Spring Boot & Spring Cloud Microservices

Pre-requisites

1. Java Technology: OOPS & Collection Framework
2. Spring Framework: Dependency Injection, Spring MVC & Annotations

Spring Boot:  
It simplifies developing the spring applications by taking care of all the generic configurations

i.e,

- You don’t have to write any configuration file like xml files

- No need to setup because the server is added into the application itself

- Set up required for applications are provided by spring boot starter projects

Starter projects in Spring boot

* Spring Boot Starter Web: It automatically configures setup for web development like Front Controller, Server Configuration (Embedded servers), Component Scanning
* Spring Boot Starter JPA: It automatically configures the dependencies required for Databases like Connections, Connection Factories, Templates (JdbcTemplate & HiberanteTemplate)

For Spring Boot we need following software’s to be installed

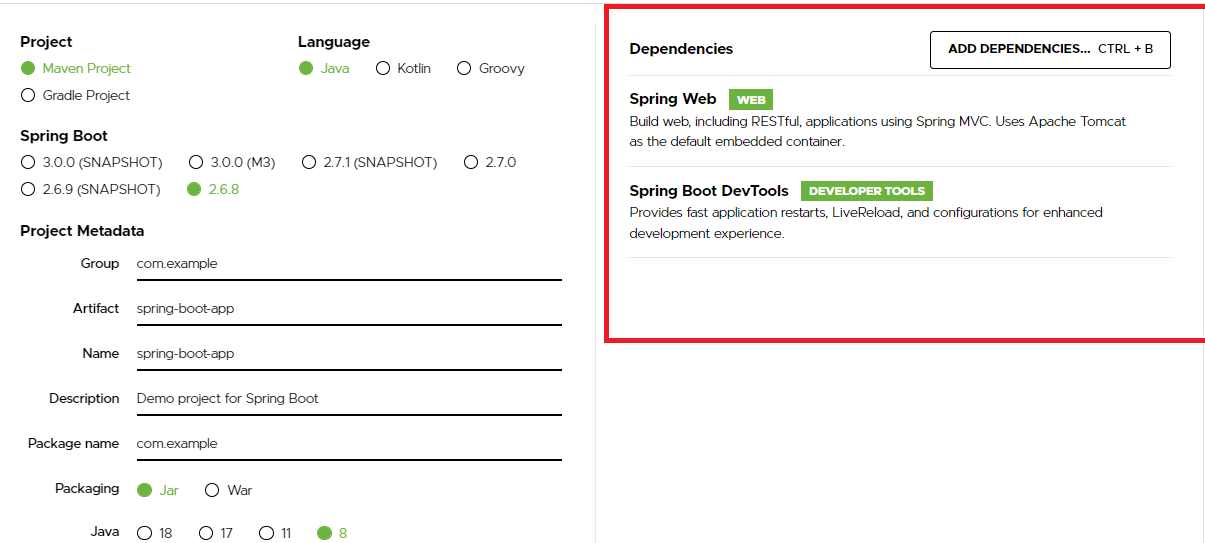
* Java
* Eclipse IDE / STS
* Postman App

Spring provides us a website to set up the spring boot project i.e., Spring Initializr

Creating our first web application in Spring Boot

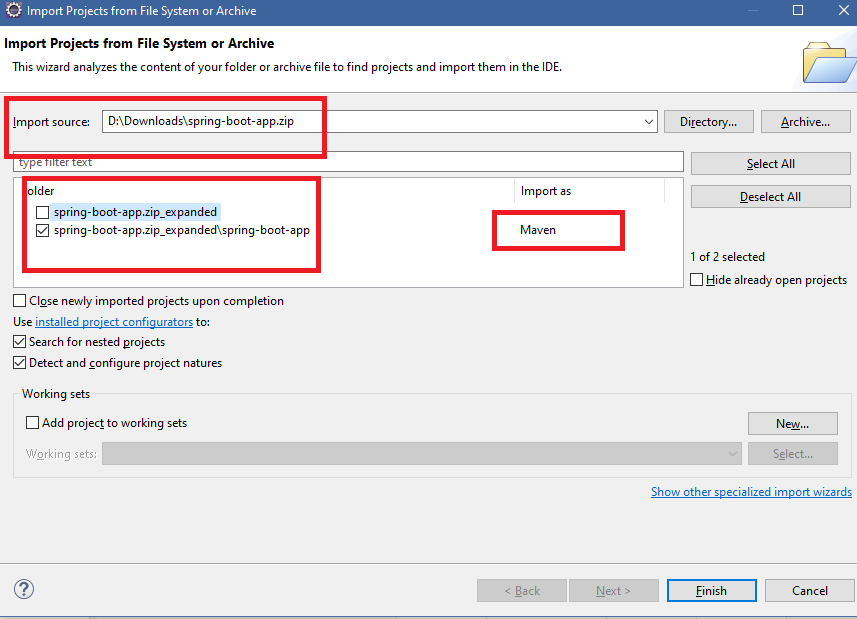
Dependencies required are:

