Interview Questions on Spring Boot

Spring vs Spring Boot

Spring	Spring Boot
A web application framework based on Java	A module of Spring
Provides tools and libraries to create customized web applications	Used to create a Spring application project which can just run/ execute
Spring is more complex than Spring Boot	Spring Boot is less complex than the Spring framework
Takes an unopinionated view	Takes an opinionated view of a platform

What is Spring Boot and mention the need for it?

Spring Boot is a Spring module which aims to simplify the use of the Spring framework for Java development. It is used to a create stand-alone Spring-based applications which you can just run. So, it basically removes a lot of configurations and dependencies. Aiming at the Rapid Application Development, Spring Boot framework comes with the auto-dependency resolution, embedded HTTP servers, auto-configuration, management endpoints, and Spring Boot CLI.

So, if you ask me why should anybody use Spring Boot, then I would say, Spring Boot not only improves productivity but also provides a lot of conveniences to write your own business logic.

Mention the advantages of Spring Boot

The advantages of Spring Boot are as follows:

- Provides auto-configuration to load a set of default configuration for a quick start of the application
- Creates stand-alone applications with a range of non-functional features that are common to large classes of projects
- It comes with embedded tomcat, servlet containers jetty to avoid the usage of WAR files
- Spring Boot provides an opinionated view to reduce the developer effort and simplify maven configurations
- Provides CLI tool to develop and test applications
- Comes with Spring Boot starters to ensure dependency management and also provides various security metrics
- Consists of a wide range of APIs for monitoring and managing applications in dev and prod.
- Integrates with Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security easily by avoiding boilerplate code.

Mention a few features of Spring Boot

Few important features of Spring Boot are as follows:

1. Spring CLI – Spring Boot CLI allows you to Groovy for writing Spring boot application and avoids boilerplate code.
2. Starter Dependency – With the help of this feature, Spring Boot aggregates common dependencies together and eventually improves productivity
3. Spring Initializer – This is basically a web application, which can create an internal project structure for you. So, you do not have to manually set up the structure of the project, instead, you can use this feature.
4. Auto-Configuration – The auto-configuration feature of Spring Boot helps in loading the default configurations according to the project you are working on. In this way, you can avoid any unnecessary WAR files.
5. Spring Actuator – This feature provides help while running Spring Boot applications.
6. Logging and Security – The logging and security feature of Spring Boot, ensures that all the applications made using Spring Boot are properly secured without any hassle.

Explain how to create Spring Boot application using Maven

Well, there are various approaches to create a Spring Boot application using maven, but if I have to name a few, then following are the ways to create a Spring Boot project/ application using maven:

- Spring Boot CLI
- Spring Starter Project Wizard
- Spring Initializr
- Spring Maven Project

Can you explain what happens in the background when a Spring Boot Application is "Run as Java Application"?

When a Spring Boot application is executed as "Run as Java application", then it automatically launches up the tomcat server as soon as it sees, that you are developing a web application.

Mention the possible sources of external configuration

There is no doubt in the fact that Spring Boot allows the developers to run the same application in different environments. Well, this is done with the support it provides for external configuration. It uses environment variables, properties files, command-line arguments, YAML files, and system properties to mention the required configuration properties. Also, the @value annotation is used to gain access to the properties. So, the most possible sources of external configuration are as follows:

- **Application Properties –** By default, Spring Boot searches for the application properties file or its YAML file in the current directory, classpath root or config directory to load the properties.

- **Command-line properties –** Spring Boot provides command-line arguments and converts these arguments to properties. Then it adds them to the set of environment properties.

- **Profile-specific properties –** These properties are loaded from the application-{profile}.properties file or its YAML file. This file resides in the same location as that of the non-specific property files and the{profile} placeholder refers to an active profile.

What are the Spring Boot starters and what are available the starters?

Spring Boot starters are a set of convenient dependency management providers which can be used in the application to enable dependencies. These starters, make development easy and rapid. All the available starters come under the org.springframework.boot group. Few of the popular starters are as follows:

- *spring-boot-starter:* – This is the core starter and includes logging, auto-configuration support, and YAML.
- *spring-boot-starter-jdbc* – This starter is used for HikariCP connection pool with JDBC
- *spring-boot-starter-web* – Is the starter for building web applications, including RESTful, applications using Spring MVC
- *spring-boot-starter-data-jpa* – Is the starter to use Spring Data JPA with Hibernate
- *spring-boot-starter-security* – Is the starter used for Spring Security
- *spring-boot-starter-aop:* This starter is used for aspect-oriented programming with AspectJ and Spring AOP
- *spring-boot-starter-test:* Is the starter for testing Spring Boot applications

Explain Spring Actuator and its advantages

Spring Actuator is a cool feature of Spring Boot with the help of which you can see what is happening inside a running application. So, whenever you want to debug your application, and need to analyze the logs you need to understand what is happening in the application right? In such a scenario, the Spring Actuator provides easy access to features such as identifying beans, CPU usage, etc. The Spring Actuator provides a very easy way to access the production-ready REST points and fetch all kinds of information from the web. These points are secured using Spring Security's content negotiation strategy.

What is Spring Boot dependency management?

Spring Boot dependency management is basically used to manage dependencies and configuration automatically without you specifying the version for any of that dependencies.

Explain what is thymeleaf and how to use thymeleaf?

Thymeleaf is a server-side Java template engine used for web applications. It aims to bring natural template for your web application and can integrate well with Spring Framework and HTML5 Java web applications. To use Thymeleaf, you need to add the following code in the pom.xml file:

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Can we change the port of the embedded Tomcat server in Spring boot?

Yes, we can change the port of the embedded tomcat server by using the application properties file. In this file, you have to add a property of “server.port” and assign it to any port you wish to. For example, if you want to assign it to 8081, then you have to mention server.port=8081. Once you mention the port number, the application properties file will be automatically loaded by Spring Boot and the required configurations will be applied on to the application.

What is the need for Spring Boot DevTools?

Spring Boot Dev Tools are an elaborated set of tools and aims to make the process of developing an application easier. If the application runs in the production, then this module is automatically disabled, repackaging of archives are also excluded by default. So, the Spring Boot Developer Tools applies properties to the respective development environments. To include the DevTools, you just have to add the following dependency into the pom.xml file:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

Mention the steps to create a Spring Boot project using Spring Initializr

Spring Initializr is a web tool provided by Spring. With the help of this tool, you can create Spring Boot projects by just providing project details. The following steps need to be followed to create a Spring Boot project using Spring Initializr:

- Choose the maven project and the required dependencies. Then, fill the other required details like Group, Artifact, and then click on Generate Project.
- Once the project is downloaded, extract the project onto your system

- Next, you have to import this project using the import option on the Spring Tool Suite IDE
  - While importing the project, remember that you have to choose the project type to be Maven and the source project should contain the pom.xml file.

Once, all the above steps are followed you will see that the Spring Boot project is created with all the required dependencies.

