**End to End:**

**First Step:**

Create a sample spring boot project using **spring initializr(**<https://start.spring.io/>**)** and update with the help of Editor.

Annotations used:

**@SpringBoot Application** (For all Spring Boot applications, we have to give this annotation to make coding easy and also **used** to mark a configuration class that declares one or more **@Bean methods** and also triggers auto-configuration and component scanning)

* In Model class, I have used

**@Component** (Everything you see on the display in a **Java** application is a **component**. This includes things like windows, panels, buttons, checkboxes, scrollbars, lists, menus, and text fields. To be used, a **component** usually must be placed in a container)

In Controller class, I have used

**@RestController (**is used to create RESTful web services using Spring MVC. Spring Rest Controller takes care of mapping request data to the defined request handler method)

**@Autowired** (to define object)

**@GetMapping** (maps HTTP GET requests onto specific handler methods; we will create endpoints here)

**@RequestParam** (is used to read the form data and bind it automatically to the parameter present in the provided method, will pass parameters here)

**Second Step:**

**Maven:**

* After completion of coding, we have to use maven to build the code and run the JAVA application.
* We have different phases in maven lifecycle like
  + Clean
  + Validate
  + Compile
  + Test
  + Package
  + Verify
  + Site
  + Install
  + Deploy

Maven will perform all the phases in the life cycle based on the goal provided, and if we click on maven install, it will perform all the phases before installation.

* After maven install goal, the application code will be converted into jar file(.jar) and place the jar in local repository
* Once the Maven install goal implemented successfully, we need to test application running locally.
* The next step is, run the application with desired java arguments provided and Test locally(<http://localhost:8080/endpoint>)

**Third Step:**

**GitHub:**

Once we tested successfully in local, we will push our code into GitHub repository with the help of git commands in Git Bash.

**Git Commands used**:

* git status – to check the status
* git diff – to see diff before adding
* git add- to add (it will add to tracked files)
* git commit -m “msg” (to commit)
* git remote -v (to see the remote connections)
* git commit –amend (to change your latest log message)
* git rebase (to combine commits and modify history of a branch)

**.gitignore: gitignore file** is a text **file** that tells Git which **files** or folders to ignore in a project.

We have 3 different stages in git:

* tracked - a file which has been previously staged or committed;
* untracked - a file which *has not* been staged or committed; or
* ignored - a file which Git has been explicitly told to ignore.

After all changes, we will push our code into master branch with the command **git push origin master**, it will push code into master branch, If we want to make any changes in our code, we need to create a sub branch that will not affect our main application code in master branch.

If we want to merge our code from subbranch to master branch, we need to raise a pull request for merging the code into master branch.

**Fourth Step:**

**Jenkins:**

* We have to install Jenkins in our local system first(<https://www.jenkins.io/download/>).
* Test locally, if it is running or not(localhost:8080), create user id and passwd.
* Once Jenkins setup is ready, we will integrate GitHub with Jenkins.
* Whenever, a new change happens in the code, a new job will trigger in Jenkins. For that integration, we need to add webhooks in GitHub and also add GitHub URL in Jenkins job.
* If we don’t want that integration, there is another method called Jenkins file.
* In the application code, we need to add Jenkins file using declarative syntax pipeline.
* We have different stages in declarative syntax pipeline. In that syntax pipeline, we need to give maven deploy command (mvn deploy) maven deploy command will deploy the jar file into nexus.

**Fifth Step:**

**Nexus:**

* Install nexus (<https://www.sonatype.com/download-oss-sonatype>) give email and details to login and install the nexus.
* Go to nexus directory in your local machine (nexus 🡪 nexus version 🡪 bin 🡪 nexus 🡪 rt.click open with terminal) then the terminal will pop like this /Users/<username>/Documents/nexus/nexus-3.24.0-02/bin/nexus {start|stop|run|run-redirect|status|restart|force-reload}.
* By using this command, you can start, stop the nexus server and check in local host( <http://localhost:8081/>)
* Once we login into nexus, we can see our maven-snapshots in the browse section. Whenever we deploy in the Jenkins using mvn deploy command, that snapshot-jar file will deploy into nexus.
* So, like these, we push our jar files to nexus repository. Whenever we need those jar files, we can download those jar files from the nexus repository.

**Sixth Step:**

**Docker:**

* Download docker for desktop from docker-hub(<https://hub.docker.com/editions/community/docker-ce-desktop-mac/>)
* Once we installed docker, check the docker version.
* First, we need to create docker file using the downloaded jar file.

**Sample docker file**

FROM image name

COPY . /app

RUN make /app

CMD python /app/app.py

Each instruction creates one layer:

* **FROM**🡪creates a layer from the downloaded Docker image.
* **COPY**🡪 adds files from your Docker client’s current directory.
* **RUN**🡪 builds your application with make.
* **CMD**🡪 specifies what command to run within the container.
* After creating docker file, run that file to build an image.
* Command to run image,
  + docker build -t <docker-image-name>:<version> (build an image)
* docker image ls (list the images)
* Once we build an image, we will push into docker registry (docker hub).
* If we want to run that image as container, the command we used is
  + docker run –name <app-name> -p <port number> <image-name>:<version>. (run container)
* This is all manual process to run application, if we want to do automatically integrate with OpenShift.
* OpenShift will pull the image from registry and run a container.

**These commands are used to push docker image into docker registry.**

* docker login -u “Username” -p “Password” <registry-url>
* docker tag <image-name>:<version> repo/<image-name>:<version>
* docker push <image-name>:<version> repo/<image-name>:<version
* This command is used to pull image from docker hub,
  + docker pull <image-name>:<version>

**Run images:**

* docker run –name <app-name> -p <port number> <image-name>:<version>. (run container)
* docker container stop app-name (stop the container)
* docker container kill app-name (kill container)
* docker network ls (list the networks)
* docker container ls (list the running containers)
* docker container rm -f $(docker ps -aq) (delete all running and stopped containers)
* docker container logs <app-name> (print the logs)