1. **Advantages of microservices** 
   * Independent Development
   * Fault tolerance
   * Mixed Technology Stack
   * Scaling up and down
   * Ensures security of each service
   * Multiple services are parallelly developed and deployed
2. **disadvantages of microservices** 
   * Increases troubleshooting challenges
   * Increases delay due to remote calls
   * Increased efforts for configuration and other operations
   * Difficult to maintain transaction safety
   * Tough to track data across various boundaries
   * Difficult to code between services
   * Cyclic dependency
3. **Spring cloud:**

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state).

1. **Embedded Containers :**

Tomcat, Jetty, undertow

1. **Eureka Server :**

Eureka Server is an application that holds the information about all client-service applications. Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address.

eureka.client.registerWithEureka = false

eureka.client.fetchRegistry = false

server.port = 8761

1. **Why Would You Need Reports & Dashboards In Microservices?**
   * find out which microservices expose what resources.
   * find out the services which are impacted whenever changes in a component occur.
   * provide an easy point which can be accessed whenever documentation is required.
   * Versions of the components which are deployed.
   * To obtain a sense of maturity and compliance from the components.
2. **Domain-driven design (DDD) :**

is the concept that the structure and language of software code (class names, class methods, class variables) should match the [business domain](https://en.wikipedia.org/wiki/Business_domain). For example, if a software processes loan applications, it might have classes such as LoanApplication and Customer, and methods such as AcceptOffer and Withdraw.

DDD connects the [implementation](https://en.wikipedia.org/wiki/Implementation) to an evolving model.

Domain-driven design is predicated on the following goals:

* placing the project's primary focus on the core [domain](https://en.wikipedia.org/wiki/Domain_(software_engineering)) and domain logic;
* basing complex designs on a model of the domain;
* initiating a creative collaboration between technical and [domain experts](https://en.wikipedia.org/wiki/Domain_expert) to iteratively refine a conceptual model that addresses particular domain problems.

1. **What are the features and characteristics of Microservices?**
   * Decoupling
   * independent components that can be easily replaced and upgraded
   * simple and focus on a single capability
   * work independently
   * Continuous Delivery
   * Responsibility
   * Decentralized Governance
   * Agility
2. **What are the best practices to design Microservices?**
   * Separate data store from each microservice
   * Keep code at similar level of maturity
   * Separate build for each microservice
   * Deploy into container
   * Treat server as stateless

### **What are the challenges you face while working Microservice Architectures?**

* + Need to Build, Deploy and, Monitor each service independently.
  + It requires great understanding of all the service in case of any failure
  + Need to maintain the configurations for each service like harbor.
  + Testing and Debugging
  + Require heavy investment
  + Heavy Infrastructure Setup
  + Excessive Planning for managing operations overhead
  + Communication between different microservices in the application.
  + Application maintenance
  + Configuration Management
  + Deployment Challenges

### **What do you know about Spring Boot?**

* + It’s a knows fact that spring has become more and more complex as new functionalities have been added. If you have to start a new spring project, then you have to add build path or add maven dependencies, configure application server, add spring configuration. So everything has to be done from scratch.
  + **Spring Boot** is the solution to this problem. Using spring boot you can avoid all the boilerplate code and configurations. So basically consider yourself as if you’re baking a cake spring is like the ingredients that are required to make the cake and spring boot is the complete cake in your hand.

1. **What is an actuator in Spring boot?**

Spring Boot actuator provides restful web services to access the current state of running an application in the production environment. With the help of actuator, you can check various metrics and monitor your application.

* + Health
  + Beans
  + Mappings
  + Shutdown

### **What problems are solved by Spring Cloud?**

* **The complexity associated with distributed systems** – This includes network issues, Latency overhead, Bandwidth issues, security issues.
* **Ability to handle Service Discovery** – Service discovery allows processes and services in a cluster to find each other and communicate.
* **Solved redundancy issues** – Redundancy issues often occur in distributed systems.
* **Load balancing** – Improves the distribution of workloads across multiple computing resources, such as a computer cluster, network links, central processing units.
* **Reduces performance issues** – Reduces performance issues due to various operational overheads.