Mention the steps to connect Spring Boot application to a database using JDBC

Spring Boot starter projects provide the required libraries to connect the application with JDBC. So, for example, if you just have to create an application and connect it with MySQL database, you can follow the below steps:

Step 1: Create a database in MySQL

```
1 | CREATE DATABASE example;
```

Step 2: Then you have to create a table inside this database.

```
1 | CREATE TABLE customers(customerid INT PRIMARY KEY NOT NULL AUTO_INCREMENT, cust
```

Step 3: Now, create a Spring Boot project and provide the required details

Step 4: Add the JDBC, MySQL and web dependencies.

Step 5: Once the project is created, you have to configure the database into application properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/example
spring.datasource.username=root
spring.datasource.password=edureka
spring.jpa.hibernate.ddl-auto=create-drop
```

Step 6: The main application.java class should have the following code:

```
package com.edureka;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SampleApplication {
    public static void main(String[] args) {
        SpringApplication.run(SampleApplication.class, args);
    }
}
```

Step 7: Next, you have to create a controller to handle the HTTP requests, by mentioning the following code:

```
package com.edureka;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class JdbcController {
@Autowired
JdbcTemplate jdbc;
@RequestMapping("/insert")
public String index(){
jdbc.execute("insert into customers(name)values('Aryya')");
return "Data Entry Successful";
}
}
```

Step 8: Finally, execute this project as a Java application.

Step 9: Next, open the URL (localhost:8080/insert), and you will see the output as Data Entry Successful. You can also go forward and check if the data is entered into the table.

What are the @RequestMapping and @RestController annotation in Spring Boot used for?

@RequestMapping	@RestController
This annotation is used to provide the routing information and tells to Spring that any HTTP request must be mapped to the respective method.	This annotation is used to add the @ResponseBody and @Controller annotation to the class
To use this annotation, you have to import org.springframework.web. bind.annotation.RequestMapping;	To use this annotation, you have to import org.springframework.web. bind.annotation.RestController;

Example: Consider you have a method example() which should map with /example URL.

```
1  package com.edureka;
2  import org.springframework.web.bind.annotation.RequestMapping;
3  import org.springframework.web.bind.annotation.RestController;
4  @RestController
5  public class SampleController {
6  @RequestMapping("/example")
7  public String example(){
8  return"Welcome To Edureka";
9  }
```

Q20. Mention the differences between JPA and Hibernate

JPA	Hibernate
JPA is a Data Access Abstraction used to reduce the amount of boilerplate code	Hibernate is an implementation of Java Persistence API and offers benefits of loose coupling

Explain Spring Data

Spring Data aims to make it easy for the developers to use relational and non-relational databases, cloud-based data services, and other data access technologies. So, basically, it makes it easy for data access and still retains the underlying data.

What are the differences between @SpringBootApplication and @EnableAutoConfiguration annotation?

@SpringBootApplication	@EnableAutoConfiguration
Used in the main class or bootstrap class	Used to enable auto-configuration and component scanning in your project
It is a combination of @Configuration, @ComponentScan and @EnableAutoConfiguration annotations.	It is a combination of @Configuration and @ComponentScan annotations

What are the steps to deploy Spring Boot web applications as JAR and WAR files?

To deploy a Spring Boot web application, you just have to add the following plugin in the pom.xml file:

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

By using the above plugin, you will get a JAR executing the package phase. This JAR will contain all the necessary libraries and dependencies required. It will also contain an embedded server. So, you can basically run the application like an ordinary JAR file.

Note: The packaging element in the pom.xml file must be set to **jar** to build a JAR file as below:

```
<packaging>jar</packaging>
```

Similarly, if you want to build a WAR file, then you will mention

```
<packaging>war</packaging>
```

Can you give an example for ReadOnly as true in Transaction management?

Example for ReadOnly as TRUE in transaction management could be as follows:

Consider a scenario, where you have to read data from the database. For example, let us say you have a customer database, and you want to read the customer details such as customerID, and customername. To do that, you will set **read-only on the transaction** as we do not want to check for the changes in the entities.

Can you explain how to deploy to a different server with Spring Boot?

To deploy a different server with Spring Boot, follow the below steps:

- Generate a WAR from the project
- Then, deploy the WAR file onto your favorite server

Note: The steps to deploy the WAR file on the server is dependent on the server you choose.

Can we create a non-web application in Spring Boot?

Yes, we can create a non-web application by removing the web dependencies from the classpath along with changing the way Spring Boot creates the application context.

What are the steps to connect an external database like MySQL or Oracle?

To connect an external database, you have to follow the below steps:

- Start by adding the dependency for MySQL Connector to pom.xml
- Then remove H2 Dependency from pom.xml
- Now, set up your MySQL database and configure your connection to the MySQL database
- Restart your project

How is Hibernate chosen as the default implementation for JPA without any configuration?

When we use the Spring Boot Auto Configuration, automatically the spring**-boot-starter-data-jpa** dependency gets added to the pom.xml file. Now, since this dependency has a transitive dependency on JPA and Hibernate, Spring Boot automatically auto-configures Hibernate as the default implementation for JPA, whenever it sees Hibernate in the classpath.

What is the difference between RequestMapping and GetMapping?

The @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET). Both these methods support the consumes. The consume options are :

consumes = “text/plain”
consumes = {“text/plain”, “application/*”}

In which layer, should the boundary of a transaction start?

The boundary of the transaction should start from the Service Layer since the logic for the business transaction is present in this layer itself.

What do you think is the need for Profiles?

Profiles are used to provide a way to segregate the different parts of the application configuration and make it available for various environments. So, basically, any @Component or a @Configuration can be marked with a @Profile to limit as it is loaded. Consider you have multiple environments,

- Dev
- QA
- Stage
- Production

Now, let's say, you want to have different application configuration in each of the environments, you can use profiles to have different application configurations for different environments. So, basically, Spring and Spring Boot provide features through which you can specify:

- The active profile for a specific environment
- The configuration of various environments for various profiles.

What is the error you see if H2 is not in the classpath?

If H2 is not present in the classpath, then you see the following error:

```
Cannot determine embedded database driver class for database type NONE
```

To resolve this error, add H2 to the pom.xml file, and restart your server.

The following code snippet can be added to add the dependency:

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Mention the dependencies needed to start up a JPA Application and connect to in-memory database H2 with Spring Boot?

The dependencies are needed to start up a JPA Application and connect to in-memory database H2 with Spring Boot

- web starter
- h2
- data JPA starter
- To include the dependencies refer to the following code:

```
<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.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Do you think, you can use jetty instead of tomcat in spring-boot-starter-web?

Yes, we can use jetty instead of tomcat in spring-boot-starter-web, by removing the existing dependency and including the following:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

How to enable HTTPS/SSL support in Spring boot?

The SSL support in spring boot project can be added via `application.properties` and by adding the below entries.

application.properties

```
server.port=8443
server.ssl.key-alias=selfsigned_localhost_sslserver
server.ssl.key-password=changeit
server.ssl.key-store=classpath:ssl-server.jks
server.ssl.key-store-provider=SUN
server.ssl.key-store-type=JKS
```

11. What is Spring Actuator? What are its advantages?

Spring boot's actuator module allows us to monitor and manage application usages in production environment, without coding and configuration for any of them. These monitoring and management information is exposed via REST like endpoint URLs.