* Web : Provides configurations for web applications like Front Controller, Server, Component Scanning – we get everything auto-configured for us
* Devtools (Optional) : auto-reload feature

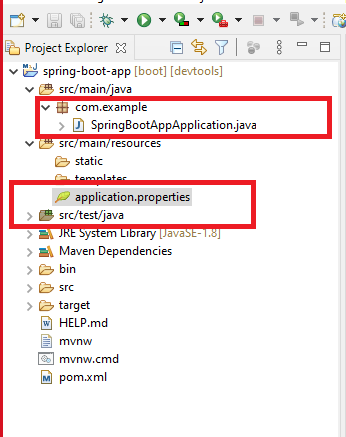


Click on Generate so that it downloads the zip file of the project, then you need to open this project from the Eclipse

Open project with Maven selected in the eclipse



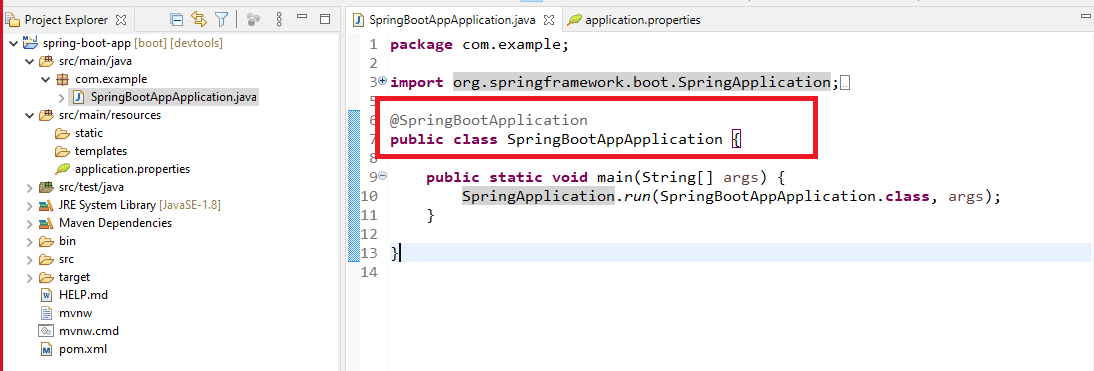
When we see the project structure we can observe two files



The Java file is the entry point file to launch your application, the application will be launched in the embedded server (Tomcat)

The application.properties is the configuration file for your application which keeps all the application related configurations like data-source informations, server port information’s, microservice related configurations

Note: Spring Boot uses application.properties as the default configuration file, but you can change this name also and load a different configuration file

