Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.

We should use Spring Boot Framework because:

* The dependency injection approach is used in Spring Boot.
* It contains powerful database transaction management capabilities.
* It simplifies integration with other Java frameworks like JPA/Hibernate ORM, Struts, etc.
* It reduces the cost and development time of the application.
* ****Spring Data:**** It simplifies data access from the relational and ****NoSQL**** databases.
* ****Spring Batch:**** It provides powerful ****batch**** processing.
* ****Spring Security:**** It is a security framework that provides robust ****security**** to applications.
* ****Spring Social:**** It supports integration with ****social networking**** like LinkedIn.
* ****Spring Integration:**** It is an implementation of Enterprise Integration Patterns. It facilitates integration with other ****enterprise applications**** using lightweight messaging and declarative adapters.

**Advantages of Spring boot:**

1. Creates stand-alone application, which can be run with java -jar command.
2. Has the build in embedded web server Tomcat/Jetty
3. No XML configuration required.
4. Reduces the boiler plate code.
5. Has number of plug-ins
6. Increase production ready set-up or environment.

**Features:**

1. Web Application
2. Spring Application. (@Annotation in the main and .run static method)
3. Application events and Listeners.
4. Externalized configuration.
5. Properties files.
6. YAML support.
7. Logging (default, unless we want custom configuration)
8. Security.
9. Type-Safe configuration

**SPRING WEB APPLICATION:  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

If we want to develop a web application, we need to add the following dependency in pom.xml file:

**<dependency>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-web**</artifactId>**

**<version>**2.2.2.RELEASE**</version>**

**</dependency>**

The single spring-boot-starter-web dependency transitively pulls in all dependencies related to web development.

* org.springframework.boot:spring-boot-starter
* org.springframework.boot:spring-boot-starter-tomcat (core, el, logging, web-socket dependencies comes in tomcat transitively)
* org.springframework.boot:spring-boot-starter-validation
* com.fasterxml.jackson.core:jackson-databind
* org.springframework:spring-web
* org.springframework:spring-webmvc

The spring-boot-starter-web auto-configures the following things that are required for the web development:

* Dispatcher Servlet --> learn this
* Error Page
* Web JARs for managing the static dependencies
* Embedded servlet container

SPRING DATA JPA:  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**<dependency>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-data-jpa**</artifactId>**

**<version>**2.2.2.RELEASE**</version>**

**</dependency>**

Spring Data is a high-level Spring Source project. Its purpose is to unify and easy access to the different kinds of persistence stores, both relational database systems, and NoSQL data stores.

Spring Data JPA adds a layer on the top of JPA. It means, Spring Data JPA uses all features defined by JPA specification, especially the entity, association mappings, and JPA's query capabilities. Spring Data JPA adds its own features such as the no-code implementation of the repository pattern and the creation of database queries from the method name.

Spring Data JPA handles most of the complexity of JDBC-based database access and ORM (Object Relational Mapping). It reduces the boilerplate code required by JPA. It makes the implementation of your persistence layer easier and faster.

Spring internally generates a ****JPQL**** (Java Persistence Query Language) query based on the method name. The query is derived from the method signature. It sets the bind parameter value, execute the query, and returns the result.

Spring Data Repository:  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Spring Data JPA provides three repositories are as follows:

CrudRepository: It offers standard create, read, update, and delete It contains method like findOne(), findAll(), save(), delete(), etc.

PagingAndSortingRepository: It extends the CrudRepository and adds the findAll methods. It allows us to sort and retrieve the data in a paginated way.

JpaRepository: It is a JPA specific repository It is defined in Spring Data Jpa. It extends the both repository CrudRepository and PagingAndSortingRepository. It adds the JPA-specific methods, like flush() to trigger a flush on the persistence context.

Before JPA, ORM was the term more commonly used to refer to these frameworks. It is the reason Hibernate is called the ORM framework.

JPA allows us to map application classes to table in the database.

* ****Entity Manager:**** Once we define the mapping, it handles all the interactions with the database.
* ****JPQL (Java Persistence Query Language):**** It provides a way to write queries to execute searches against entities. It is different from the SQL queries. JPQL queries already understand the mapping that is defined between entities. We can add additional conditions if required.
* ****Criteria API:**** It defines a Java-based API to execute searches against the database.

Spring boot **Actuator:**  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

It includes a number of additional features that help us to monitor and manage the Spring Boot application. It contains the actuator endpoints (the place where the resources live). We can use ****HTTP**** and ****JMX**** endpoints to manage and monitor the Spring Boot application. If we want to get production-ready features in an application, we should use the S****pring Boot actuator.****

**<dependency>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-actuator**</artifactId>**

**<version>**2.2.2.RELEASE**</version>**

**</dependency>**

There are ****three**** main features of Spring Boot Actuator:

* ****Endpoints****
* ****Metrics****
* ****Audit****

****Endpoint:**** The actuator endpoints allows us to monitor and interact with the application. Spring Boot provides a number of built-in endpoints. We can also create our own endpoint. We can enable and disable each endpoint individually. Most of the application choose ****HTTP****, where the Id of the endpoint, along with the prefix of ****/actuator,****is mapped to a URL.