The simplest way to enable the features is to add a dependency to the `spring-boot-starter-actuator` starter pom file.

pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Spring Boot includes a number of built-in endpoints and lets we add our own. Further, each individual endpoint can be enabled or disabled as well.

ENDPOINT	USAGE
`/env`	Returns list of properties in current environment
`/health`	Returns application health information.
`/auditevents`	Returns all auto-configuration candidates and the reason why they 'were' or 'were not' applied.
`/beans`	Returns a complete list of all the Spring beans in your application.
`/trace`	Returns trace logs (by default the last 100 HTTP requests).
`/dump`	It performs a thread dump.
`/metrics`	It shows several useful metrics information like JVM memory used, system CPU usage, open files, and much more.

Q : How can I enable auto reload of my application with Spring Boot?

Use Spring Boot Developer Tools.

Adding Spring Boot Developer Tools to your project is very simple.

Add this dependency to your Spring Boot Project pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
</dependency>
```

Error : HAL browser gives me unauthorized error - Full authentication is required to access this resource. How can I fix it?

```
{
  "timestamp": 1488656019562,
  "status": 401,
  "error": "Unauthorized",
  "message": "Full authentication is required to access this resource.",
  "path": "/beans"
}
```

Two options

Option 1 : Disable security

application.properties

Option 2 : Search for password in the log and pass it in the request header

Spring Boot Interview Prep

What are the features of Spring boot providing to develop the microservices application?

Spring boot is predominately used to develop the micros services-based application, most of the key features leverage to ease the configuration development and deployment of the microservices architecture.

- Spring boot comes with the monitoring tool called as an Actuator which does the health check of spring boot production application.
- Externalised configuration

@Value("${cassandra.password}")

private String password;

- Embedded server's support for example Tomcat, Jetty.
- No need to deploy the war file, simply run it.
- Provide convention over configuration using Auto-Configuration feature, example like below annotation.

@EnableAutoConfiguration

What are the key features of spring boot which makes Spring boot superior over the JAX-RS?

There are significant advantages of using spring boot over the JAX-RS which is listed below.

- Easy deployment
- Simple scalability
- Compatible with Containers
- Minimum configuration
- Lesser production time
- Easy to understand, develop and configure the spring application
- Increase the productivity
- Reduce the development time.
- Provide the health check and monitoring feature.
- Easy to orchestrate using docker.

All this advantage makes spring boot is one of best alternative to develop the microservices application, along with one of the key benefits is, to make it compatible to use the other framework like messaging services, hibernate and spring cloud.

Why we should avoid the Spring boot framework?

Besides advantages, there are few issues, where we should think about to adopt the spring boot framework to develop the microservice based architecture.

- Spring boot unnecessary increase the size of the build with unused dependencies.
- Not able to create the war file manually and difficult to configure externally.
- Doesn't provide much control and tuning the running build.
- It's only suitable for micro-services which eventually need to deploy in docker, but not large or mono lithics web services.

Spring boot doesn't provide good compatibility if we are integrating third party framework.

Why do we encourage to use the spring boot over the other framework?

Spring boot provides good compatibility with other spring frameworks which is used to provide the security, persistency features. Spring boot provides good support with docker containerization, which makes it a good choice to deploy the microservice based application and easy to maintain.

- Provide the easiest way to configure the java beans.
- Provide a powerful batch processing and manage rest endpoints.
- Provide auto-configuration mechanism, that means no manual configuration needed.
- Provide annotation-based configuration, so no need to configure the xml file manually.
- Ease the dependency management.
- It includes the Embedded servlet container.

Spring boot comes with spring cloud framework, which has many libraries which are used to handle all types of nonfunctional requirement, which is usually not available in other frameworks.

How spring boot work internally?

Spring boot provides many abstraction layers to ease the development, underneath there are vital libraries which work for us.

Below is the key function performing internally.

- Using @EnableAutoConfigure annotation the spring boot application configures the spring boot application automatically.
- E.g. If you need MySQL DB in your project, but you haven't configured any database connection, in that case, Spring boot auto configures as in memory database.
- The entry point of spring boot application is a class which contains @SpringBootApplication annotation and has the main method.
- Spring boot scan all the components included in the project by using @ComponentScan annotation.
- Let's say we need the Spring and JPA for database connection, then we no need to add the individual dependency we can simply add the spring-boot-starter-data-jpa in the project.
- Spring boot follows the naming convention for dependency like spring-boot-starter.

What is the important dependency of spring boot application?

Below are the key dependencies which you need to add in maven based or Gradle based applications, to make the application compatible to use spring boot functionality.

- spring-boot-starter-parent
- spring-boot-starter-web
- spring-boot-starter-actuator
- spring-boot-starter-security
- spring-boot-starter-test
- spring-boot-maven-plugin

These dependencies come with associated child dependencies, which are also downloaded as a part of parent dependencies.

. What is a Spring-boot interceptor? Why do we need this features and is there any real use case scenario where spring boot interceptor fits?

Spring boot interceptor is typically used to intercept the request and response call made by the UI and microservices-based application, the need of this to add, filter, modified the information contain in request and response.

- Interceptor in Spring Boot one can use to add the request header before sending the request to the controller
- Interceptor in Spring Boot can add the response header before sending the response to the client.
- Spring boot works on the below technique.
    - Before sending the request to the controller
    - Before sending the response to the client
    - After completing the request and response.

The real-world use case of spring-boot interceptor is authentication and authorization, where we filter the information from the request which contain the credential information which use to authenticate and other information like role which require authorization.

. What are the important annotation for Spring rest? What is the use case of this annotation?

- **@RestController:** Define at class level, so that spring container will consider as RestENd point
- **@RequestMapping(value = "/products"):** Define the REST URL for method level.
- **@PathVariable:** Define as a method argument
- **@RequestBody:** Define as a method argument
- **@ResponseEntity:** To convert the domain object into the response format
- **@hasAuthority:** To grant the access of corresponding endpoints
- **@GetMapping:** To make endpoint compatible for get request.
- **@PostMapping:** To make endpoint compatible for post request.
- **@PutMapping:** To make endpoint compatible for put request.
- **@DeleteMapping:** To make endpoint compatible for delete request.
- **@ResponseStatus:** To generate the HTTP status.
- **@ResponseBody:** To Generate the response message.

Explain the main or entry-level method of Spring Boot Application.

Main Spring Boot class typically looks like:

```
@SpringBootApplication
public class SpringBoot1Application {
public static void main(String[] args) {
SpringApplication.run(SpringBoot1Application.class, args);
}
}
```

As you can see above, the main method calls static run() method SpringApplication class and provides itself(the SpringBoot1Application.class) & args. Now the class supplied (SpringBoot1Application.class) is annotated with @SpringBootApplication.

Now, SpringBootApplication annotation is defined as :

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
@Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
@Filter(type = FilterType.CUSTOM,
classes = AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {
```

For example, a typical configuration might look like:

```
@Configuration
public class MyConfiguration {
@Bean
public MyBean1 myBean()
{
return new MyBean1();
}
@Bean
public MyBean2 myBean()
{
return new MyBean2();
}
@Bean
public MyBean1 myBean()
{
return new MyBean1();
}
}
```

. Is it possible to have a Spring Boot Application without using @SpringBootApplication annotation? If yes, explain why would you do that?