The Java file looks like below  


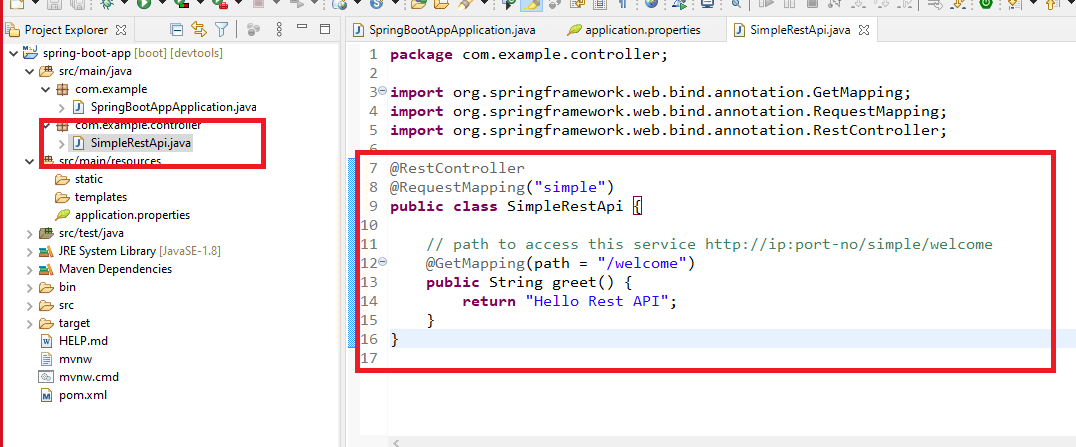
@SpringBootApplication:

* It takes care of doing all the auto-configurations for the application based on the library we add in our classpath
* The class having this needs to be loaded so that an application which is fully configured will be running for use.

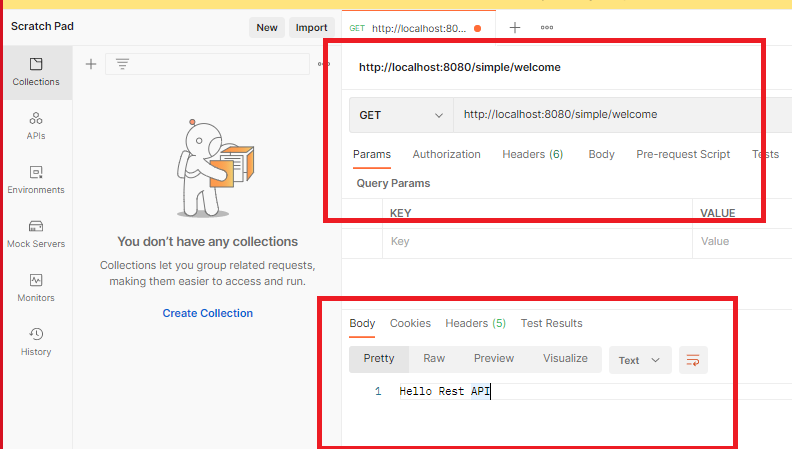
Since we have a web starter added we can create webservices in the application so that different applications can consume the data.

In Spring Framework we can create webservices using @RestController on top of the class

SimpleRest.java

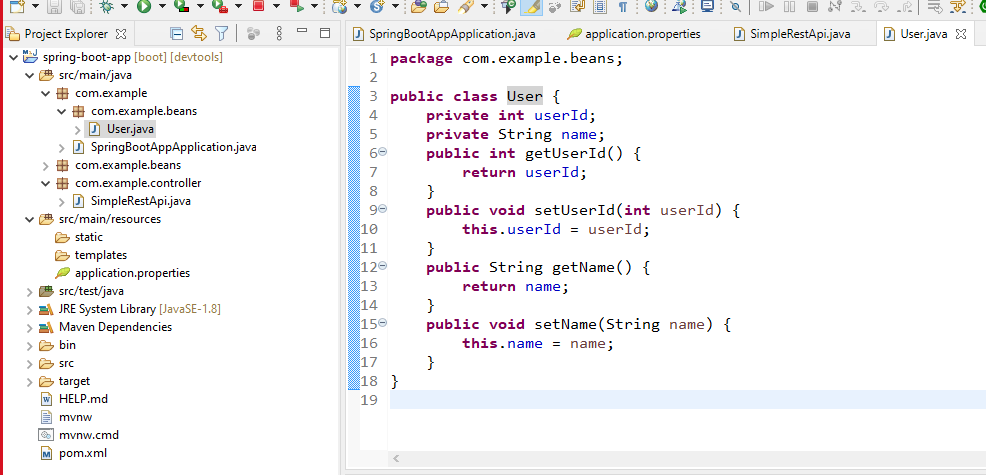


Now we are ready to run this application, since it is run in an embedded server the default port will be 8080 and the server is apache tomcat, but we can change all these configurations if required.

