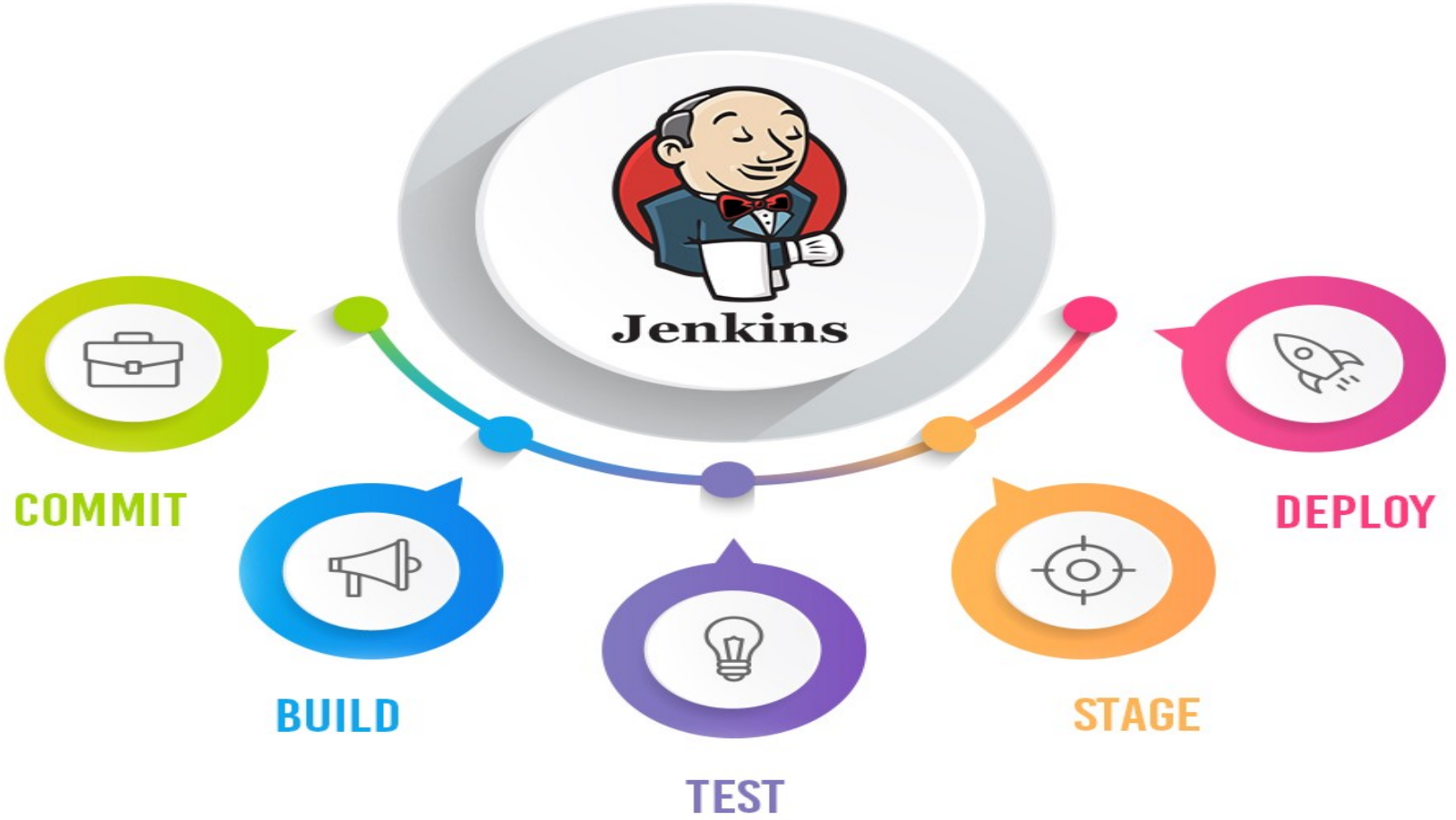


COMP 3104

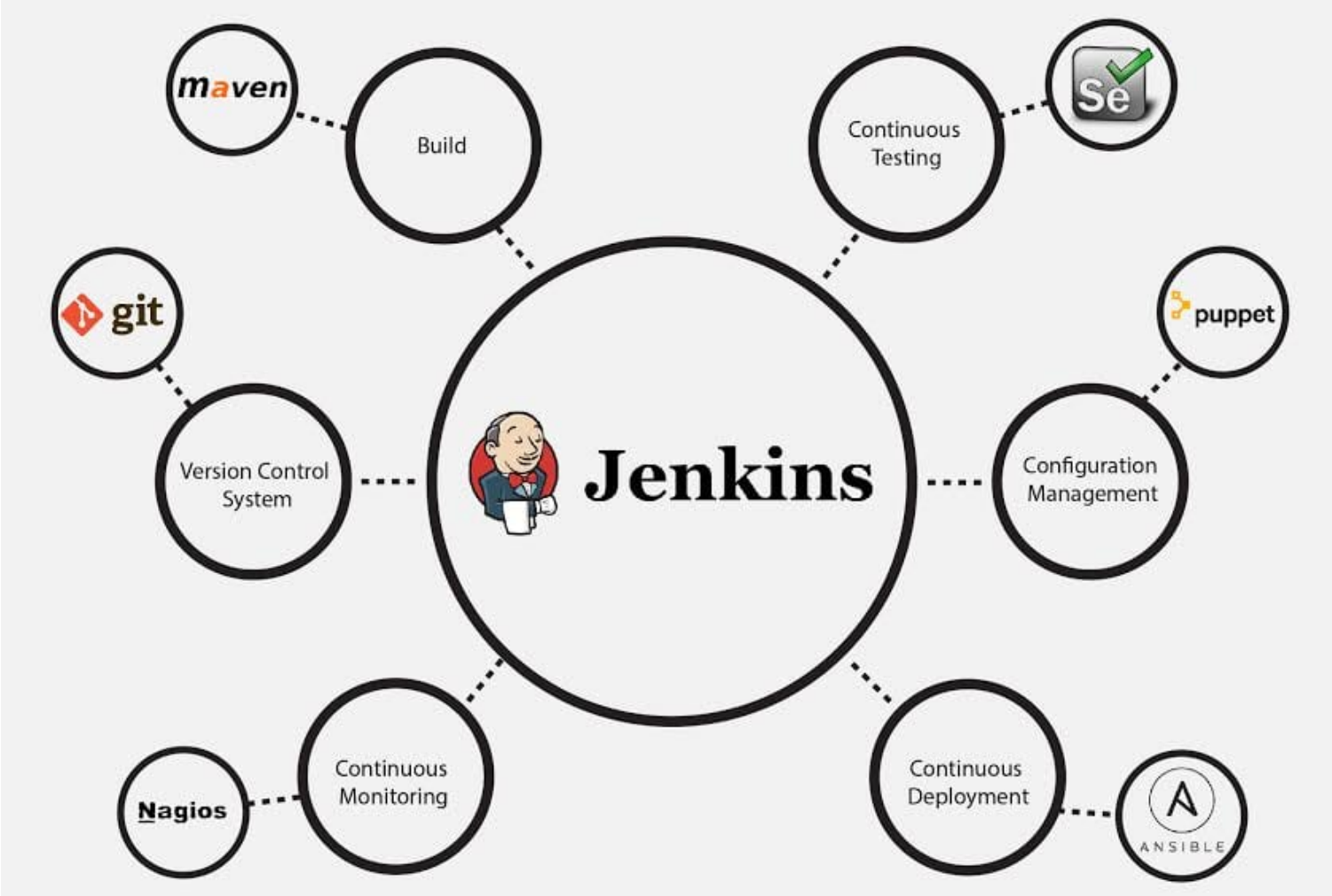
DevOps

What is Jenkins ?

- Jenkins is an open-source automation tool written in Java with plugins built for Continuous Integration purposes.
- Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project.
- It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.
- Jenkins helps organizations to accelerate the software development process through automation.
- Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis, and much more.
- Jenkins is powered by plugins to achieve Continuous Integration.
- Plugins allow the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For e.g. Git, Maven 2 project, Amazon EC2, HTML publisher, etc.