@SpringBootApplication annotation is not mandatory, it is possible to have a Spring boot based application without using it.

You will do that when you don't want to go with implicit features provided by @SpringBootApplication annotation.

SpringBootApplication annotation is defined as :

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
@Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
@Filter(type = FilterType.CUSTOM,
classes = AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {
```

Let's say you don't want to use component scan in your application but would like to go with enabling feature that executes the code for dependencies with default configuration (@EnableAutoConfiguration)

You can probably define your class as :

```
 @Configuration
@EnableAutoConfiguration
@Import({ MyConfiguration1.class, MyConfiguration2.class,MyConfiguration3.class })
public class MySpringBootApplication {
public static void main(String[] args) {
SpringApplication.run(MySpringBootApplication.class, args);
}
}
```

Here MySpringBootApplication is Spring boot based Application however it does not go with component scan instead it looks for specific configuration classes and import them to set up the configuration.

Another Application may use the following configuration :

```
@Configuration
@ComponentScan(basePackages = "com.example.demo.components")
public class MySpringBootApplication2 {
public static void main(String[] args) {
SpringApplication.run(MySpringBootApplication2.class, args);
}
}
```

Explain how you will go about Security in Spring Boot Application?

With a Spring Boot Application, you can fallback to Spring Security. Include the Spring Security Boot Starter :

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
    <version>2.1.6.RELEASE</version>
</dependency>
You can go with HTTP basic or form login.
```

To update the username or password , override the following properties in application properties file :

Spring.security.user.name = username1

Spring.security.user.password = password1

To enable method level security, you can use @EnableGlobalMethodSecurity.

To disable default Security configuration in-built, you need to exclude Security Auto Configuration class as follows :

To disable default Security configuration in-built, you need to exclude Security Auto Configuration class as follows :

```
@SpringBootApplication(exclude = { SecurityAutoConfiguration.class })
public class SpringBootApplication1 {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootApplication1.class, args);
    }
}
```

This can be also achieved by adding following to properties file :

```
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.security.SecurityAutoConfiguration
```

@ConfigurationTo Override default Security you can implement WebSecurityConfigurerAdapter class ,

for example :

```
@EnableWebSecurity
public class BasicConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth)
      throws Exception {
        auth
          .inMemoryAuthentication()
          .withUser("username")
            .password("password1")
            .roles("GUEST")
            .and()
          .withUser("admin")
            .password("admin")
            .roles("GUEST", "ADMIN");
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        HTTP
          .authorizeRequests()
          .anyRequest()
          .authenticated()
          .and()
          .httpBasic();
    }
}
```

@EnableWebSecurity is optional if the default security configuration is disabled.

Oauth2 is commonly used for authorization. To integrated OAuth2:

- Add a starter for Oauth2
- Use @EnableAuthrizationServer
- Use @EnableResourceServer in an application where resource located
- On Client Side, use either of @EnableOAuth2Sso or @EnableOAuth2Client.

How to implement Spring web using Spring boot?

Web Application Convenience

Web Application Convenience

- Boot automatically configures
  - A DispatcherServlet & ContextLoaderListener
  - Spring MVC using same defaults as @EnableWebMvc
- Plus many useful extra features:
  - Static resources served from classpath
    - /static, /public, /resources or /META-INF/resources
  - Templates served from /templates
    - If Velocity, Freemarker, Thymeleaf, or Groovy on classpath
  - Provides default /error mapping
    - Easily overridden
  - Default MessageSource for I18N

What is YAML?

Yaml Ain't a Markup Language
– Recursive acronym
• Created in 2001
• Alternative to .properties files
– Allows hierarchical configuration
• Java parser for YAML is called SnakeYAML
– Must be in the classpath
– Provided by spring-boot-starters

• YAML is convenient for hierarchical configuration data
– Spring Boot properties are organized in groups
– Examples: server, database, etc

YAML for Properties

• Spring Boot support YAML for Properties

– An alternative to properties files

application.properties

```
database.host = localhost
database.user = admin
```

application.yml

```
database:
    host: localhost
    user: admin
```

What Is Spring Cloud?

Spring Cloud, in microservices, is a system that provides integration with external systems. It is a short-lived framework that builds an application, fast. Being associated with the finite amount of data processing, it plays a very important role in microservice architectures.

For typical use cases, Spring Cloud provides the out of the box experiences and a sets of extensive features mentioned below:

- Versioned and distributed configuration.
- Discovery of service registration.
- Service to service calls.
- Routing.
- Circuit breakers and load balancing.
- Cluster state and leadership election.
- Global locks and distributed messaging.

Why Would You Opt for Microservices Architecture?

This is a very common microservices interview question which you should be ready for! There are plenty of pros that are offered by a microservices architecture. Here are a few of them:

- Microservices can adapt easily to other frameworks or technologies.
- Failure of a single process does not affect the entire system.
- Provides support to big enterprises as well as small teams.
- Can be deployed independently and in relatively less time.

Can we disable the default web server in the Spring Boot application?

The major strong point in Spring is to provide flexibility to build your application loosely coupled. Spring provides features to disable the web server in a quick configuration. Yes, we can use the application.properties to configure the web application type, i.e. `spring.main.web-application-type=none`.

How Can You Set Up Service Discovery?

There are multiple ways to set up service discovery. I'll choose the one that I think to be most efficient, Eureka by Netflix. It is a hassle free procedure that does not weigh much on the application. Plus, it supports numerous types of web applications.

Eureka configuration involves two steps – client configuration and server configuration.
Client configuration can be done easily by using the property files. In the clas spath, Eureka searches for a eureka-client.properties file. It also searches for overrides caused by the environment in property files which are environment specific.

For server configuration, you have to configure the client first. Once that is done, the server fires up a client which is used to find other servers. The Eureka server, by default, uses the Client configuration to find the peer server.

Why Do People Hesitate to Use Microservices?

I have seen many devs fumble over this question. After all, they're getting asked this question when interviewing for a microservices architect role, so acknowledging its cons can be a little tricky. Here are some good answers:

- **They require heavy investment** – Microservices demand a great deal of collaboration. Since your teams are working independently, they should be able to synchronize well at all times.
- **They need heavy architecture set up** – The system is distributed, the architecture is heavily involved.
- **They need excessive planning for handling operations overhead** – You need to be ready for operations overhead if you are planning to use a microservices architecture.
- **They have autonomous staff selection** – Skilled professionals are needed who can support microservices that are distributed heterogeneously.

List down the advantages of Microservices Architecture.

Advantage	Description
Independent Development	All microservices can be easily developed based on their individual functionality
Independent Deployment	Based on their services, they can be individually deployed in any application
Fault Isolation	Even if one service of the application does not work, the system still continues to function
Mixed Technology Stack	Different languages and technologies can be used to build different services of the same application
Granular Scaling	Individual components can scale as per need, there is no need to scale all components together

Advantages of Microservices Architecture

What do you know about Microservices?

