**Microservices questions and answers**

In microservice architecture application divided into small part or modules, those small modules are known as microservices.

Monolithic app will be at one place and runs on one server.

**Q) Difference between Monolithic architecture and microservices.**

**🡪** The main difference is that monolithic architecture is tightly coupled and agility is not there, also application runs on single server, continuous integration and deployment is not possible, need to wait for all team mates to complete their respective tasks then only can integrate and deploy, if we want to do any changes as it is tightly coupled it will create a problem (big ball of mud) means other dependent services will also get affected

Microservices architecture is loosely coupled with the services, agility is there, every server runs on different servers, continuous integration and deployment is there, no need to wait for other team mates to complete their task, we can do changes easily in microservices

| **Monolithic** | **Microservices** |
| --- | --- |
| Whole application runs on a single server | Every service runs on different server |
| It is tightly coupled | It is loosely coupled |
| If we want to do any changes in application it will create conflicts and difficult to perform changes as it is tightly coupled, known as big ball of mud | We can perform changes in microservices as it is loosely coupled |
| The main drawback of monolithic architecture is continuous integration and deployment | We can do continuous integration and deployment |
| We need to wait for everyone’s service to be completed for deployment | We don’t need to wait for other team members for deployment |
| Agility is not there in monolothic | Agility is there in microservices |
|  |  |
|  |  |

**1) What Is Spring Cloud Gateway?**

🡪 Spring cloud provide us one thing that is Spring cloud gateway(previously named as api gateway and it had lot of issues)

Spring cloud gateway will check from which devices the request has been came. It will actually generate code and convert in such a format and send to particular service

It is also called as router. It will route clients request.

* It will do basic authentication and authorization
* It will do the loas balancing. Load balancing means if multiple/lot of request coming to order service you will have to create multiple instances of this order it will automatically some request to every instances.

For microservices architecture, spring cloud helps in all the above things and more, making it easier to develop, use, and manage microservices. Spring Cloud is a**set of libraries to create applications that run on On-Premises or Cloud-like Cloud Foundry, AWS**, etc.

Spring Cloud provides a solution to the commonly encountered patterns when developing a distributed system.

# **What is Spring Cloud?**

Spring Cloud is a framework for building robust cloud applications. Spring Cloud provides a solution to the commonly encountered patterns when developing a distributed system.

## Why is Spring Cloud used?

Spring Cloud framework provides tools for developers to build a robust cloud application quickly. We can also build the microservice-based applications, for example, **configuration management, service discovery, circuit breakers, intelligent routing, cluster state, micro-proxy, a control bus, one time tokens, etc**. Using Spring Cloud, a developer can quickly develop services and applications that implement the design patterns. These patterns work well in any distributed environment, including the **bear metal data centers, developer's laptop,** and managed platform such as **Cloud Foundry**.

**Bear metal data centers:** It is a physical server dedicated to a single-tenant (a person who occupies server on rent). These are not shared between the customers. The tenant can optimize the performance according to its needs for performance, security, and reliability. The bare metal data centers are also known as Single-tenant physical server or managed dedicated server. The operating system is installed directly on the bare metal server and delivers better performance.

**Cloud Foundry:** Cloud Foundry is an open-source, multi-cloud Platform as a Service (PaaS). You can deploy your application on your own computing infrastructure.

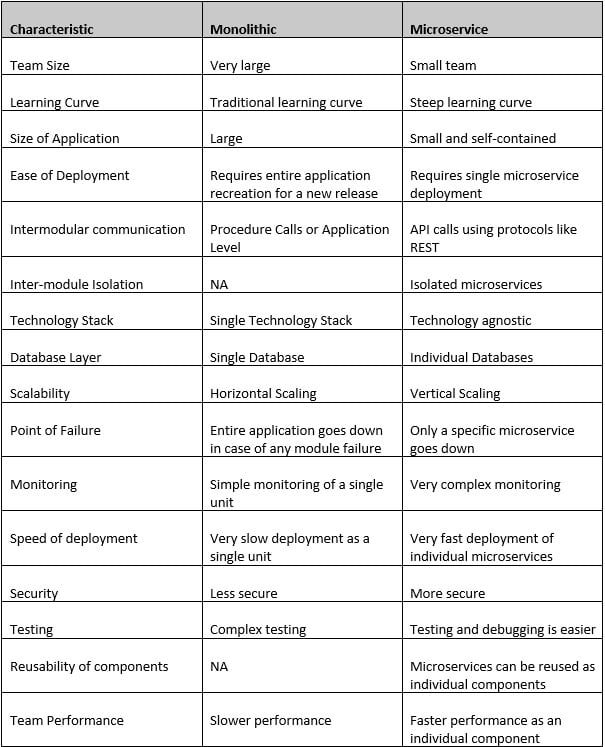
**2) What is discovery server? Why there is a need of discovery server? How to set up discovery server?**

**🡪 (Eureka server/ Discovery server/ Registry 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 Server is also known as Discovery Server.