Web Reference Link

Jenkins role at various DevOps stages



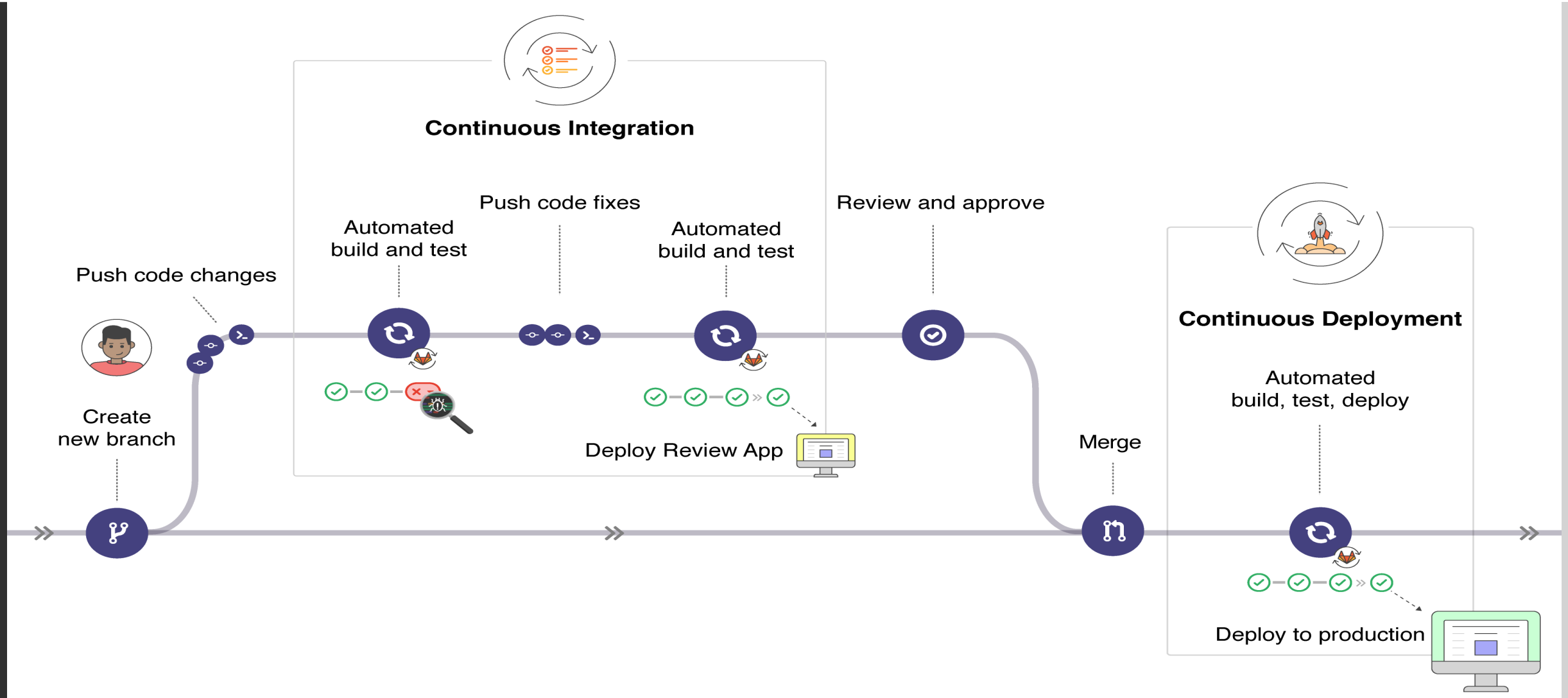
Advantages of Jenkins

- It is an open-source tool with great community support.
- It is easy to install.
- It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share it with the community.
- It is free of cost.
- It is built with Java and hence, it is portable to all the major platforms.

Jenkins Features

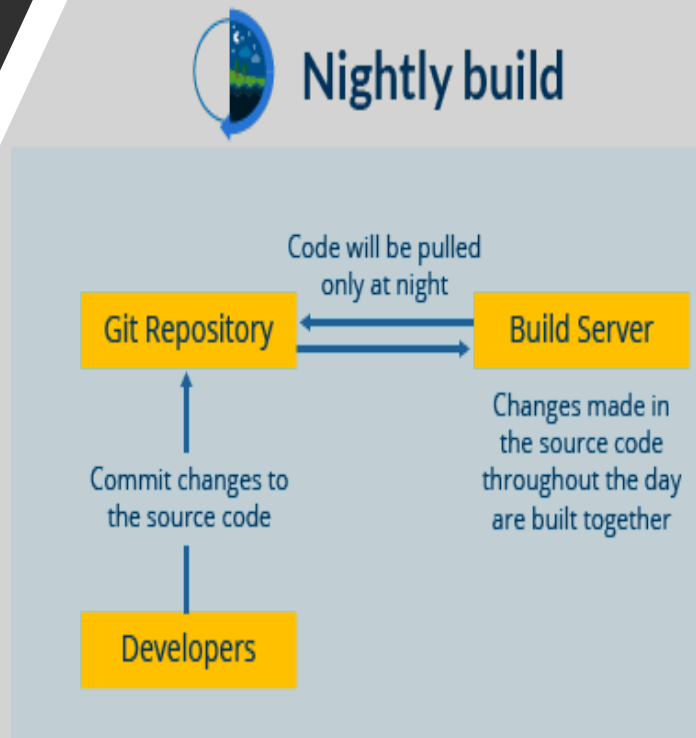
- The following are some facts about Jenkins that makes it better than other Continuous Integration tools:
    - **Adoption:** Jenkins is widespread, with more than 147,000 active installations and over 1 million users around the world.
    - **Plugins:** Jenkins is interconnected with well over 1,000 plugins that allow it to integrate with most of the development, testing and deployment tools.
- It is evident from the above points that Jenkins has a very high demand globally.