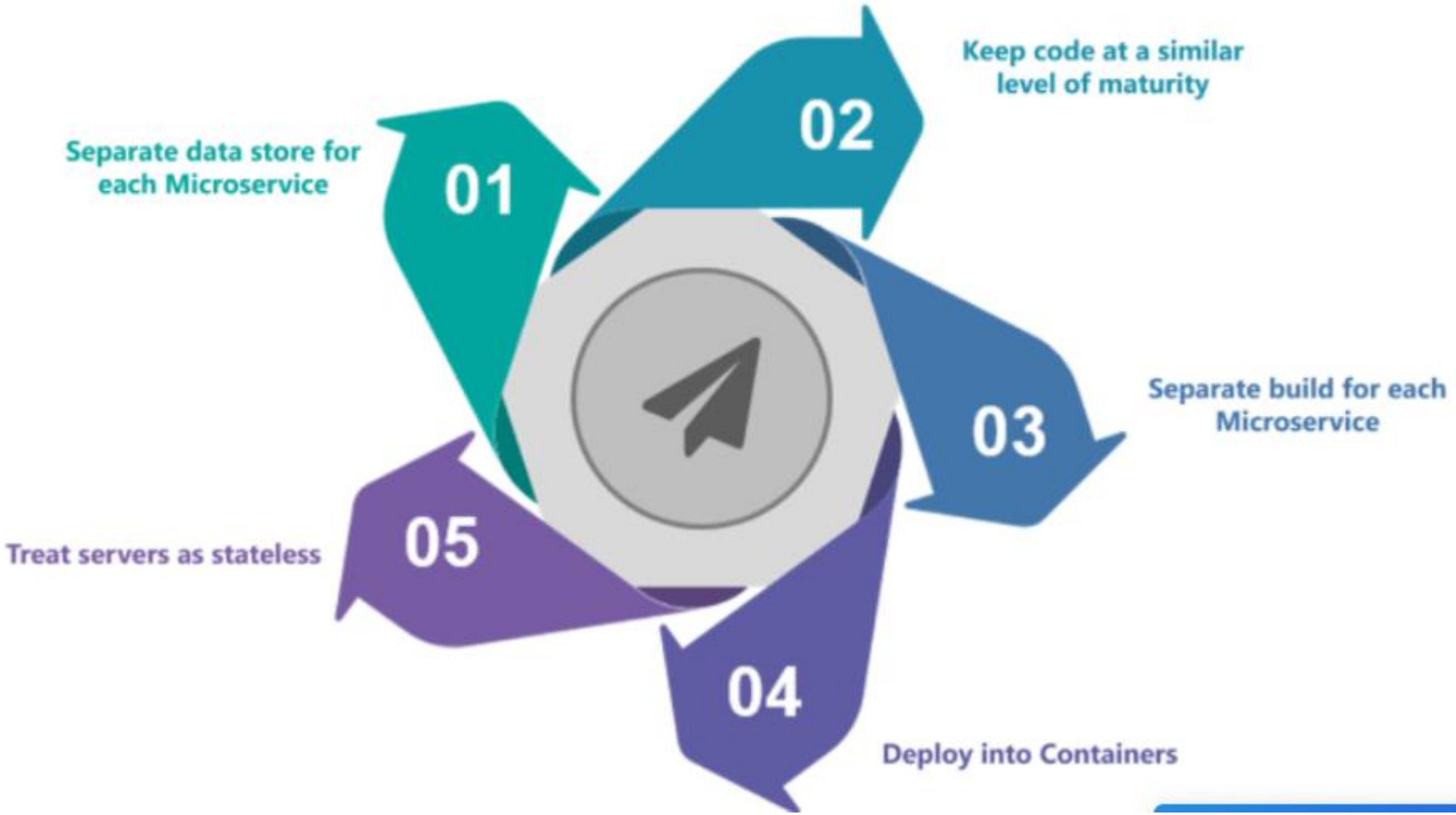
- **Microservices**, aka ***Microservice Architecture***, is an architectural style that structures an application as a collection of small autonomous services, modeled around a **business domain.**
- In layman terms, you must have seen how bees build their honeycomb by aligning hexagonal wax cells.
- They initially start with a small section using various materials and continue to build a large beehive out of it.
- These cells form a pattern resulting in a strong structure which holds together a particular section of the beehive.
- Here, each cell is independent of the other but it is also correlated with the other cells.
- This means that damage to one cell does not damage the other cells, so, bees can reconstruct these cells without impacting the complete beehive.

What are the features of Microservices?

- **Decoupling** – Services within a system are largely decoupled. So the application as a whole can be easily built, altered, and scaled
- **Componentization** – Microservices are treated as independent components that can be easily replaced and upgraded
- **Business Capabilities** – Microservices are very simple and focus on a single capability
- **Autonomy** – Developers and teams can work independently of each other, thus increasing speed
- **Continous Delivery** – Allows frequent releases of software, through systematic automation of software creation, testing, and approval
- **Responsibility** – Microservices do not focus on applications as projects. Instead, they treat applications as products for which they are responsible
- **Decentralized Governance** – The focus is on using the right tool for the right job. That means there is no standardized pattern or any technology pattern. Developers have the freedom to choose the best useful tools to solve their problems
- **Agility** – Microservices support agile development. Any new feature can be quickly developed and discarded again

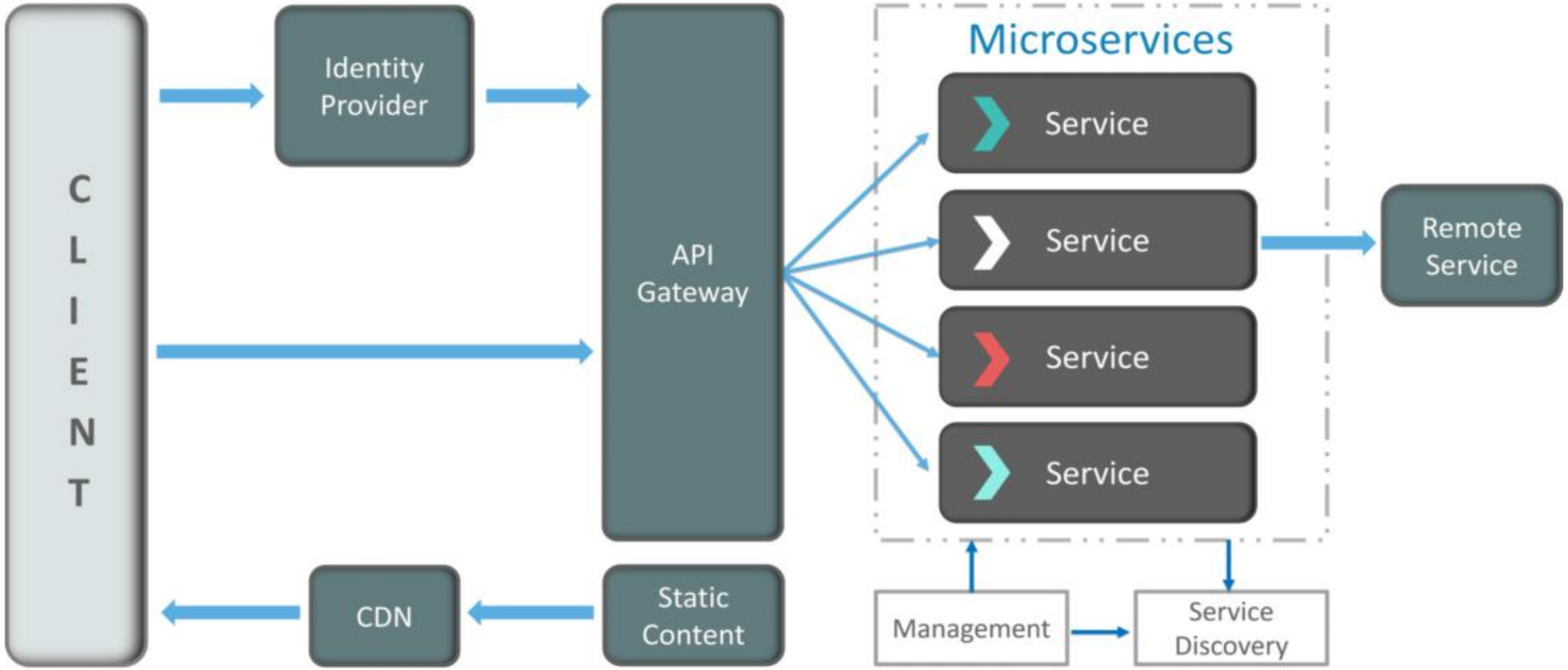
What are the best practices to design Microservices?

