1. What is the difference between manual testing and automated testing?

Both manual and automated testing offer benefits and disadvantages. It's worth knowing the difference, and when to use one or the other for best results. In manual testing (as the name suggests), test cases are executed manually (by a human, that is) without any support from tools or scripts. But with automated testing, test cases are executed with the assistance of tools, scripts, and software. Testing is an integral part of any successful software project. The type of testing (manual or automated) depends on various factors, including project requirements, budget, timeline, expertise, and suitability. Three vital factors of any project are of course time, cost, and quality – the goal of any successful project is to reduce the cost and time required to complete it successfully while maintaining quality output. When it comes to testing, one type may accomplish this goal better than the other.

2. What does Assert class?

Assert is a method useful in determining Pass or Fail status of a test case, The assert methods are provided by the class org.junit.Assert which extends java.lang.Object class.

There are various types of assertions like Boolean, Null, Identical etc.

Junit provides a class named Assert, which provides a bunch of assertion methods useful in writing test cases and to detect test failure.

The assert methods are provided by the class org.junit.Assert which extends java.lang.Object class.

3. How can be tested 'private' methods?

Private methods are great for breaking logic into smaller parts. If a method gets too big, you should refactor it and break it down into private methods so it's easier to read. But you don't need to test those private methods individually.

To test private methods, you just need to test the public methods that call them. Call your public method and make assertions about the result or the state of the object. If the tests pass, you know your private methods are working correctly.

Whatever you do, don't change private methods to public just so you can test them. Test your public methods and verify that they work, and your private methods will be covered, too.

4. What is Monolithic Architecture?

A monolithic architecture is the traditional unified model for the design of a software program. Monolithic, in this context, means composed all in one piece. According to the Cambridge dictionary, the adjective monolithic also means both too large and unable to be changed. Monolithic software is designed to be self-contained; components of the program are interconnected and interdependent rather than loosely coupled as is the case with modular software programs. In a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled. Furthermore, if any program component must be updated, the whole application has to be rewritten, whereas in a modular application, any separate module (such as a microservice) can

be changed without affecting other parts of the program. Modular architectures reduce the risk that a change made within one element will create unanticipated changes within other elements, because modules are relatively independent. Modular programs also lend themselves to iterative processes more readily than monolithic programs. However, there are benefits to monolithic architectures as well. Monolithic programs typically have better throughput than modular approaches, such as the microservice architecture (MSA) and they can be easier to test and debug because, with fewer elements there are fewer variables that come into play.

5. What are the best practices to write a Unit Test Case?

Essential Unit Test Best Practices:

1) Tests Should Be Fast
2) Tests Should Be Simple
3) Test Shouldn't Duplicate Implementation Logic
4) Tests Should Be Readable
5) Tests Should Be Deterministic
6) Make Sure They're Part of the Build Process
7) Distinguish Between the Many Types of Test Doubles and Use Them Appropriately
8) Adopt a Sound Naming Convention for Your Tests
9) Don't Couple Your Tests with Implementation Details

6. Why does JUnit only report the first failure in a single test?

JUnit only report the first failure in a single test because reporting multiple failures in a single test is generally a sign that the test does too much, and it is too big a unit test. JUnit is designed to work best with a number of small tests. It executes each test within a separate instance of the test class. It reports failure on each test.

7. What are the benefits and drawbacks of Microservices?

Microservices have become hugely popular in recent years. Mainly, because they come with a couple of benefits that are super useful in the era of containerization and cloud computing. You can develop and deploy each microservice on a different platform, using different programming languages and developer tools. Microservices use APIs and communication protocols to interact with each other, but they don't rely on each other otherwise. The biggest pro of microservices architecture is that teams can develop, maintain, and deploy each microservice independently. This kind of single-responsibility leads to other benefits as well. Applications composed of microservices scale better, as you can scale them separately, whenever it's necessary. Microservices also reduce the time to market and speed up your CI/CD pipeline. This means more agility, too. Besides, isolated services have a better failure tolerance. It's easier to maintain and debug a lightweight microservice than a complex application, after all.

As microservices heavily rely on messaging, they can face certain problems. Communication can be hard without using automation and advanced methodologies such as Agile. You need to introduce DevOps tools such as CI/CD servers, configuration management platforms, and APM tools to manage the network. This is great for companies who already use these methods. However, the adoption of

these extra requirements can be a challenge for smaller companies. Having to maintain a network lead to other kinds of issues, too. What we gain on the simplicity of single-responsibility microservices, lose on the complexity of the network. Or at least a part of it. For instance, while independent microservices have better fault tolerance than monolithic applications, the network has worse.