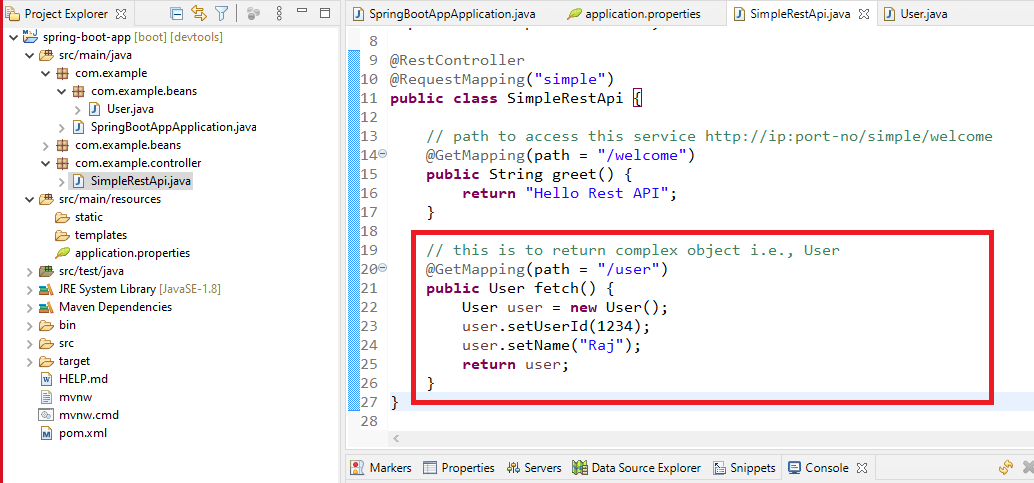


Here the response data is simple type hence it is coming in text format, however if the data is an object then it comes in JSON format.

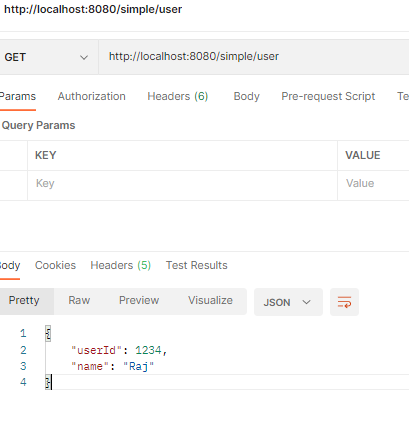
User.java



Now the Rest service can respond in json format if it returns user object



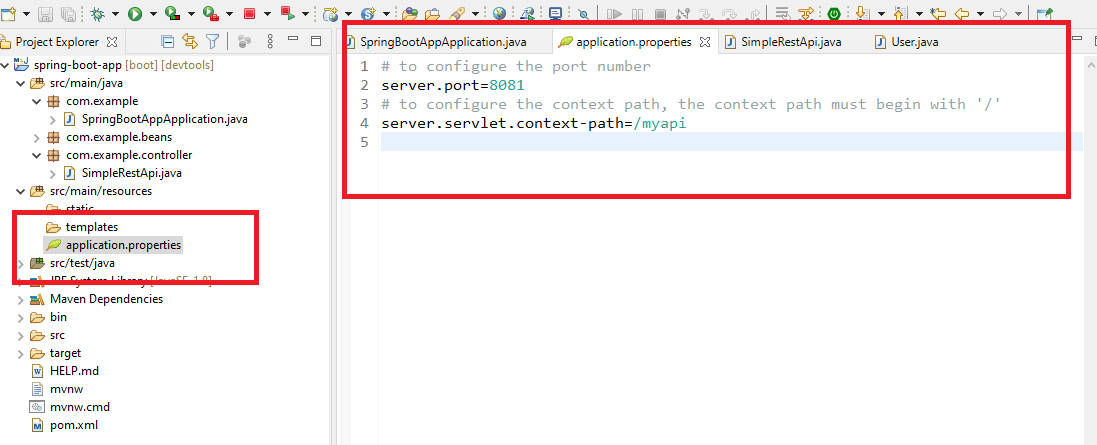
Output:



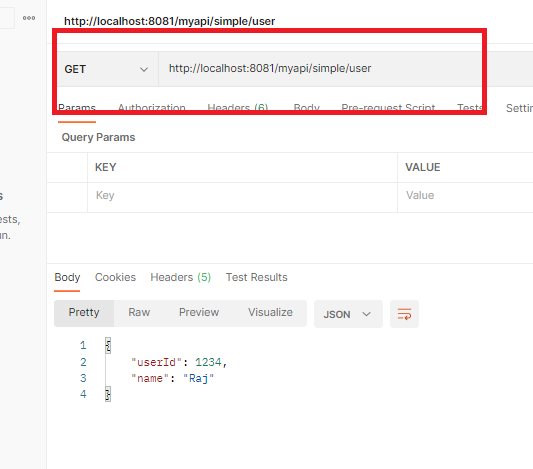
Configurations that needs to be done for the application

* How to add the context path to the application i.e., the root path to access the application
* How to configure the port number
* How to configure data-source information

All these configurations we need to do it in the application.properties

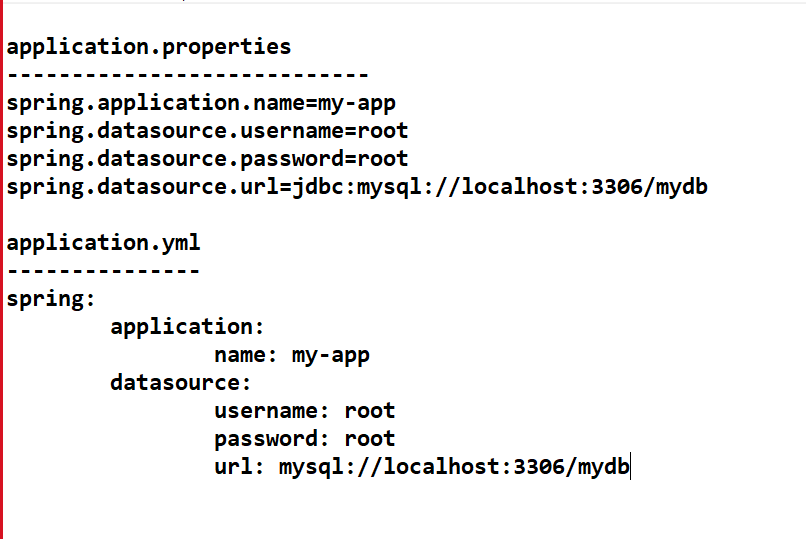


Now we must able to access the application using /myapi

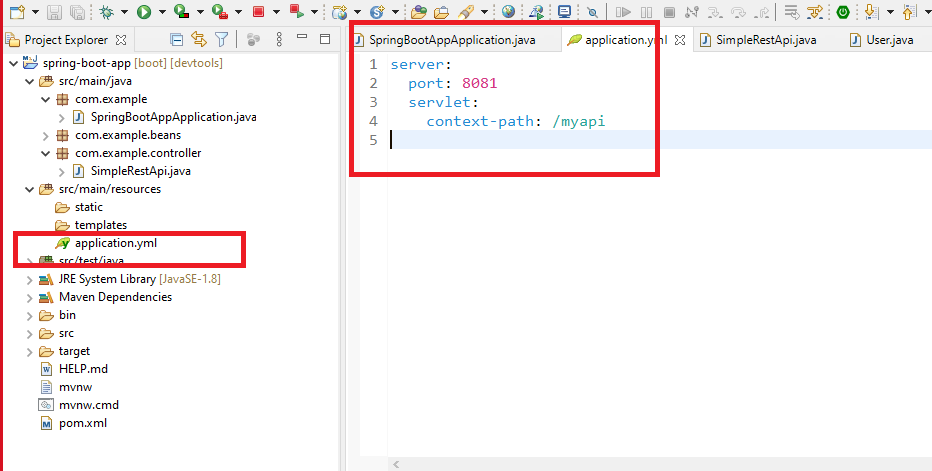


YAML Files:

These are alternate files for property files, they use indentation to mention the properties & sub-properties so that you can avoid writing property names repeatedly.



In Eclipse you will get plugin to convert properties to yaml files



Note: In Eclipse the plugin is taking care of providing the indentations to the existing properties, however if have yml already & want to add properties then we must take care of providing the indentation.

