Spring and why is it relevant to microservices?

Ans

- **Ease of Development**: Spring makes it easier to build complex applications by providing a clear and consistent way to handle tasks like connecting to databases, creating web services, and handling messages between components.
- **Dependency Injection**: Spring helps keep different parts of your application separate and easy to change by managing how they connect to each other. This is important for building microservices that can be updated or scaled independently.
- **Linkages:** In traditional programming, the places where different parts of code connect (like calling a class's constructor) are fixed once the code is written and compiled. Spring's approach allows these connections to be more flexible and changeable.
- **Microservices** Architecture Support: Spring Boot, a part of Spring, makes it simpler to set up and run microservices. It takes care of many technical details, so developers can focus more on the specific features of their microservices.
- **Integration with Cloud Services**: Spring Cloud provides tools and features for building microservices that can work well in cloud environments. It includes things like helping services find each other, managing their settings, and handling errors in a scalable way.

Ques4) Building a microservice with Spring Boot?

Ans)



- **Setup your development environment**: Install Java JDK and an IDE like IntelliJ IDEA or Eclipse.
- **Create a new Spring Boot project**: You can use Spring Initializr (https://start.spring.io/) to generate a new project. Choose the dependencies you need, such as Spring Web for building RESTful APIs.
- **Write your microservice code**: Define your service logic, endpoints, and any necessary configurations. For example, you can create a REST controller to handle HTTP requests.
- **Run and test your microservice:** Start your Spring Boot application and test it using tools like Postman or curl. Verify that your service is behaving as expected.
- **Deploy your microservice:** You can package your application as a JAR file and deploy it to a server or cloud platform of your choice. Spring Boot's embedded Tomcat server makes deployment easy.
- **Monitor and manage your microservice**: Use Spring Boot Actuator to monitor and manage your microservice. Actuator provides endpoints for metrics, health checks, and more.
- **Scale your microservice**: As your application grows, you can scale your microservice horizontally by deploying multiple instances behind a load balancer.

- **Update and maintain your microservice**: Spring Boot's auto-configuration and dependency management features make it easy to update and maintain your microservice over time.

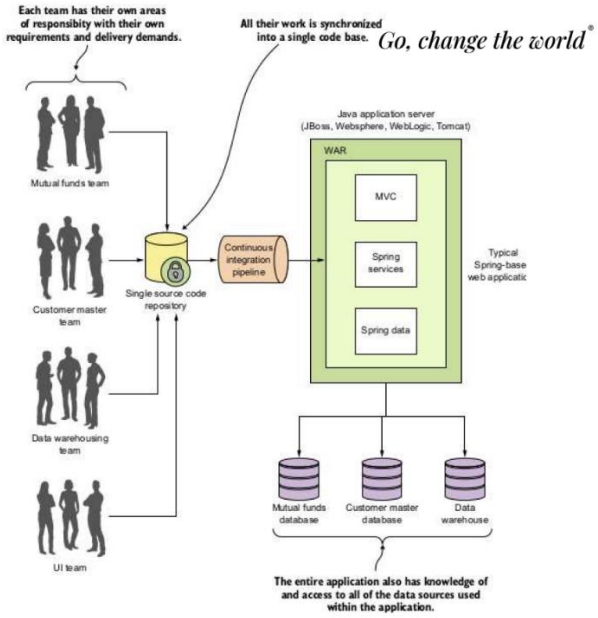
Ques5) Difference between monolithic approach and microservice approach?

or

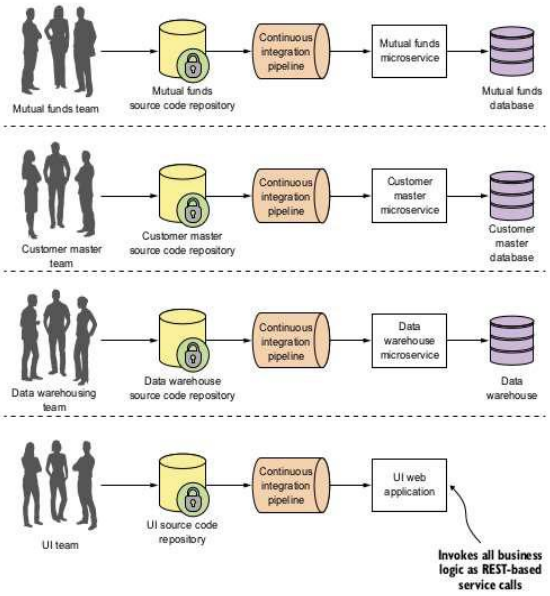
Why there is a need to change the approach we build the application to microservices?