For example, the ****/health**** endpoint provides the basic health information of an application. The actuator, by default, mapped it to ****/actuator/health****.

****Metrics:**** Spring Boot Actuator provides dimensional metrics by integrating with the****micrometer****. The micrometer is integrated into Spring Boot. It is the instrumentation library powering the delivery of application metrics from Spring. It provides vendor-neutral interfaces for ****timers, gauges, counters, distribution summaries,**** and ****long task timers**** with a dimensional data model.

****Audit:**** Spring Boot provides a flexible audit framework that publishes events to an ****AuditEventRepository.**** It automatically publishes the authentication events if spring-security is in execution.

Spring boot starter test:  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The ****spring-boot-starter-test**** is the primary dependency for the test. It contains the majority of elements required for our tests.

There are several different types of tests that we can write to help test and automate the health of an application. Before starting any testing, we need to integrate the testing framework.

**<dependency>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-test**</artifactId>**

**<version>**2.2.2.RELEASE**</version>**

**<scope>**test**</scope>**

**</dependency>**

Spring Boot Packaging:  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In the J2EE application, modules are packed as ****JAR, WAR,**** and ****EAR****. It is the compressed file formats that is used in the J2EE. J2EE defines three types of archives:

**WAR:  
\*\*\*\*\*\***

stands for ****Web Archive.**** WAR file represents the web application. Web module contains servlet classes, JSP files, HTML files, JavaScripts, etc. are packaged as a JAR file with .****war**** extension. It contains a special directory called ****WEB-INF****.

WAR is a module that loads into a web container of the Java Application Server. The Java Application Server has ****two****containers: ****Web Container**** and ****EJB Container****.

The ****Web Container**** hosts the web applications based on Servlet API and JSP. The web container requires the web module to be packaged as a WAR file. It is a WAR file special JAR file that contains a ****web.xmlv**** file in the ****WEB-INF**** folder.

An ****EJB Container**** hosts Enterprise Java beans based on EJB API. It requires EJB modules to be packaged as a JAR file. It contains an ****ejb-jar.xml**** file in the ****META-INF**** folder.

**JAR:  
\*\*\*\*\***

stands for ****Java Archive.**** An EJB (Enterprise Java Beans) module that contains bean files (class files), a manifest, and EJB deployment descriptor (XML file) are packaged as JAR files with the extension .****jar.**** It is used by software developers to distribute Java classes and various metadata.

In other words, A file that encapsulates one or more Java classes, a manifest, and descriptor is called JAR file. It is the lowest level of the archive. It is used in J2EE for packaging EJB and client-side Java Applications. It makes the deployment easy.

**EAR:  
\*\*\*\***

stands for ****Enterprise Archive.**** EAR file represents the enterprise application. The above two files are packaged as a JAR file with the .****ear**** extension. It is deployed into the Application Server. It can contain multiple EJB modules (JAR) and Web modules (WAR). It is a special JAR that contains an ****application.xml**** file in the ****META-INF**** folder.

**Spring Boot Auto-configuration:**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Spring Boot auto-configuration automatically configures the Spring application based on the jar dependencies that we have added.

For example, if the H2 database Jar is present in the classpath and we have not configured any beans related to the database manually, the Spring Boot's auto-configuration feature automatically configures it in the project.

We can enable the auto-configuration feature by using the annotation @EnableAutoConfiguration. But this annotation does not use because it is wrapped inside the @SpringBootApplication annotation. The annotation @SpringBootApplication is the combination of three annotations: @ComponentScan, @EnableAutoConfiguration, and @Configuration. However, we use @SpringBootApplication annotation instead of using @EnableAutoConfiguration.

@SpringBootApplication=@ComponentScan+@EnableAutoConfiguration+@Configuration

Spring boot AOP: Aspect Oriented Programming  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

pattern that increases modularity by allowing the separation of the ****cross-cutting concern****. These cross-cutting concerns are different from the main business logic. We can add additional behavior to existing code without modification of the code itself.

**Aspect:** A code unit that encapsulates pointcuts, advices, and attributes.

**Point-cuts:** It defines the set of entry points in which advice is executed.

**Join-point:**

**Advice:** It is an implementation of cross-cutting concerns (Before Advice, After Advice, Around Advice, After Throwing, After Returning

**<dependency>**

**<groupId>**org.springframework.boot**</groupId>**

**<artifactId>**spring-boot-starter-aop**</artifactId>**

**</dependency>**

****Before Advice:**** An advice that executes before a join point, is called before advice. We use ****@Before**** annotation to mark an advice as Before advice.

****After Advice:**** An advice that executes after a join point, is called after advice. We use ****@After****annotation to mark an advice as After advice.

****Around Advice:**** An advice that executes before and after of a join point, is called around advice.

****After Throwing Advice:**** An advice that executes when a join point throws an exception.

****After Returning Advice:**** An advice that executes when a method executes successfully.