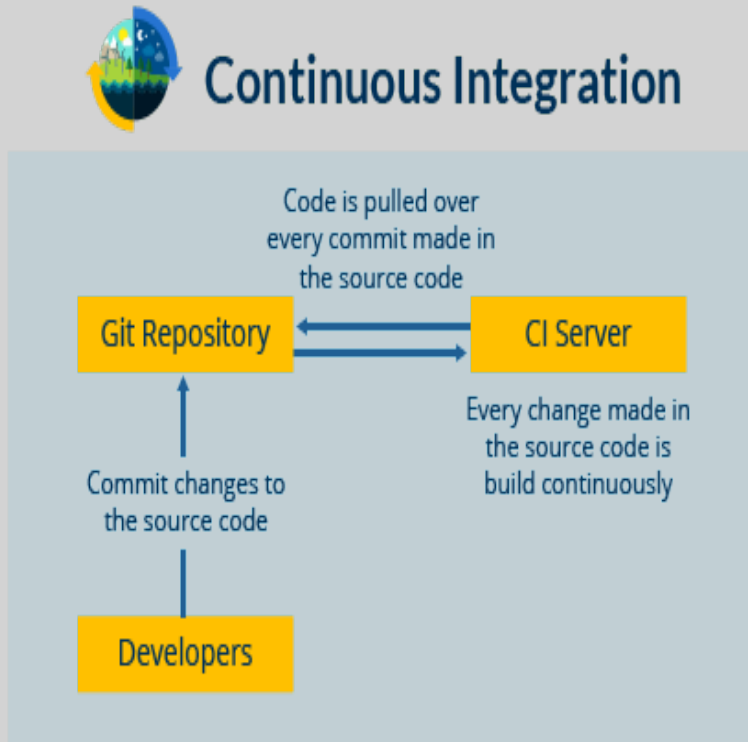
What is CI/CD ?


Continuous Integration
Push code changes
Create
new branch
Automated
build and test
Push code fixes
Automated
build and test
Deploy Review App
Review and approve
Merge
Continuous Deployment
Automated
build, test, deploy
Deploy to production

What is Continuous Integration? (CI)

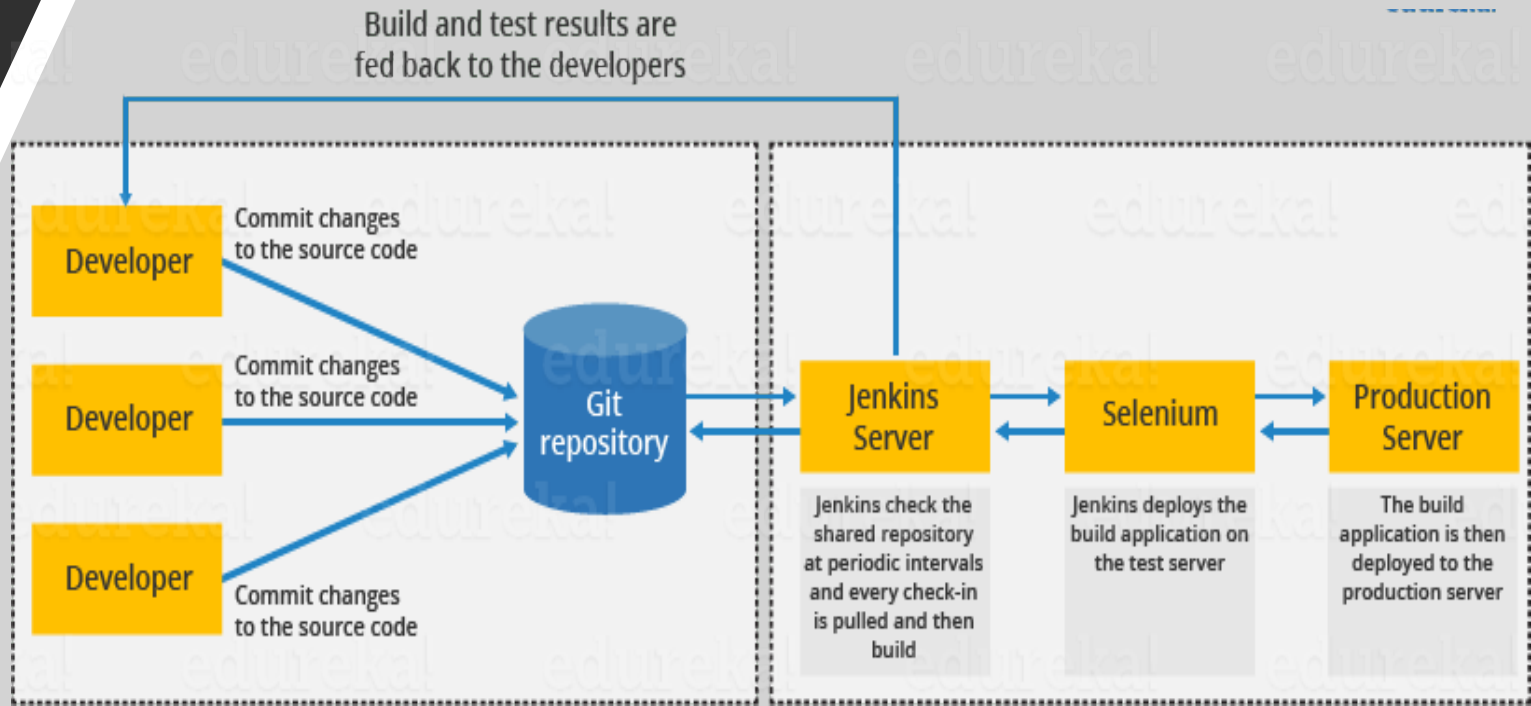
- Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently.
- Every commit made in the repository is then built that allows the teams to detect the problems early.
- Apart from this, depending on the Continuous Integration tool, there are several other functions like deploying the build application on the test server, providing the concerned teams with the build and test results, etc.