Ans:

Monolithic approach



Microservices Approach



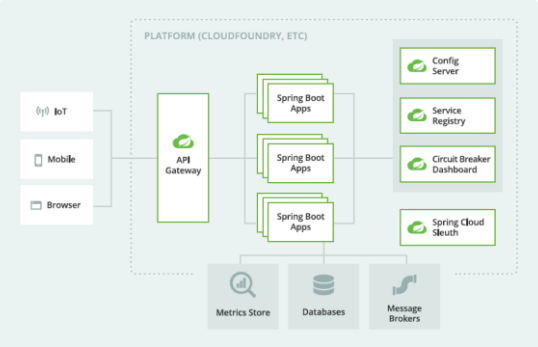
1. **competitive pressures:**
    a. Complexity has gone way up
    b. Customers want faster delivery
    c. Performance and scalability
    d. Customers expect their applications to be available

2. **Break our applications into small services - build systems**
    a. Flexible—Decoupled services, The smaller the unit of code
    b. Resilient
    c. Scalable

3. **Small, Simple, and Decoupled Services = Scalable, Resilient, and Flexible Applications**

Ques6) Architecture of microservice?

Ans



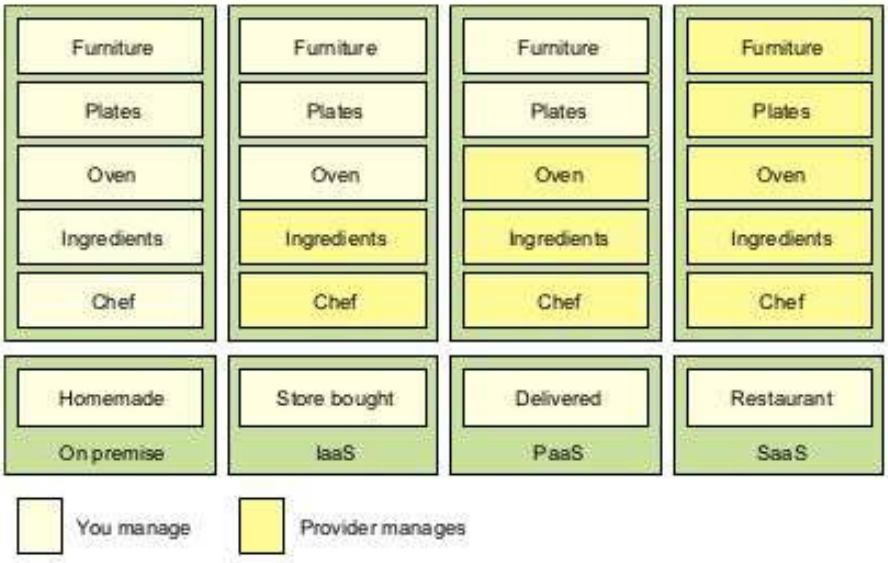
- **Service**: A microservice is a small, independently deployable service that focuses on a specific business capability. Each service is developed, deployed, and scaled independently.
- **API Gateway:** The API Gateway is the entry point for clients to access the microservices. It routes requests to the appropriate services and can handle tasks such as authentication, rate limiting, and caching.
- **Database**: Each microservice typically has its own database or data storage solution. This allows for better isolation and scalability, but also introduces challenges in data consistency and synchronization.
- **Config Server:** A Config Server in a microservices architecture centralizes configuration management, allowing services to retrieve their configuration from a single source, which can be dynamically updated without requiring service restarts.
- **Service Registry**: A Service Registry, like Eureka, is used for service discovery, enabling services to locate and communicate with each other without hard-coded dependencies, promoting dynamic and flexible architectures.
- **Circuit Breaker Dashboard**: A Circuit Breaker Dashboard, such as Hystrix Dashboard, provides a visual representation of circuit breaker states and metrics, aiding in monitoring and understanding the resilience of microservices.
- **Spring Cloud Sleuth**: Spring Cloud Sleuth is a distributed tracing solution that provides visibility into the flow of requests across

microservices, helping to diagnose and troubleshoot issues in complex distributed systems.