To keep track of all microservice locations

Therefore, we need a service registry (Discovery server)**to keep track of all microservice locations**. So that every service needs to be accessed by the other service that registers itself with a discovery server (friendly name IP + Port) E.g: app: {10.0.1.0: 8080}.

**3) Monolithic Apps vs Microservices. What are the advantages of microservices?** 

**4) What are the challenges faced while using Microservices?**

**🡪** The challenges that one has to face while using microservices can be both functional and technical as given below: Functional Challenges:**Require heavy infrastructure setup. Need Heavy investment. Require excessive planning to handle or manage operations overhead**. Technical Challenges: Microservices are always interdependent.

**5) How independent micro-services communicate with each other?**

**🡪** RestTemplate is**class using that easily communication between microservices is possible**.

While developing microservices, Its always requires communicating between microservices. Data exposed by one microservices should be required to read from another microservices.  Spring boot also provides good solution for communicate between microservices using Rest API. RestTemplate is class using that easily communication between microservices is possible. Spring boot RestTemplate Example:

RestTemplateBuilder class is used to createRestTemplate class. RestTemplateBuilder bean automatically created by spring boot.

@Autowired

private RestTemplateBuilder restTemplate;

RestTemplate Get method Example

@Autowired

**private** RestTemplateBuilder restTemplate;

**public** java.util.List<Employee> getAllEmployee() {

Employee[] employee = restTemplate.build()

.getForObject("http://localhost:8080/getAllEmployee", Employee[].class);

**return** Arrays.asList(employee);

}

**6) Principles/characteristics of microservices?**

Characteristics of microservices are:

* Single responsibility and loosely coupled
* Simple and focused on a specific business functionality,
* Can be developed independently from other teams and software,
* They are very lowly/loosely coupled to other components,
* They can be developed using different programming language and tools,
* Allows that applications can be subdivided in autonomous and independent modules.

**7) What is config server and why do we need it? How to implement it?**

**🡪** Spring Cloud **Config** **Server** enables you to store all the configuration for multiple microservices for different environments in a Git or a SVN Repository. A set of folder structures and conventions need to be followed for the setup to work.

Spring Cloud Config or Config Server is one of the project under Spring Cloud and is used for centralised external configuration.

**8) What is API gateway? Explain advantages of API gateway.**

**🡪** Let’s say we have different types of users, ios user, someone is accessing from desktop, someone is accessing from android app, so in order to use that app they need to directly contact with microservices i.e. not the ideal. So we have introduced the api gateway, all the requests will route through the api gateway, they have to just make request to api gateway, according to the route it will forwarded to the microservice.

And apart from routing there are other functionalities like monitoring is there, it does some kind of basic authentication as well.

**9) Why do we need hystrix? How to implement it? How to implement Hystrix Dashboard?**

**🡪** The Hystrix framework library helps to**control the interaction between services by providing fault tolerance and latency tolerance**. It improves overall resilience of the system by isolating the failing services and stopping the cascading effect of failures, and providing fallback options, all of which improve our system’s overall resiliency

It implements the circuit breaker pattern which work on circuit-breaker transitions from CLOSED to OPEN when a circuit meets a specified threshold and error percentage exceeds the threshold error percentage. While it is open, it short-circuits all requests made against that circuit-breaker. After some amount of time, the next single request is let through (this is the HALF-OPEN state). If the request fails, the circuit-breaker returns to the OPEN state for the duration of the sleep window. If the request succeeds, the circuit-breaker transitions to CLOSED and all requests made against that circuit-breaker are passed through to the service. More you can explore [here](https://github.com/Netflix/Hystrix/wiki/How-it-Works).