8. What is the role of actuator in spring boot?

In essence, Actuator brings production-ready features to our application. Monitoring our app, gathering metrics, understanding traffic, or the state of our database become trivial with this dependency.

The main benefit of this library is that we can get production-grade tools without having to implement these features ourselves.

Actuator is mainly used to expose operational information about the running application — health, metrics, info, dump, env, etc. It uses HTTP endpoints or JMX beans to enable us to interact with it.

Once this dependency is on the classpath, several endpoints are available for us out of the box. As with most Spring modules, we can easily configure or extend it in many ways.

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

Design: Compared to monolithic apps, organizations face increased complexity when designing microservices.

Security: Microservices are often deployed across multi-cloud environments, resulting in increased risk and loss of control and visibility of application components—resulting in additional vulnerable points.

Testing: The testing phase of any software development lifecycle (SDLC) is increasingly complex for microservices-based applications. Given the standalone nature of each microservice, you have to test individual services independently.

Increased operational complexity: Each microservice's team is usually tasked with deciding the technology to use and manage it. As each service should be deployed and operated independently, maintaining operations may open a can of worms for those who are not prepared.

Communication: Independently deployed microservices act as miniature standalone applications that communicate with each other. To achieve this, you have to configure infrastructure layers that enable resource sharing across services.

10. How independent microservices communicate with each other?

One of the biggest challenges when moving to microservices-based application is changing the communication mechanism. Because microservices are distributed and microservices communicate with each other by inter-service communication on network level. Each microservice has its own instance and process. Therefore, services must interact using an inter-service communication protocols like HTTP, gRPC or message brokers AMQP protocol.

11. What do you mean by Domain driven design?

Domain-driven design is the idea of solving problems of the organization through code. The business goal is important to the business users, with a clear interface and functions. This way, the microservice can run independently from other microservices. Moreover, the team can also work on it independently, which is, in fact, the point of the microservice architecture. Many developers claim microservices have made them more efficient. This is due to the ability to work in small teams. This allows them to develop different small parts that will later be merged as a large app. They spend less time coordinating with other developers and more time on developing the actual code. Eventually, this creates more value for the end-user.

12. What is container in Microservices?

Containers are a form of operating system virtualization. A single container might be used to run anything from a small microservice or software process to a larger application. Inside a container are all the necessary executables, binary code, libraries, and configuration files. Compared to server or machine virtualization approaches, however, containers do not contain operating system images. This makes them more lightweight and portable, with significantly less overhead. In larger application deployments, multiple containers may be deployed as one or more container clusters. Such clusters might be managed by a container orchestrator such as Kubernetes.

13. What are the main components of Microservices architecture?

Clients: The client apps usually need to consume functionality from more than one microservice.

Identity Providers: The services are fine-grained and lightweight. The Identity Microservice must allow both user-driven and server-to-server access to identity data.

API Gateway: The API Gateway is responsible for request routing, composition, and protocol translation. It provides each of the application's clients with a custom API.

Messaging Formats: Synchronize and Asynchronized are the 2 types of messages through which they communicate. Every microservice in order to communicate either synchronously or asynchronously with other microservices.

Databases: Microservice owns a private database to capture their data and implement the respective business functionality.

Static Content: After the microservices communicate within themselves, they deploy the static content to a cloud-based storage service that can deliver them directly to the clients via Content Delivery Networks (CDNs).

Management: Management feature is a capability that allows operations and business users to configure services in run-time.

Service Discovery: In a microservices application, the set of running service instances changes dynamically. Instances have dynamically assigned network locations.

14. How does a Microservice architecture work?

A microservice attempts to address a single concern, such as a data search, logging function, or web service function. This approach increases flexibility—for example, updating the code of a single function without having to refactor or even redeploy the rest of the microservices architecture. The failure points are more independent of each other creating a more stable overall application architecture. This approach also creates an opportunity for microservices to become self-healing. For example, suppose that a microservice in one cluster contains three subfunctions. If one of those subfunctions fails, the is being repaired. With orchestration tools such as Kubernetes, self-healing can occur without human intervention; it occurs behind the scenes, is automatic, and is transparent to the end user.