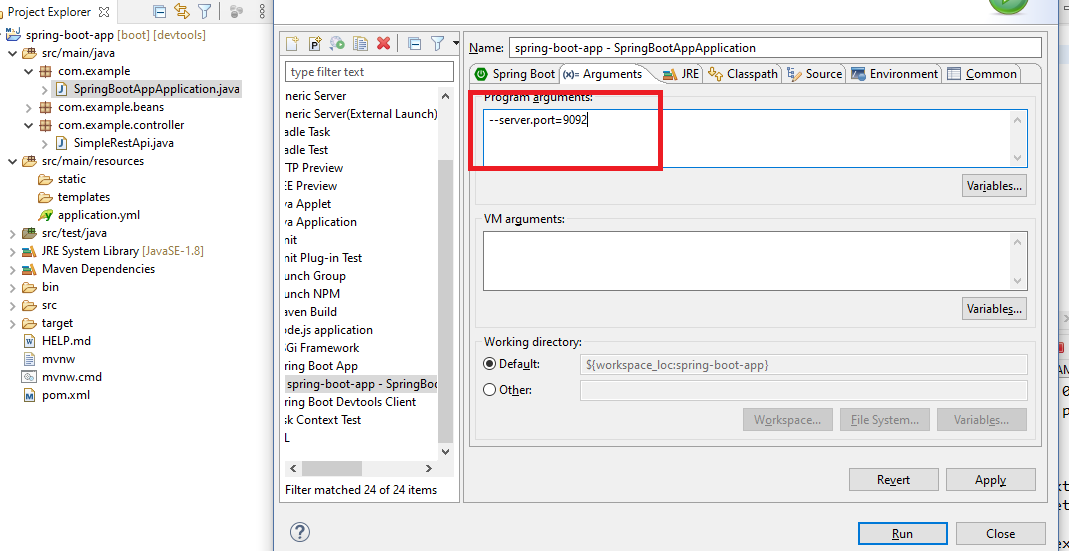
Overriding the properties

We can override the properties mention in the application.properties/yml while running the application.

* You can launch the application in eclipse and change the command line arguments
* If you have a deployable jar file then you can use command line arguments in the terminal

How to launch the application in eclipse by changing command line arguments

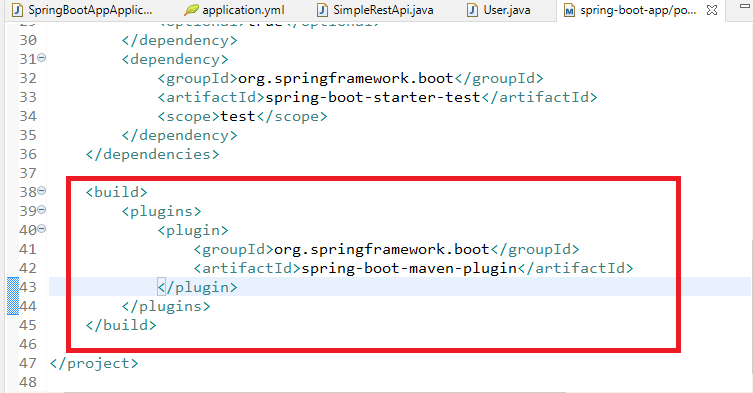
We need to use the properties with two hyphens followed by properties separated by dot



If in case you have a deployable jar then you can use the same arguments at the of time running

i.e., java -jar file-name.jar –server.port=9092

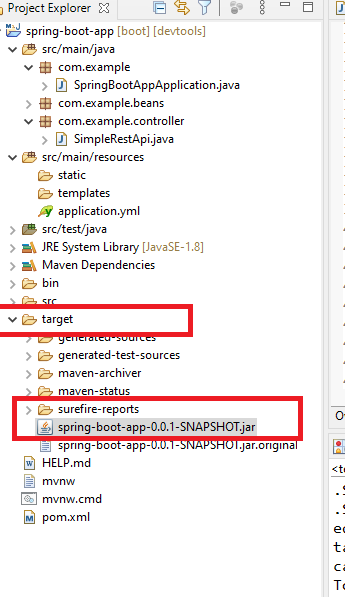
We can use maven built to build the deployable jar file, In Spring Boot its already configured to build a jar which will be an executable jar because of the entry in pom.xml