The following are the best practices to design microservices:



How does Microservice Architecture work?

A microservice architecture has the following components:



- **Clients** – Different users from various devices send requests.
- **Identity Providers** – Authenticates user or clients identities and issues security tokens.
- **API Gateway** – Handles client requests.
- **Static Content** – Houses all the content of the system.
- **Management** – Balances services on nodes and identifies failures.
- **Service Discovery** – A guide to find the route of communication between microservices.
- **Content Delivery Networks** – Distributed network of proxy servers and their data centers.
- **Remote Service** – Enables the remote access information that resides on a network of IT devices.

What is Cohesion?

The degree to which the elements inside a module belong together is said to be **cohesion**.

What is Coupling?

The measure of the strength of the dependencies between components is said to be **coupling**. A good design is always said to have **High Cohesion** and **Low Coupling**.

What are the pros and cons of Microservice Architecture?

Pros of Microservice Architecture	Cons of Microservice Architecture
Freedom to use different technologies	Increases troubleshooting challenges
Each microservices focuses on single capability	Increases delay due to remote calls
Supports individual deployable units	Increased efforts for configuration and other operations
Allow frequent software releases	Difficult to maintain transaction safety
Ensures security of each service	Tough to track data across various boundaries
Mulitple services are parallelly developed and deployed	Difficult to code between services

What are the challenges you face while working Microservice Architectures?

Developing a number of smaller microservices sounds easy, but the challenges often faced while developing them are as follows.

- **Automate the Components**: Difficult to automate because there are a number of smaller components. So for each component, we have to follow the stages of Build, Deploy and, Monitor.
- **Perceptibility**: Maintaining a large number of components together becomes difficult to deploy, maintain, monitor and identify problems. It requires great perceptibility around all the components.
- **Configuration Management**: Maintaining the configurations for the components across the various environments becomes tough sometimes.
- **Debugging**: Difficult to find out each and every service for an error. It is essential to maintain centralized logging and dashboards to debug problems.

Disadvantages of Monolithic Architecture

1. **Flexibility:** Monolithic architecture is not flexible. We can’t use different technologies. The technology stack is decided at the start and followed throughout. Once development matures, sometimes it becomes difficult to upgrade the technology stack versions, let alone incrementally adopt a newer technology.
2. **Reliability:** It's not reliable. If one feature goes down, the entire application might go down.
3. **Development speed:** Development is really slow in monolithic architecture. It's difficult for new team members to understand and modify the code of a large monolithic application. Code quality declines over time. With the increasing size of the code base, the IDE is overloaded and becomes slower. The larger the application, the longer it takes to start up. All these factors have a huge impact on the developers' productivity.
4. **Building complex applications:** It's difficult to build a complex application because of the limitations in terms of technologies.

5. **Scalability:** Monolithic applications are difficult to scale up once they get larger. We can create new instances of the monolith and ask the load balancer to distribute traffic to the new instances, but monolithic architecture can't scale with an increasing load. Each copy of the application instance will access all of the data, which makes caching less effective and increases memory consumption and I/O traffic. Also, different application components have different resource requirements — one might be CPU intensive while another might memory intensive. With monolithic architecture, we cannot scale each component independently.
6. **Continuous deployment:** Continuous deployment is extremely difficult. Large monolithic applications are actually an obstacle to frequent deployments. In order to update one component, we have to redeploy the entire application.

Benefits of Microservices Framework:

- **Agility:** By breaking down functionality to the most basic level and then abstracting the related services, you only need to modify and redeploy when a change is needed in the entire application.
- **Efficiency:** Leveraging a microservices-based architecture enables far more efficient usage of code and underlying infrastructure, with an ability to independently develop and deploy services without waiting on decisions regarding the entire application.
- **Resiliency:** By dispersing functionality across multiple services eliminates application susceptibility to a single point of failure. If one of the services fails, the others will continue to work.
- **Revenue:** Faster iterations and decreased downtime increase revenue.
- **Retention:** Continuous improvement offers lead to increased user engagement and retention.

Characteristics of Microservice Architecture

- In a microservices architecture, services are small, independent, and loosely coupled.
- Each service is a separate codebase, which can be managed by a small development team.
- Services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.
- Services are responsible for persisting their own data or external state. This differs from the traditional model, where a separate data layer handles data persistence.
- Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.
- Services don't need to share the same technology stack, libraries, or frameworks.

How Microservice Architecture Tackles the Drawbacks of Monolithic Architecture

1. **Flexibility:** Microservices architecture is quite flexible. Different microservices can be developed in different technologies. Since a microservice is smaller, the code base is quite less, so it's not that difficult to upgrade the technology stack versions. Also, we can incrementally adopt a newer technology without much difficulty.
2. **Reliability:** Microservices architecture can be very reliable. If one feature goes down, the entire application doesn't go down. We can fix the issue in the corresponding microservice and immediately deploy it.
3. **Development speed:** Development is pretty fast in microservices architecture. Since the volume of code is much less for a microservice, it's not difficult for new team members to understand and modify the code. They become productive right from the start. Code quality is maintained well. The IDE is much faster. A microservice takes much less time to start up. All these factors considerably increase developers' productivity.

4. **Building complex applications:** With microservice architecture, it's easy to build complex applications. If the features of the application are analyzed properly, we can break it down into independent components which can be deployed independently. Then, even the independent components can be further broken down into small independent tasks which can be deployed independently as a microservice. Deciding the boundaries of a microservice can be quite challenging. It’s actually an evolutionary process, but once we decide on a microservice, it’s easy to develop, as there are no limitation in technologies.
5. **Scalability:** Scalability is a major advantage in microservice architecture. Each microservice can be scaled individually. Since individual microservices are much smaller in size, caching becomes very effective.
6. **Continuous deployment:** Continuous deployment becomes easier. In order to update one component, we have to redeploy only that particular microservice.