1. **What are the 12 factors in microservice**
   * **Codebase** : project should be in Git and able to create multiple branches like for production, development, deployment, testing, etc.
   * **Dependencies** : project should have the mechanism to add the dependencies like MEVEN or gradle
   * **Config** : config should be separated from the business logic like Spring config server
   * **Backing services** : project should have a mechanism to attach and detach service to add new instance or to remove instance like DB instance if it is getting overloaded (on off functionality)
   * **Build, run, release :** all these three process should be separated and should not run at once
   * **Stateless processes :** should not save any kind of information in session or DB like bank detail or credit card details.
   * **Port Binding :** the port which we are using for any service should be known to service itself not to any other service or server
   * **Concurrency** : it should be scalable and can add the resources to handle the load balance
   * **Disposability** : shut down should be gracefully i.e all the open connection should be closed before server goes down.
   * **Dev-prod parity** : code should be same at all the env like Prod, dev, testing, etc so that it will for testing and debugging
   * **Logs** : it should be kept at centralized location like splunk
   * **Admin Process** : all the admin task should be an automated process

### **What is End to End Microservices Testing?**

* + UI testing: is to valiadate whole system
  + End to End test : is to validate API gateway, Backend services along with DB
  + Integration testing: is to validate all Backend services along with DB
  + Unit testing : is to validate just the functionality of a single method.

#### **What are the steps in an End-to-End Microservices Testing?**

End-To-End testing of a microservice application ensures that every process in the form is running properly. This validates that the system as a whole is working properly. As the microservices application is built with multiple modules orchestrated dynamically, an end to end testing covers all the gaps between the services.

The steps to an end-to-end microservices testing are:

* Define what you expect from an e2e testing.
* Define the scope of the system to be tested.
* Perform authentication in a test environment.
* Choose a testing framework that addresses most of the issues.
* Test Asynchronous flows
* Automate Testing

### **What is the purpose of Docker?**

Docker provides a container environment that can be used to host any application. In this, the software application and the dependencies which support it are tightly-packaged together.

So, this packaged product is called a **Container** and since it is done by Docker, it is called **Docker container!**

#### **How does Docker help in Microservices?**

Microservices, as we know, are self-contained, individual units that perform only one business function, so much so that each unit can be considered an application on its own. The application development environment and application deployment environment are bound to vary in many aspects. This gives rise to deployment issues. Docker provides a static background for the application to run, thus avoiding deployment issues. It is, in fact, a containerization tool. It reduces overhead and deploys thousands of microservices on the same server. Docker ensures that an application microservices will run on their own environments and are entirely separate from their operating system.

#### **What are the main components of Microservices?**

* + Containers, Clustering, and Orchestration
  + IaC [Infrastructure as Code Conception]
  + Cloud Infrastructure
  + API Gateway
  + Enterprise Service Bus
  + Service Delivery

#### **How do you implement a Spring Security in a Spring Boot Application?**

* + Add spring-boot-starter-security in the file pom.xml.
  + Create a Spring config class that will override the required method while extending the WebSecurityConfigurerAdapter to achieve security in the application.

#### **What is Distributed Transaction?**

Distribution transaction has two or more network hosts that are engaged. Transactions are handled by a transaction manager that takes care of developing and handling transactions. If the transaction involves more than one peer, transaction managers of each peer communicate with each other using subordinate or superior relationships.

Same way, resources are handled by the resource manager that also coordinates with the distributed transaction coordinator for transaction atomicity and isolation.

#### **What is OAuth?**

OAuth stands for Open-standard Authorization Protocol or framework that describes how unrelated servers and services can safely allow authenticated access to their assets without sharing the initial related, single login credential. This is also known as secure, third-party, user-agent, delegated authorization.