[fault tolerance is the property that enables the system to continue operating properly in the event of the failure of one or more faults within some of its components]

**10) What are different ways to deploy microservices?**

**🡪** From simple to complex, here are the **five ways of running microservices**:

1. **Single machine, multiple processes**: buy or rent a server and run the microservices as processes.
2. **Multiple machines, multiple processes**: the obvious next step is adding more servers and distributing the load, offering more scalability and availability.
3. **Containers**: packaging the microservices inside a container makes it easier to deploy and run along with other services. It’s also the first step towards Kubernetes.
4. **Orchestrator**: orchestrators such as Kubernetes or Nomad are complete platforms designed to run thousands of containers simultaneously.
5. **Serverless**: serverless allows us to forget about processes, containers, and servers, and run code directly in the cloud.

**11) What is Domain Driven Design?**

**🡪** Domain-Driven Design microservices are the**key to simplifying microservice application development**. Microservices are small functions that solve a particular business problem. Decomposing your monolithic applications into microservices requires you to define these business functions into domains of expertise called a Domain-Driven Design (DDD).

**12) How to implement Distributed tracing in microservices? Why need of distributed tracing? => Explain zipkin server**

🡪 Distributed tracing, sometimes called distributed request tracing, is**a method to monitor applications built on a microservices architecture**. IT and DevOps teams use distributed tracing to follow the course of a request or transaction as it travels through the application that is being monitored.

**13) What are advantages and disadvantages of microservices?**

**Advantages:**

* Minimal work team
* Scalability
* Modular functionality, independent modules.
* Developer freedom to develop and deploy services independently
* Use of containers, allowing for a quick deployment and development of the application

**Disadvantages**

* High memory use
* Time required to fragment different microservices
* Complexity of managing a large number of services
* Developer need to solve problems such as network latency or load balancing
* Complex testing over the distributed deployment

**14) What are advantages and disadvantages of monolithic architecture?**

# Advantages of Monolithic Architecture

* **Simplicity of development**. The monolithic approach is a standard way of building applications. No additional knowledge is required. All source code is located in one place which can be quickly understood.
* **Simplicity of debugging**. The debugging process is simple because all code is located in one place. You can easily follow the flow of a request and find an issue.
* **Simplicity of testing**. You test only one service without any dependencies. Everything is usually clear.
* **Simplicity of deployment**. Only one deployment unit (e.g. jar file) should be deployed. There are no dependencies. In cases when UI is packaged with backend code you do not have any breaking changes. Everything exists and changes in one place.
* **Simplicity of application evolution**. Basically, the application does not have any limitation from a business logic perspective. If you need some data for new feature, it is already there.
* **Cross-cutting concerns and customizations are used only once**. You should take care of cross-cutting concerns only once. For instance, security, logging, exception handling, monitoring, choosing and setting up [tomcat](http://tomcat.apache.org/) parameters, setup of data source connection pool, etc.
* **Simplicity in onboarding new team members**. The source code is located in one place. New team members can easily debug some functional flow and to get familiar with the application.
* **Low cost in the early stages of the application**. All source code is located in one place, packaged in a single deployment unit, and deployed. What can be easier? There is no overhead neither in infrastructure cost nor development cost.

Because of these advantages, monolithic architecture is usually used in the early stages of application development. The reasons for that are the next ones:

* **The main function of the application is to be profitable**. As a result, it is important to quickly implement some POC (Proof of Concept) solutions to verify the application in the real world. Also, it is important to bring customers to the system. Improvements can be implemented in the future.
* **The requirements are usually unclear at the early stages of development**. It is hard to create meaningful architecture when the requirements are unclear. Real customers can define the business needs after some of the functionality already works.

The problems with Monolithic Architecture start to appear when the application becomes successful. The reason for this is very simple: the growth of the application. Usually, after some period of time, the Monolithic Application changes into another one because of the next reasons.