- **Metrics Stores**: Metrics stores, such as Prometheus and Graphite, are used to store and query metrics data collected from various components in a microservices architecture. They enable monitoring and performance analysis of the system.
- **Message Brokers**: Message brokers, like Apache Kafka and RabbitMQ, facilitate communication between microservices by providing a reliable and scalable messaging infrastructure. They help in decoupling services and enable asynchronous communication patterns.

Ques7) What exactly is the cloud?

Ans



The cloud refers to a network of servers that are hosted on the internet and used to store, manage, and process data, rather than on a local server or personal computer. Cloud computing provides access to a variety of services, including servers, storage, databases, networking, software, and analytics, over the internet.

There are different types of cloud computing models:

- **Infrastructure as a Service (IaaS):** Provides virtualized computing resources over the internet. Users can rent virtual machines, storage, and other resources from a cloud provider.
- **Platform as a Service (PaaS):** Provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.
- **Software as a Service (SaaS):** Delivers software applications over the internet on a subscription basis. Users can access the software through a web browser without needing to install or maintain it.

Cloud computing offers several advantages, including scalability, cost-effectiveness, flexibility, and the ability to access resources from anywhere with an internet connection. It has become a popular choice for individuals and businesses looking to improve efficiency and reduce costs.

Ques8) Why the cloud and microservices?

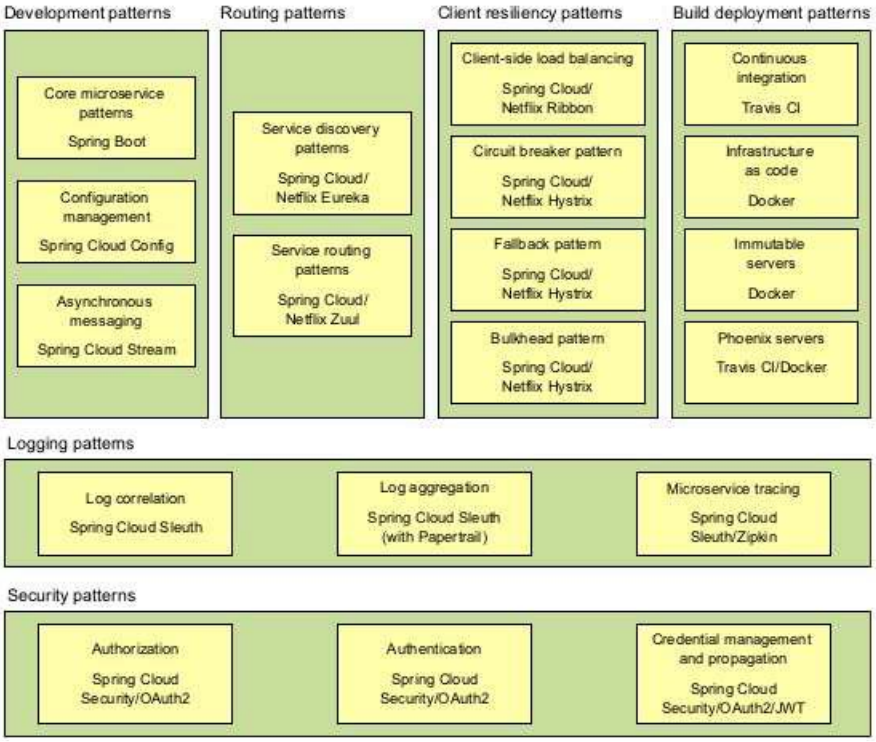
Ans

1. As a developer writing a microservice, whether your service is going to be deployed to one of the following:
    a. Physical server
    b. Virtual machine images
    c. Virtual container
2. Cloud-based microservices centers around the concept of elasticity.
3. This is a common deployment topology used for microservices:
    a. **Simplified infrastructure management**—IaaS cloud providers give you the ability to have the most control over your services. New services can be started and stopped with simple API calls. With an IaaS cloud solution, you only pay for the infrastructure that you use.
    b. **Massive horizontal scalability**—IaaS cloud providers allow you to quickly and succinctly start one or more instances of a service. This capability means you can quickly scale services and route around misbehaving or failing servers.
    c. **High redundancy through geographic distribution**—By necessity, IaaS providers have multiple data centers. By deploying your microservices using an IaaS cloud provider, you can gain a higher level of redundancy beyond using clusters in a data center.