Continuous Integration with Jenkins

1. First, a developer commits the code to the source code repository. Meanwhile, the Jenkins server checks the repository at regular intervals for changes.
2. Soon after a commit occurs, the Jenkins server detects the changes that have occurred in the source code repository. Jenkins will pull those changes and will start preparing a new build.
3. If the build fails, then the concerned team will be notified.
4. If built is successful, then Jenkins deploys the built in the test server.
5. After testing, Jenkins generates a feedback and then notifies the developers about the build and test results.
6. It will continue to check the source code repository for changes made in the source code and the whole process keeps on repeating.



What is a pipeline?

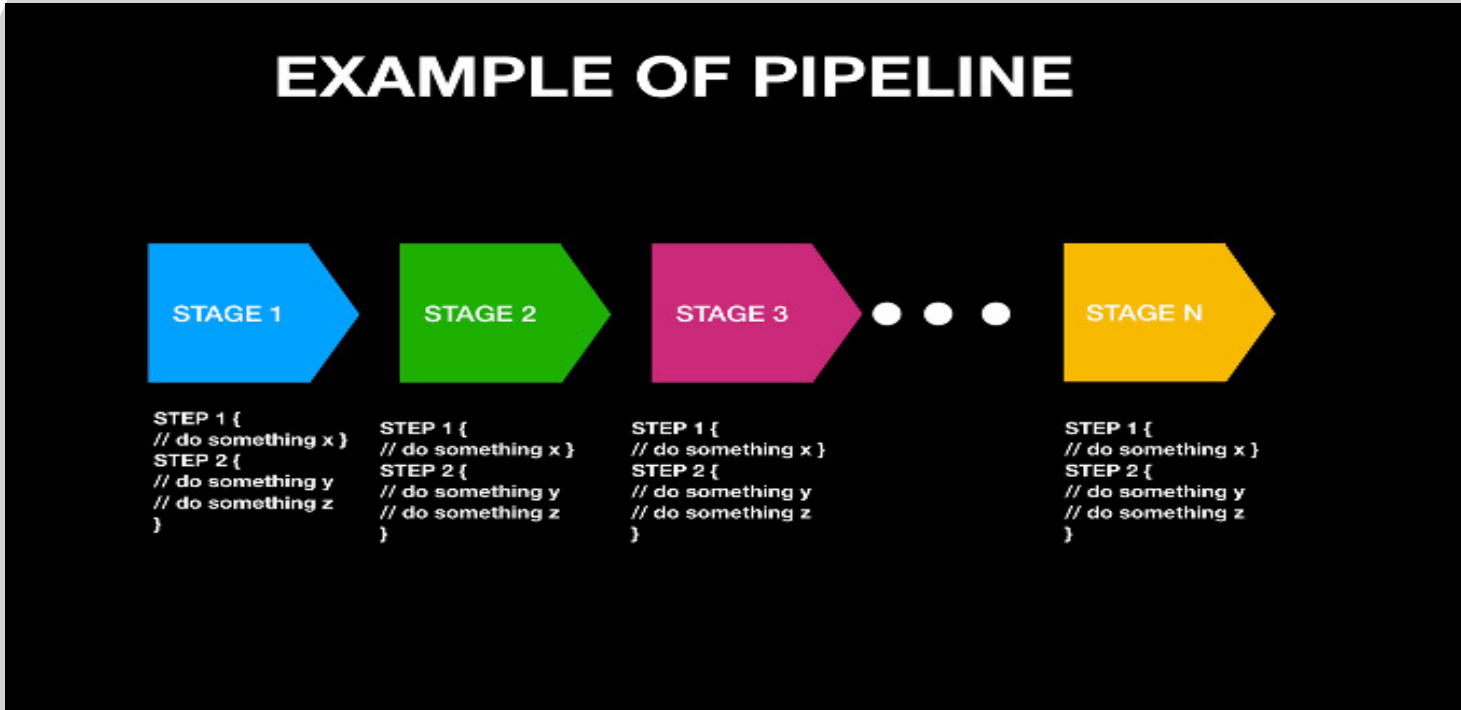
- Jenkins itself is not a pipeline.
- Just creating a new Jenkins job does not construct a pipeline.
- Jenkins is like a remote control—it's the place you click a button.
- Jenkins offers a way for other application APIs, software libraries, build tools, etc. to plug into Jenkins, and it executes and automates the tasks.
- On its own, Jenkins does not perform any functionality but gets more and more powerful as other tools are plugged into it.
- A pipeline is a separate concept that refers to the groups of events or jobs that are connected together in a sequence