# Disadvantages of Monolithic Architecture

* **Slow speed of development**. The simplest disadvantage relates to [CI/CD pipeline](https://datamify.com/java-static-code-analysis-overview/). Imagine the monolith that contains a lot of services. Each service in this monolith is covered with tests that are executed for each Pull Request. Even for a small change in a source code, you should wait a lot of time (e.g. 1 hour) for your pipeline to succeed. And what happens when the pipeline fails for some reason? You wait again. All services are located in a single place. The size of the team is big. What happens when your colleague merges their changes? You rebase/merge and wait again.
* **High code coupling**. Of course, you can keep a clear service structure inside your repository. However, as practice shows, eventually, you will end up with a spaghetti code in at least a few places. As a result, the system becomes harder to understand especially for new team members.
* **Code ownership cannot be used**. The system is growing. The logical step is to split responsibilities between several teams. E.g. one team can work on Flight Service, another — for Billing Service. However, there are no boundaries between those services. One team can affect another.
* **Testing becomes harder**. Even a small change can negatively affect the system. As a result, the regression for full monolithic service is required.
* **Performance issues**. Potentially, you can scale the whole monolithic service in cases of performance issues. But what to do with the database? The single database is used for all services. You can start to optimize your database queries or use read replicas. However, there is a limit to this type of optimization.
* **The cost of infrastructure**. In cases of performance issues, you should scale the whole monolithic service. It brings additional costs for application operability.
* **Legacy technologies**. Imagine that you have the application written on Java 8. How much time is it required to migrate the whole monolith with multiple services underneath to Java 11? What to do with the tasks that are required to bring new functionality? It can be the case that the application will never be migrated.
* **Lack of flexibility**. Using Monolithic Architecture you are tight to the technologies that are used inside your monolith. You cannot use other tools even if they are better for the problem at hand.
* **Problems with deployment**. Even a small change requires the redeployment of the whole monolith.

**15) What is the biggest issue in monolithic application and how its been resolved in microservices?**

* Biggest issue is with assembling and deployment which was major difficult task also agility was no there in monolithic application. Its been resolved in microservices, as loose coupling is there in microservices so continuous development and deployment is possible.

**16) Explain spring cloud and gateway**

**🡪** Spring Cloud Gateway is a**non-blocking and an open source API Gateway**. It is built on top of the Spring Ecosystem, including Spring 5, Spring Boot 2 and Project Reactor. The Gateway is about routing requests/responses between clients and API's and provide some central cross-cutting capabilities such as security, metrics, etc.

**17) What is discovery design pattern? Why is it used in microservices?**

**🡪** In the Microservices architecture, service discovery is very important to implement. It helps client application to search for services without hardcode network location. We can implement this service discovery pattern in two ways such as**client-side discovery and server-side discovery**.

**Use of integration design pattern?**

**Why do we need to have separate databases?**

**Why do we need hystrix?**

**What is Circuit Breaker in Microservices?**

In a microservice based application, Circuit Breaker is a technique, where we stop executing an erroneous method and redirect every request to a custom method (Fallback method). Generally, we stop execution of a particular method if it is continuously throwing an exception. When we break the circuit, we also avoid any cascading failures to the downstream services. It’s basic function is to interrupt current flow after a fault is detected. A circuit breaker can be reset to resume normal operation either manually or automatically.

**What is a Fallback Method in Microservices?**

If an actual method of a microservice is throwing exception continuously, then we avoid execution of the actual logic for some time. Instead, we redirect the request to a Dummy method that provides the response back to client’s request. Such dummy method is called as Fallback method. This method can provide dummy responses such as ‘Service Not Working’, ‘Unable to Process’, ‘try after some time’ etc.

**Why do we need a Circuit Breaker?**

It’s very common for software applications to make remote calls to a software program, possibly running on different machines across a network. One of the big differences between in-memory calls and remote calls is that remote calls may fail, or hang without a response until some timeout limit is reached. What’s worse if you have many requests on a unresponsive service, then you can run out of critical resources leading to cascading failures across multiple systems. [Michael Nygard](https://www.oreilly.com/pub/au/2781#:~:text=Michael%20Nygard%20has%20been%20a,of%20the%20earliest%20Java%20books.) in his wonderful book [Release It](https://amzn.to/2PUEZdY) has publicized the Circuit Breaker pattern to prevent this kind of fatal cascade. The basic idea behind the circuit breaker is very simple. You wrap a protected function call in a circuit breaker object, which monitors for failures. Once the failures reach a certain threshold, the circuit breaker falls.

**What are the states of Circuit Breaker?**

Depending on the state, Circuit Breaker changes it’s behavior. There are three states of a Circuit Breaker.

**Closed**

If Client request is sent to actual service method only, then it is called as CLOSED CIRCUIT. Hence, this states represents that service is running properly and providing the expected functionality.

**Open**

If Client request is redirected to Fallback method then such case is an OPEN CIRCUIT. Hence, this state represents that service is unavailable or faulty and error rate is beyond the threshold.

**Half-Open**

Once the state becomes OPEN, we wait for sometime in the OPEN state. After certain period of time, the state becomes HALF\_OPEN.  
During this period, we do send some requests to Service to check if we still get the proper response. If the failure rate is below the threshold, the state would become CLOSED. If the failure rate is above the threshold, then the state becomes OPEN once again. This cycle continues till the service becomes stable.