4. The cloud and microservices are often used together because they complement each other and offer several advantages when used in conjunction:
    a. Scalability:
    b. Flexibility:
    c. Resilience:
    d. Cost-effectiveness:
    e. Speed of development:

Ques9) How to solve the Microservices problem USING VARIOUS DESIGN PATTERNS

Ans



Core Development Patterns:

- Microservices: Decompose your application into small, loosely coupled services that focus on specific business capabilities.
- Service Registry: Use a service registry like Eureka to allow services to dynamically discover and communicate with each other.

Routing Patterns:

- API Gateway: Implement an API Gateway to route client requests to the appropriate microservice and handle cross-cutting concerns like authentication and rate limiting.
- Service Mesh: Use a service mesh like Istio to manage service-to-service communication, including routing, load balancing, and security.

Client Resiliency Patterns:

- Circuit Breaker: Implement a circuit breaker pattern using tools like Hystrix to prevent cascading failures and provide fallback mechanisms.
- Retry: Use a retry pattern to automatically retry failed requests with exponential backoff to improve reliability.

Security Patterns:

- OAuth2: Use OAuth2 for authentication and authorization between services and clients.
- JWT: Use JSON Web Tokens (JWT) for secure information exchange between services.
- TLS: Use Transport Layer Security (TLS) for secure communication between services.

Logging and Tracing Patterns:

- Centralized Logging: Use a centralized logging solution like ELK stack (Elasticsearch, Logstash, Kibana) to aggregate and analyze logs from all services.
- Distributed Tracing: Use a distributed tracing tool like Zipkin or Jaeger to trace requests as they flow through your microservices architecture.

Build and Deployment Patterns:

- Containerization: Use Docker for containerization to package your microservices and their dependencies into lightweight, portable containers.
- Orchestration: Use Kubernetes or Docker Swarm for orchestration to manage and scale your containers in a clustered environment.
- Continuous Integration/Continuous Deployment (CI/CD): Implement CI/CD pipelines to automate the build, test, and deployment of your microservices.

Ques) How does Spring Cloud contribute to building microservices with Spring Boot?

Answer:

Spring Cloud provides a set of tools and libraries that complement Spring Bootfor building microservices. It includes features like service discovery, distributedconfiguration management, circuit breakers, and more to simplify the development ofmicroservices.

- **Service Discovery**: Spring Cloud provides a service discovery mechanism that allows microservices to register themselves with a service registry (e.g., Eureka) and discover other services. This enables dynamic service registration and discovery, crucial for building resilient and scalable microservices architectures.
- **Load Balancing**: Spring Cloud integrates with client-side load balancing (e.g., Ribbon) to distribute the load across multiple instances of a service. This helps in improving the performance and scalability of microservices by efficiently utilizing resources.
- **Configuration Management**: Spring Cloud Config provides a centralized configuration management solution for microservices. It allows you to manage configurations across different environments and update them dynamically without requiring a redeployment of services.
- **Circuit Breaker**: Spring Cloud integrates with Netflix Hystrix to implement the circuit breaker pattern, which helps in preventing cascading failures and provides fallback mechanisms in case of service failures.
- **API Gateway**: Spring Cloud Gateway provides a way to build a gateway service that routes requests to the appropriate microservice based on the request path. It also allows you to apply cross-cutting concerns such as security, rate limiting, and logging at the gateway level.
- **Distributed Tracing**: Spring Cloud Sleuth integrates with Zipkin or Jaeger to provide distributed tracing capabilities for microservices. It helps in monitoring and troubleshooting the flow of requests across different services.
- **Monitoring and Metrics**: Spring Cloud integrates with tools like Prometheus and Grafana to provide monitoring and metrics for microservices. It allows you to collect and visualize metrics such as request latency, error rates, and throughput.

x----------x----------x----------

UNIT – V

1.

What is a microservice?

Answer:

A microservice is a small, independent, and self-contained service that performsa specific business function within a larger software application. Microservices architectureinvolves breaking down an application into a collection of loosely coupled, independentlydeployable services.

2.

What is Spring, and why is it relevant to microservices?