Jenkins pipeline?

Stage* and *Step

- **Stage:** A block that contains a series of steps. A stage block can be named anything; it is used to visualize the pipeline process.
- **Step:** A task that says what to do. Steps are defined inside a stage block.
- The Jenkins pipeline is provided as a *codified script* typically called a **Jenkinsfile**, although the file name can be different.

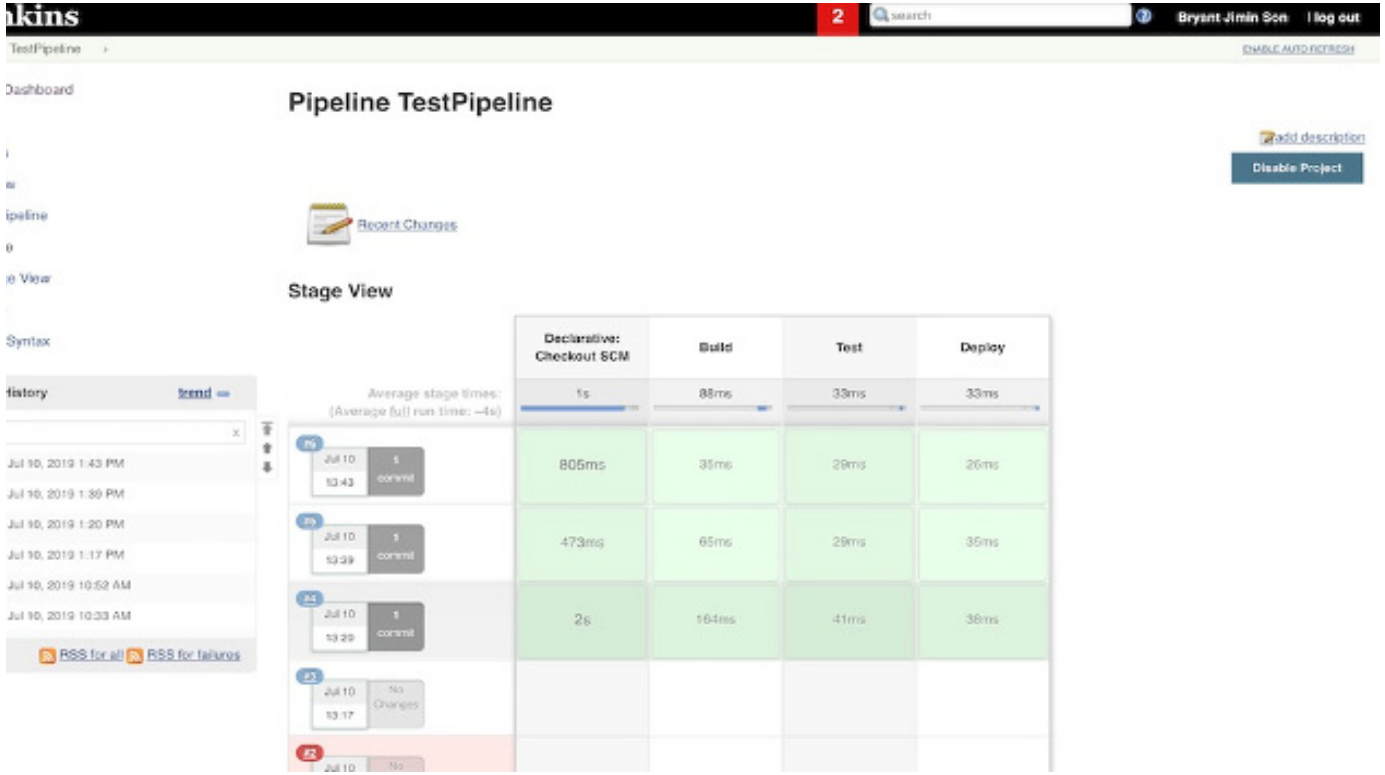
In the example diagram above, Stage 1 can be named "Build," "Gather Information," or whatever, and a similar idea is applied for the other stage blocks. "Step" simply says what to execute, and this can be a simple print command (e.g., **echo "Hello, World"**), a program-execution command (e.g., **java HelloWorld**), a shell-execution command (e.g., **chmod 755 Hello**), or any other command—as long as it is recognized as an executable command through the Jenkins environment.