What are the key differences between SOA and Microservices Architecture?

The key differences between SOA and microservices are as follows:

SOA	Microservices
Follows "**share-as-much-as-possible**" architecture approach	Follows "**share-as-little-as-possible**" architecture approach
Importance is on **business functionality** reuse	Importance is on the concept of "**bounded context**"
They have **common governance** and standards	They focus on **people collaboration** and freedom of other options
Uses **Enterprise Service bus (ESB)** for communication	Simple messaging system
They support **multiple message protocols**	They use **lightweight protocols** such as **HTTP/REST** etc.
Multi-threaded with more overheads to handle I/O	**Single-threaded** usually with the use of Event Loop features for non-locking I/O handling

SOA	Microservices
Maximizes application service reusability	Focuses on **decoupling**
Traditional Relational Databases are more often used	**Modern Relational Databases** are more often used
A systematic change requires modifying the monolith	A systematic change is to create a new service
DevOps / Continuous Delivery is becoming popular, but not yet mainstream	Strong focus on DevOps / Continuous Delivery

	Microservices	SOA	Monolithic
Design	Services are built in small units and expressed formally with business-oriented APIs.	Services can range in size anywhere from small application services to very large enterprise services including much more business functionality.	Monolithic applications evolve into huge size, a situation where understanding the entirety of the application is difficult.
Usability	Services exposed with a standard protocol, such as a RESTful API, and consumed/reused by other services and applications.	Services exposed with a standard protocol, such as SOAP and consumed/reused by other services – leverage messaging middleware.	Limited re-use is realized across monolithic applications.

Scalability	Services exist as independent deployment artifacts and can be scaled independently of other services.	Dependencies between services and reusable sub-components can introduce scaling challenges.	Scaling monolithic applications can often be a challenge.
Agility	Smaller independent deployable units ease build/release management, thereby high operational agility.	Enhances components sharing that increases dependencies and limits management capabilities.	Difficult to achieve operational agility in the repeated deployment of monolithic application artifacts.
Developme nt	Developing services discretely allows developers to use the appropriate development framework for the task at hand.	Reusable components and standard practices helps developers with implementation.	Monolithic applications are implemented using a single development stack (i.e., JEE or .NET), which can limit the availability of “the right tool for the job”.

What problems are solved by Spring Cloud?

While developing distributed microservices with Spring Boot we face few issues which 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.

Can you give a gist about Rest and Microservices?

REST

Though you can implement microservices in multiple ways, REST over HTTP is a way to implement Microservices. REST is also used in other applications such as web apps, API design, and MVC applications to serve business data.

Microservices

Microservices is an architecture wherein all the components of the system are put into individual components, which can be built, deployed, and scaled individually. There are certain principles and best practices of Microservices that help in building a resilient application.

In a nutshell, you can say that REST is a medium to build Microservices.

What do you understand by Distributed Transaction?

Distributed Transaction is any situation where a single event results in the mutation of two or more separate sources of data which cannot be committed atomically. In the world of microservices, it becomes even more complex as each service is a unit of work and most of the time multiple services have to work together to make a business successful.

What is an Idempotence and where it is used?

Idempotence is the property of being able to do something twice in such a way that the end result will remain the same i.e. as if it had been done once only.

Usage: Idempotence is used at the remote service, or data source so that, when it receives the instruction more than once, it only processes the instruction once.

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!**

Question: How do you implement a Spring Security in a Spring Boot Application?

Answer:

- 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.

Question: How will you Configure Spring Boot Application Login?

Answer: Spring Boot Application Login can be configured by specifying the logging.level in the application. Properties file. It is usually pre-configured as console output.

Question: What are the fundamental characteristics of a Microservices Design?

Answer:

- Services split up and organized around business functionality.
- Separate modules handled and owned by different development teams.
- Decentralized Framework.
- Maintenance of respective modules by respective development teams.
- Separate modules may be maintained by different databases.
- Modules in a Microservice Architecture are separately deployable. They can be updated, enhanced, or deleted without disrupting the entire architecture.
- Real-time monitoring of the application.

Question: What are the main challenges in Microservice Deployment?

Answer: The challenges in Microservice can be both technical as well as functional.

From the point of business, the main challenges are:

- Require heavy investment
- Heavy Infrastructure Setup
- Excessive Planning for managing operations overhead
- Staff Selection and maintenance.

From a technical standpoint –

- Communication between different microservices in the application.
- Component automation
- Application maintenance
- Configuration Management
- Heavy Operations Overhead
- Deployment Challenges
- Testing and Debugging Challenges

Question: What are the advantages and disadvantages of Microservices?

Advantages:

- Improved Scalability
- Fault Isolation
- Localized Complexity
- Increased Agility
- Simplified Debugging & Maintenance
- Better correspondence of developers with business users.
- Smaller development teams
- Better scope for technology upgradation.

Disadvantages:

- Complicated as a whole.
- Requires accurate pre-planning
- Modular dependencies are hard to calculate.
- Less control over third party applications
- Modular Interdependencies are challenging to track.
- More opportunities for malicious intrusions.
- Complete end-to-end testing is difficult.
- Deployment Challenges.

Question: What is OAuth?

Answer: 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 logon credential. This is also known as secure, third-party, user-agent, delegated authorization.

Question: What are the steps in an End-to-End Microservices Testing?

Answer: 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

Question: How does Docker help in Microservices?

Answer: 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.

Question: What is a Client Certificate?

Answer: A Client Certificate is a digital certificate that is used by client systems to make authenticated requests to a remote server. It plays a key role in many mutual authentication designs, providing strong assurances of a requester's identity.

Question: What is the role of RESTful APIs in Microservices?

Answer: A microservice is based on the concept where all it's component services require to interact with one another to complete the business functionality. This requires each microservice to have an interface. RESTful APIs provide a logical model for building these interfaces. It is based on the open networking principles of the Web. Thus, it serves as the most critical enabler of microservices.

Question: What is Tasklet, and what is a Chunk?

Answer: The Tasklet is a simple interface with one method to execute. A tasklet can be used to perform single tasks like running queries, deleting files, etc. In Spring Batch, the tasklet is an interface that can be used to perform unique tasks like clean or set up resources before or after any step execution.

Spring Batch uses a 'Chunk Oriented' processing style within its most common implementation. Chunk Oriented Processing refers to reading the data one at a time and creating chunks that will be written out, within a transaction boundary.

Question: What is Eureka in Microservices?

Answer: Eureka is alternatively known as the Netflix Service Discovery Server. It uses Spring Cloud and is not heavy on the application development process.

Question: How will you balance the server-side load by utilizing Spring Cloud?

Answer: The server-side load balancing can be done by using Netflix Zuul. It is also known as a JVM based router.

Question: When will you see fit to use the Netflix Hystrix?