Answer:

Spring is a framework for building enterprise Java applications. It is relevant tomicroservices because it provides a comprehensive set of tools and libraries for developingscalable, maintainable, and loosely coupled components. Spring Boot, a part of the Springframework, simplifies the process of building microservices by offering convention overconfiguration and automatic setup.

3.

Why change the way we build applications with microservices?

Answer:

Traditional monolithic applications can become complex and challenging tomaintain as they grow. Microservices offer advantages such as improved scalability,flexibility, and ease of deployment. By breaking down applications into smaller services,development teams can work more independently, and applications become more modular.

4.

What exactly is the cloud?

Answer:

The cloud refers to a network of remote servers hosted on the internet that store,manage, and process data rather than on a local server or a personal computer. Cloudcomputing provides on-demand access to computing resources, including storage,processing power, and services, often with a pay-as-you-go model.

5.

Why the cloud and microservices?

Answer:

The cloud complements microservices by providing a scalable and flexibleinfrastructure. Microservices can be easily deployed and scaled horizontally in the cloudenvironment. Cloud services also offer various tools and services that enhance thedevelopment, deployment, and maintenance of microservices.

6.

What are the core microservice development patterns?

Answer:

Core microservice development patterns include:

Service Independence:

Each microservice is independent and has its owndatabase.

Decentralized Data Management:

Each microservice manages its own data.

Automated Deployment:

Microservices are deployed independently.

7.

Explain microservice routing patterns.

Answer:

Microservice routing patterns involve directing requests from clients to theappropriate microservices. Examples include API Gateway pattern, where a single entrypoint handles client requests and routes them to the corresponding microservices based onthe request.

8.

Describe microservice client resiliency patterns.

Answer:

Microservice client resiliency patterns include strategies to handle failuresgracefully. Examples include circuit breaker pattern, where a microservice can detect

failures and prevent cascading failures in the system by short-circuiting requests.

9.

What are microservice security patterns?

Answer:

Microservice security patterns address security concerns in a distributed system.Examples include token-based authentication, where each microservice validatesauthentication tokens, and role-based access control to control access based on user roles.

10.

Explain microservice logging and tracing patterns.

Answer:

Microservice logging and tracing patterns involve capturing and analyzing logsand traces to monitor and troubleshoot microservices. Examples include centralizedlogging and distributed tracing to gain insights into the behavior of microservices.

11.

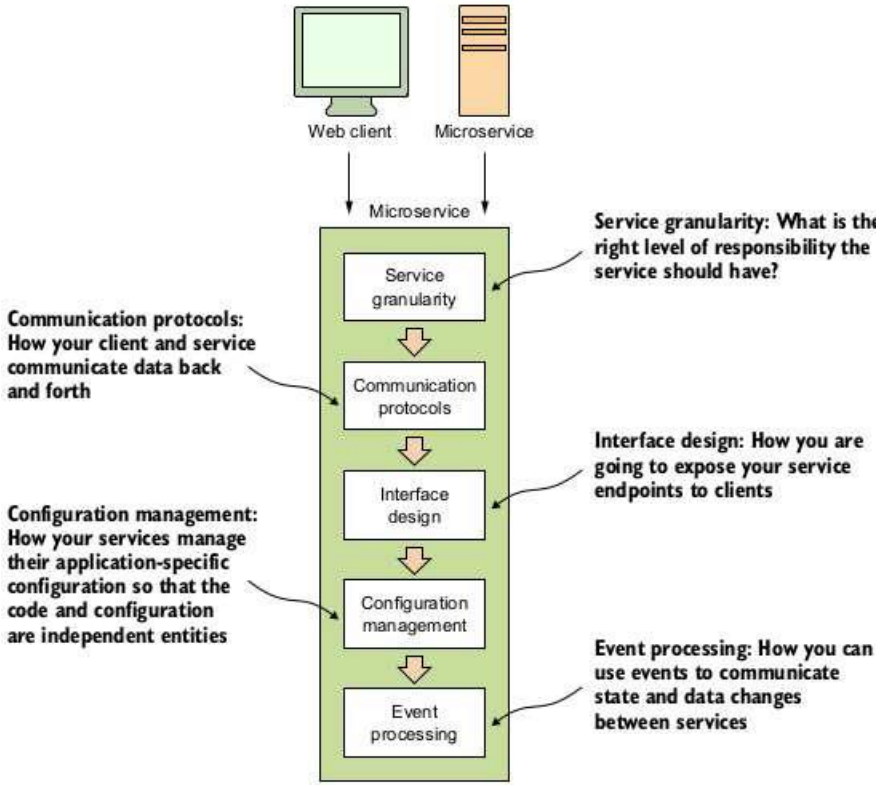
What are microservice build/deployment patterns?