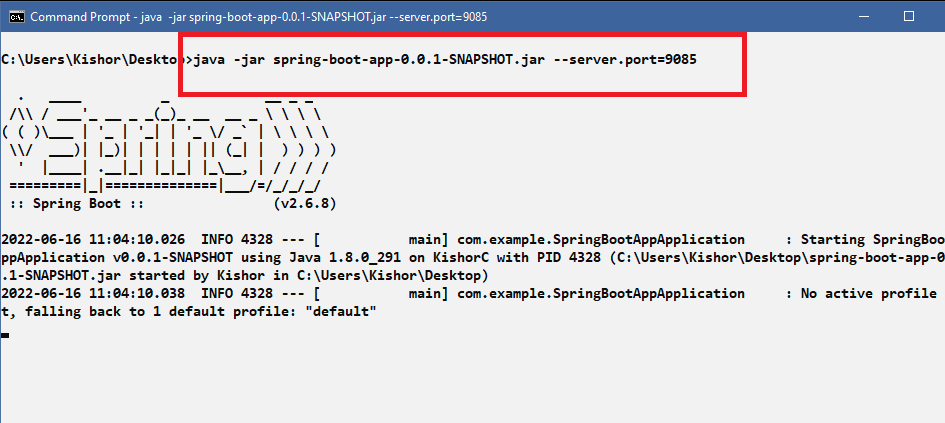
Note: This entry comes only we create project from spring initializr

How to build the jar from eclipse

Project -> Run As -> Maven Built -> Goals: package



Note: refresh the target folder to see the build file, we can run this jar using java command now



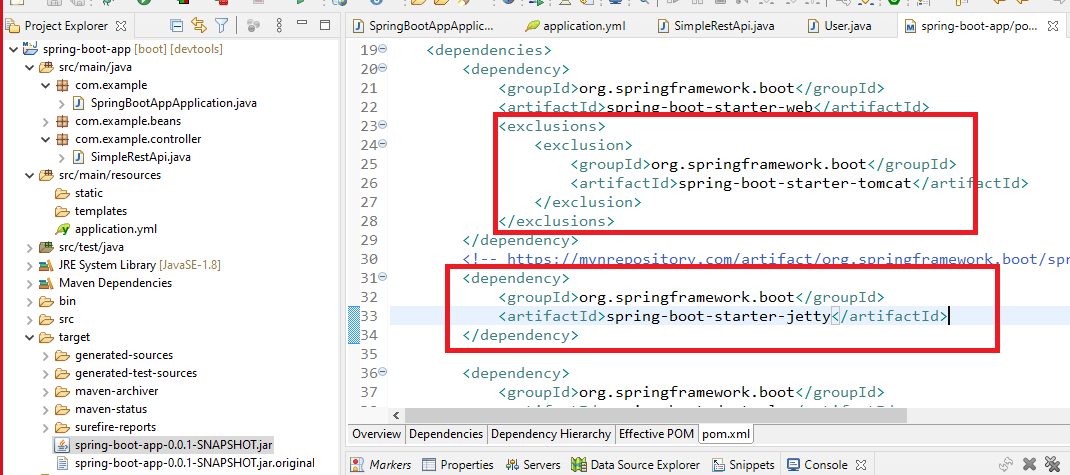
Spring boot provides following Embedded servers

1. Tomcat
2. Jboss – Undertow
3. Eclipse – Jetty

To configure different servers you need to exclude the default container tomcat and add different embedded server.

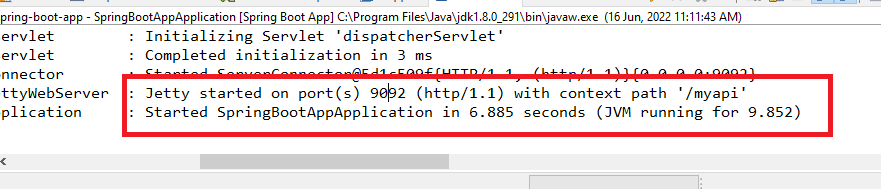
Steps

1. Exclude the embedded tomcat dependency
2. Add the other embed server you want to use



Output:

We must see in the console the jetty server has been used instead of tomcat



By default spring boot applications are packaged as jar, however if we want to deploy in our own external server then we need to package as war.