Answer: Hystrix is an error tolerance and latency library. Hystrix mainly isolates the access points. It also makes sure that all 3rd Party libraries and services are restricted. Thus, we can use Hystrix to ensure that an application runs efficiently and avoids the kind of failures that occur in distributed systems.

Question: What is Spring Batch Framework?

Answer: Spring Batch is an open-source framework for batch processing – execution of a series of jobs. Spring Batch provides classes and APIs to read/write resources, transaction management, job processing statistics, job restart, and partitioning techniques to process high volume data.

Question: How can you access RESTful Microservices?

Answer: Considering the microservice architecture concept, each microservice needs to have an interface. Based on the principles of open networking of the Web, RESTful APIs provide the most logical model for building interfaces between the various components of the microservices architecture. RESTful APIs can be accessed in two ways:

- Using a REST Template that is load balanced.
- I am using multiple microservices.

Question: How do independent Microservices communicate with each other?

Answer: Microservices can communicate with each other through:

- HTTP for traditional Request-Response.
- Websockets for streaming.
- Brokers or Server Programs running Advanced Routing Algorithms.

Question: What are the tools that can be considered for managing a microservice architecture?

Answer: The main tools that can be used to build/manage a microservice architecture are:

1. **MongoDB:** It is a document-based open-source distributed database. Here data is stored in JSON format with a different structure for different documents. It also supports a lot of programming languages like C, C++, C#, PERL, PHP, Python, Java, Ruby, Scala, etc.
2. **Elasticsearch:** It is a full-text search engine.
3. **KAFKA:** It is an event queue system. All transactions are processed via the event queue, thus avoiding the web like random interactions between different services. Kafka renders a microservice architecture robust and clean.
4. **JENKINS:** It is an automation tool that enables Continuous Integration and Continuous Development. It supports many plugins and easily integrates with almost every tool.
5. **DOCKER:** 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.
6. **KUBERNETES:** With thousands of services running in an application, Kubernetes, as an engine orchestrates the entire process.

21. What Are The Fundamentals Of Microservices Design?

This is probably one of the most frequently asked microservices interview questions. Here is what you need to keep in mind while answering to it:

- Define a scope
- Combine loose coupling with high cohesion
- Create a unique service which will act as an identifying source, much like a unique key in a database table
- Creating the correct API and take special care during integration.
- Restrict access to data and limit it to the required level
- Restrict access to data and limit it to the required level
- Maintain a smooth flow between requests and response

- Automate most processes to reduce time complexity
- Keep the number of tables to a minimum level to reduce space complexity
- Monitor the architecture constantly and fix any flaw when detected.
- Data stores should be separated for each microservices.
- For each microservices, there should be an isolated build.
- Deploy microservices into containers.
- Servers should be treated as stateless.

Q: What are the advantages of using Spring Cloud ?

A: When developing distributed microservices with Spring Boot we face the following issues-

- **Complexity associated with distributed systems-**
  This overhead includes network issues, Latency overhead, Bandwidth issues, security issues.
- **Service Discovery-**
  Service discovery tools manage how processes and services in a cluster can find and talk to one another. It involves a directory of services, registering services in that directory, and then being able to lookup and connect to services in that directory.
- **Redundancy-**
  Redundancy issues in distributed systems.
- **Loadbalancing-**
  Load balancing improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives.
- **Performance issues-**
  Performance issues due to various operational overheads.
- **Deployment complexities-**
  Requirement of Devops skills.

Q: What are the different Microservices Design Patterns?

A: The different Microservices Design Patterns are -

- Aggregator Microservice Design Pattern
- API Gateway Design Pattern
- Chain of Responsibility Design Pattern
- Branch Microservice Design Pattern
- Circuit Breaker Design Pattern
- Asynchronous Messaging Design Pattern

How big a single microservice should be?

A good, albeit non-specific, rule of thumb is as small as possible but as big as necessary to represent the domain concept they own said by Martin Fowler

Size should not be a determining factor in microservices, instead bounded context principle and single responsibility principle should be used to isolate a business capability into a single microservice boundary.

Microservices are usually small but not all small services are microservices. If any service is not following the Bounded Context Principle, Single Responsibility Principle, etc. then it is not a microservice irrespective of its size. So the size is not the only eligibility criteria for a service to become microservice.

In fact, size of a microservice is largely dependent on the language (Java, Scala, PHP) you choose, as few languages are more verbose than others.

What should be preferred communication style in microservices: synchronous or asynchronous?

---

1. You must use asynchronous communication while handling HTTP POST/PUT (anything that modifies the data) requests, using some reliable queue mechanism (RabbitMQ, AMQP, etc.)
2. It's fine to use synchronous communication for Aggregation pattern at API Gateway Level. But this aggregation should not include any business logic other than aggregation. Data values must not be transformed at Aggregator, otherwise, it defeats the purpose of Bounded Context. In Asynchronous communication, events should be published into a Queue. Events contain data about the domain, it should not tell what to do (action) on this data.
3. If microservice to microservice communication still requires synchronous communication for GET operation, then seriously reconsider the partitioning of your microservices for bounded context, and create some tasks in backlog/technical debt.

. How to maintain ACID in microservice architecture?

ACID is an acronym for four primary attributes namely atomicity, consistency, isolation, and durability ensured by the database transaction manager.

- **Atomicity**

In a transaction involving two or more entities, either all of the records are committed or none are.

- **Consistency**

A database transaction must change affected data only in allowed ways following specific rules including constraints/triggers etc.

- **Isolation**

Any transaction in progress (not yet committed) must remain isolated from any other transaction.

- **Durability**

Committed records are saved by a database such that even in case of a failure or database restart, the data is available in its correct state.

In a distributed system involving multiple databases, we have two options to achieve ACID compliance:

1. One way to achieve ACID compliance is to use a two-phase commit (a.k.a 2PC), which ensures that all involved services must commit to transaction completion or all the transactions are rolled back.
2. Use eventual consistency, where multiple databases owned by different microservices become eventually consistent using asynchronous messaging using messaging protocol. Eventual consistency is a specific form of weak consistency.

2 Phase Commit should ideally be discouraged in microservices architecture due to its fragile and complex nature. We can achieve some level of ACID compliance in distributed systems through eventual consistency and that should be the right approach to do it.

The Twelve-Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

The Twelve-Factor App

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

@Bean -> Basic Spring bean We can use for normal business logic. Common independet functioality

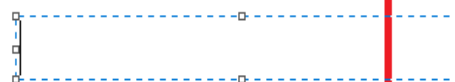
@Component -> Some thing like bean, but it can be applied with @configuration and Component scan.

@Configuration -> If we want to load IOC Container using Annotatin Approach , we can use @configigiuration

@Repository -> Sterotype -> Some basic functionalitities are attached -> Repostiory DAO, jdbc connection, closing the connection, managing conn pool

@Service -> Sterotype -> Distributed Transaction , AOP Middelware services

@RestController -> Entry point for all external front ends.



ALL THE BEST FOR YOUR FUTURE

KEEP IN TOUCH

vijumca@gmail.com
(FaceBook/LinkedIn)

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep

Spring Boot Interview Prep