Answer:

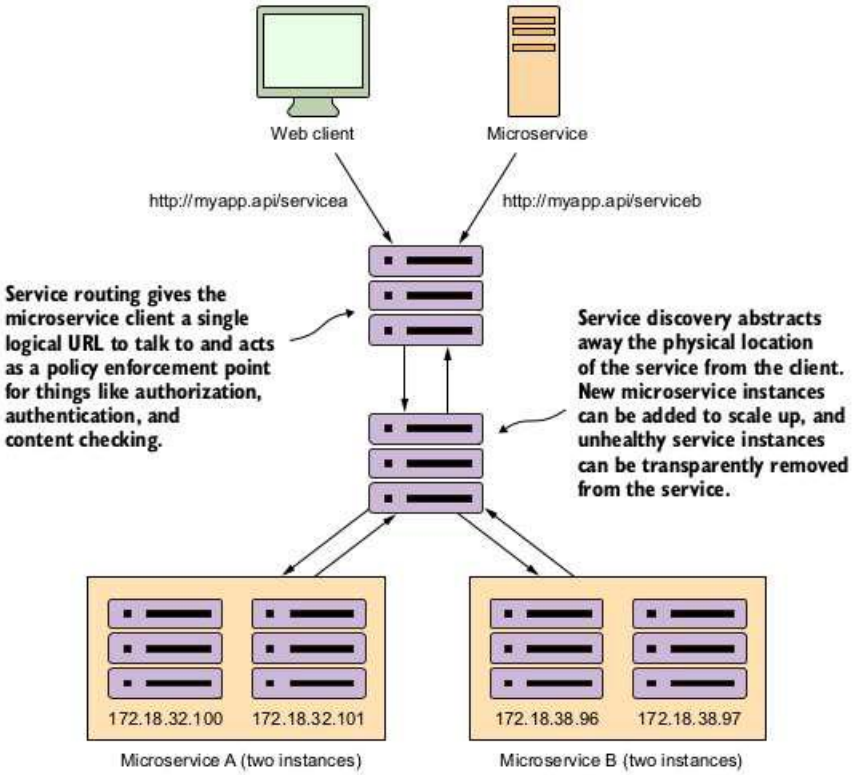
Microservice build/deployment patterns include techniques for building anddeploying microservices. Examples include containerization with Docker and usingcontainer orchestration tools like Kubernetes for managing and scaling containers.

x----------x----------x----------

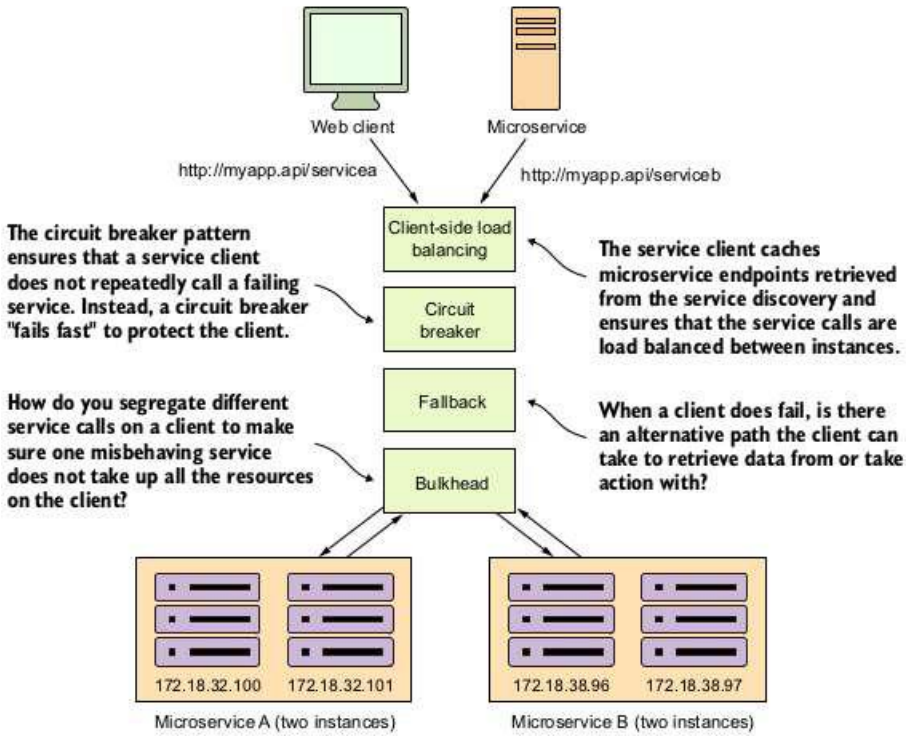
1. **Core Development Patterns**:



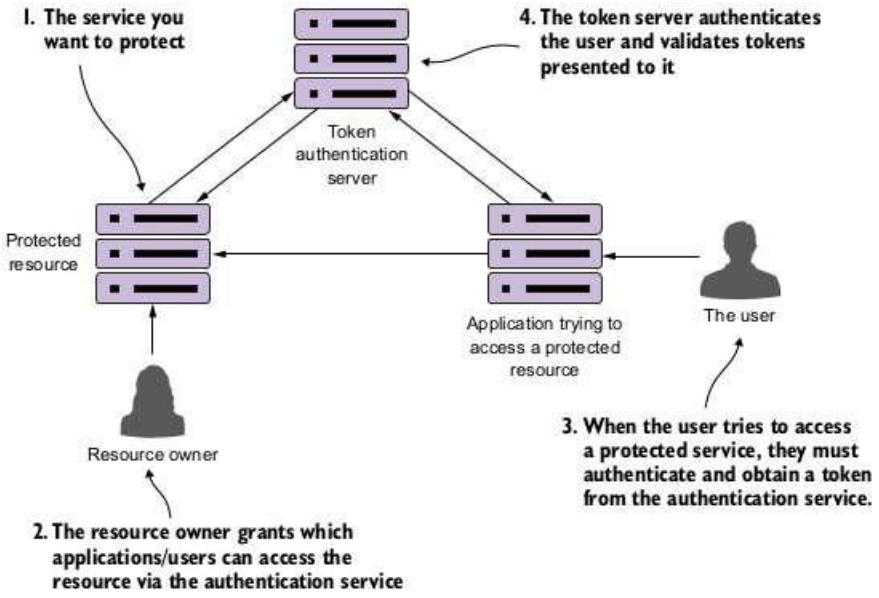
2. **Routing Patterns**



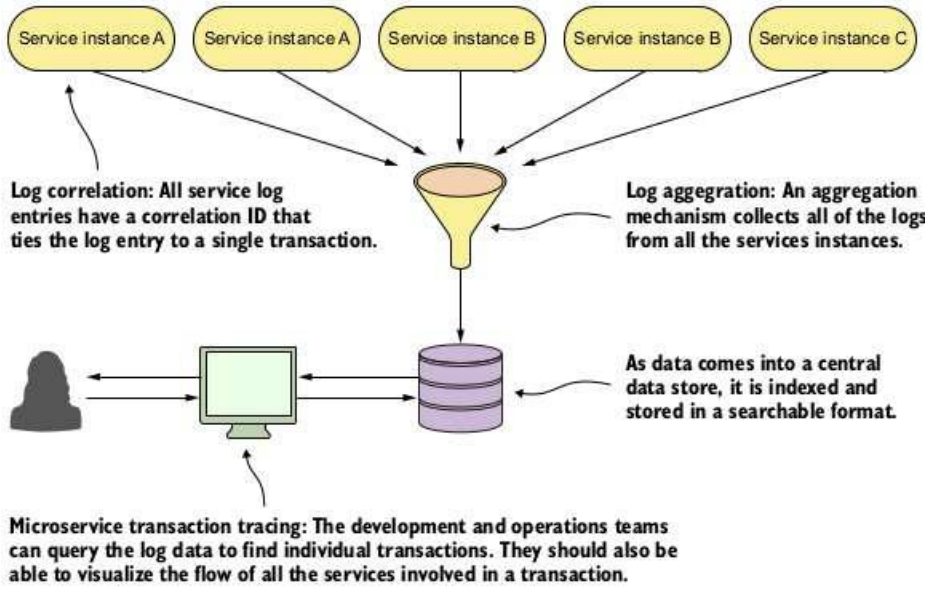
3. **Client Resiliency Patterns**



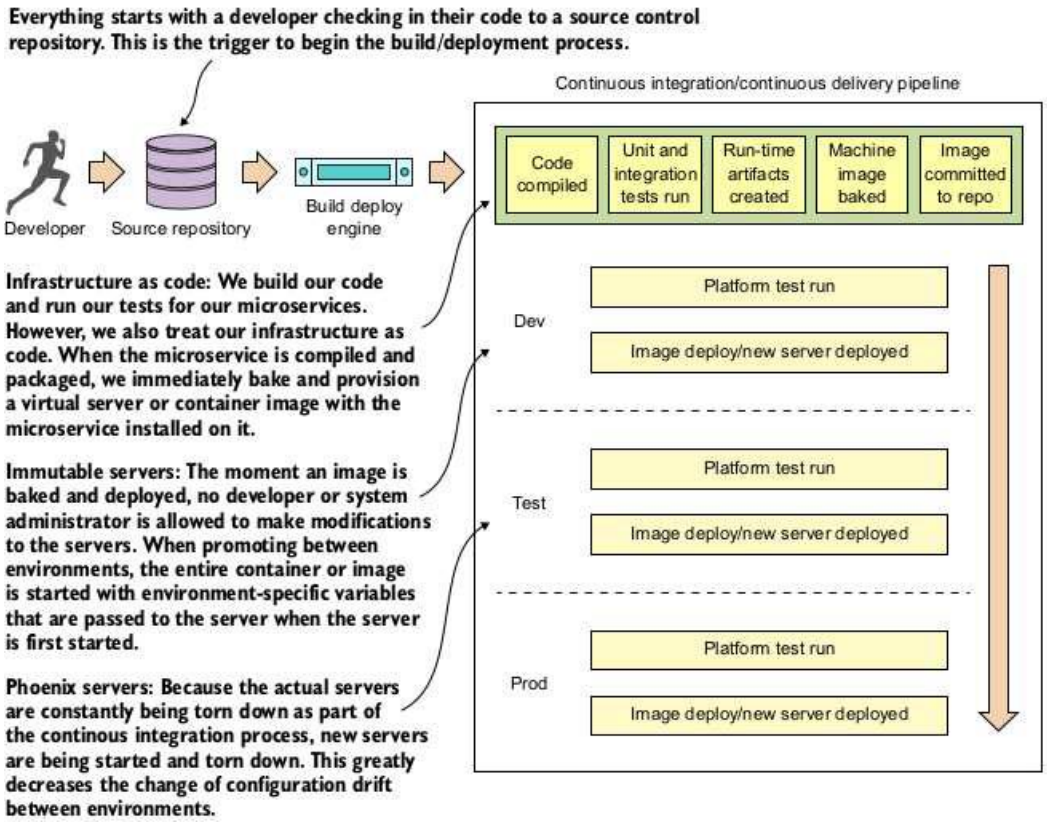
4. **Security Patterns**



5. **Logging and Tracing Patterns**:



6. **Build and Deployment Patterns**:


Everything starts with a developer checking in their code to a source control repository. This is the trigger to begin the build/deployment process.
Continuous integration/continuous delivery pipeline
Developer
Source repository
Build deploy engine
Code compiled
Unit and integration tests run
Run-time artifacts created
Machine image baked
Image committed to repo
Dev
Platform test run
Image deploy/new server deployed
Test
Platform test run
Image deploy/new server deployed
Prod
Platform test run
Image deploy/new server deployed
Infrastructure as code: We build our code and run our tests for our microservices. However, we also treat our infrastructure as code. When the microservice is compiled and packaged, we immediately bake and provision a virtual server or container image with the microservice installed on it.
Immutable servers: The moment an image is baked and deployed, no developer or system administrator is allowed to make modifications to the servers. When promoting between environments, the entire container or image is started with environment-specific variables that are passed to the server when the server is first started.
Phoenix servers: Because the actual servers are constantly being torn down as part of the continous integration process, new servers are being started and torn down. This greatly decreases the change of configuration drift between environments.