Example: Jenkinsfile

```
// Example of Jenkins pipeline script

pipeline {
  stages {
    stage("Build") {
      steps {
        // Just print a Hello, Pipeline to the console
        echo "Hello, Pipeline!”


        // Compile a Java file. This requires JDKconfiguration from Jenkins
        javac HelloWorld.java


        // Execute the compiled Java binary called HelloWorld. This requires JDK configuration from Jenkins
        java HelloWorld


        // Executes the Apache Maven commands, clean then package. This requires Apache Maven configuration from Jenkins
        mvn clean package ./HelloPackage


        // List the files in current directory path by executing a default shell command
        sh "ls -ltr"
      }
    }
    // And next stages if you want to define further...
  } // End of stages
} // End of pipeline
```

Jenkins Pipeline



- A **Jenkins pipeline** is the way to execute a Jenkins job sequentially in a defined way by codifying it and structuring it inside multiple blocks that can include multiple step
- OK. Now that you understand what a Jenkins pipeline is, I'll show you how to create and execute a Jenkins pipeline. At the end of the tutorial, you will have built a Jenkins pipeline like this's containing tasks.

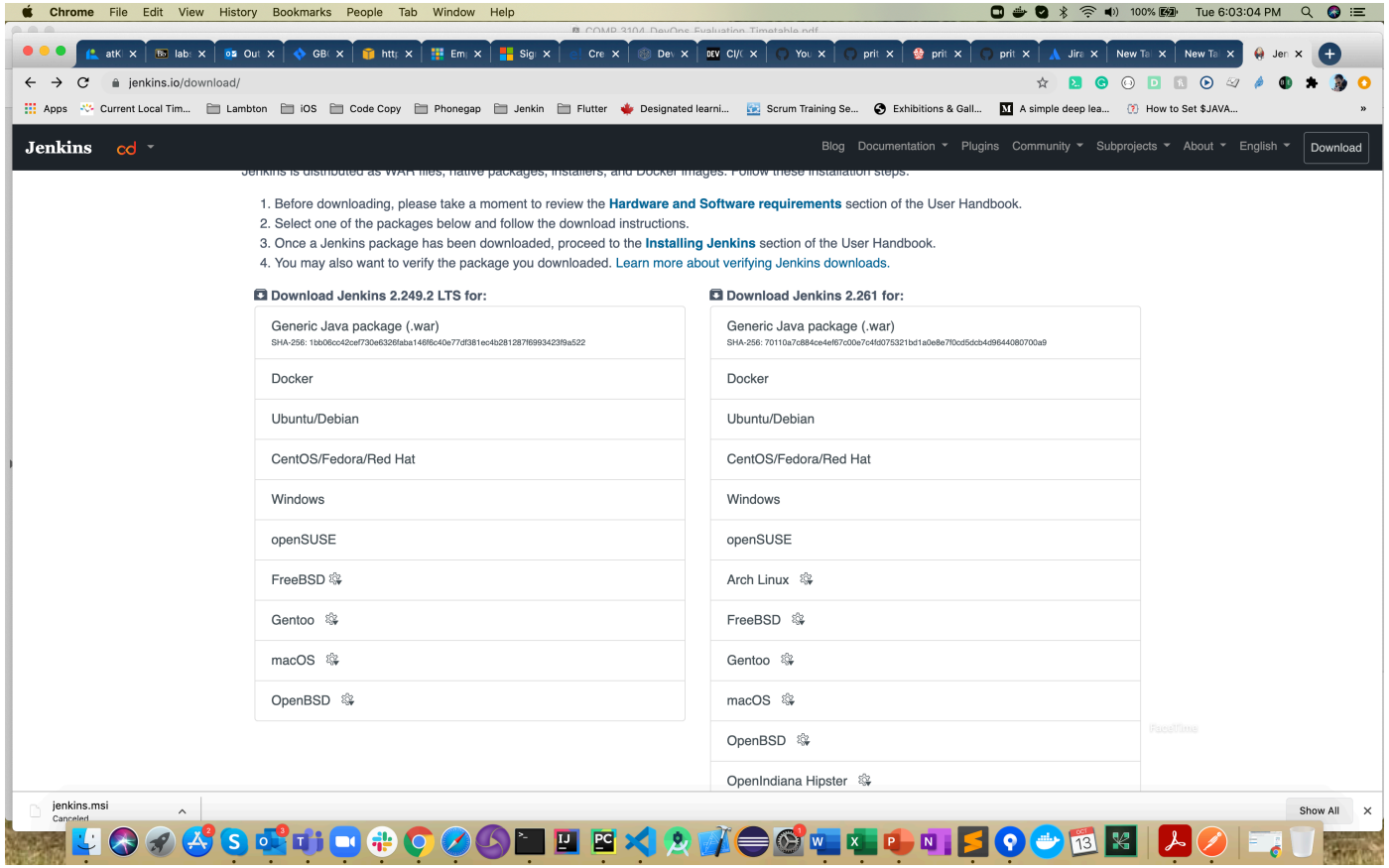
Jenkins Installation

Prerequisites

- **Java Development Kit:** If you don't already have it, install a JDK and add it to the environment path so a Java command (like **java jar**) can be executed through a terminal.
- **Supported Java version 8.0 to 11.0 / [52, 55]**
- **Basic computer operations:** You should know how to type some code, execute basic Linux commands through the shell, and open a browser.

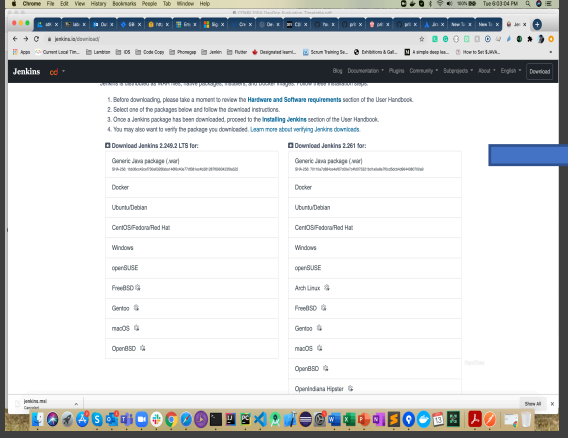
Web Reference Link

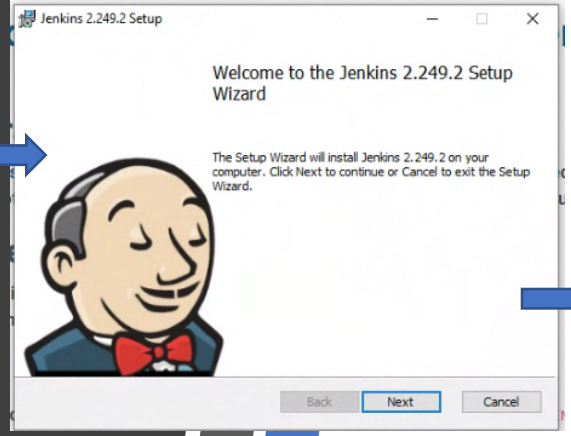
Jenkins Installation

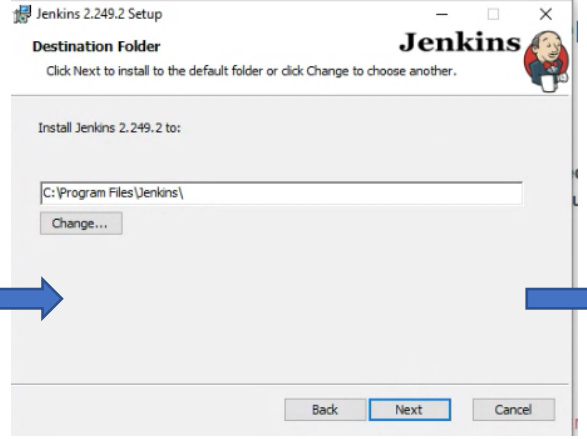


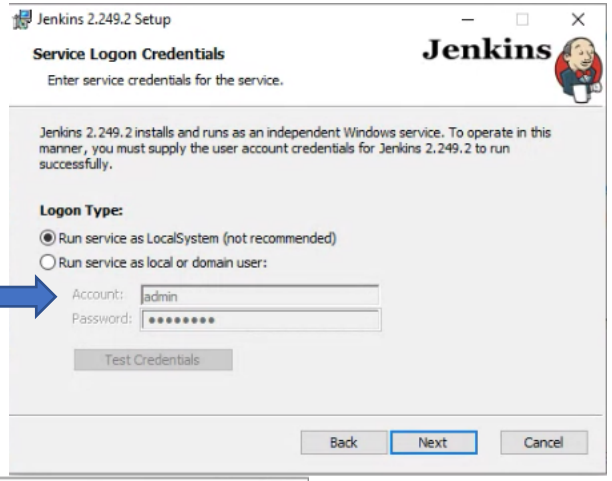
- **Step 1: Download Jenkins**
- Navigate to the Jenkins download page. Scroll down to **Generic Java package (.war)** and click on it to download the file; save it someplace where you can locate it easily. (If you choose another Jenkins distribution, the rest of tutorial steps should be pretty much the same, except for Step 2.) The reason to use the WAR file is that it is a one-time executable file that is easily executable and removable.

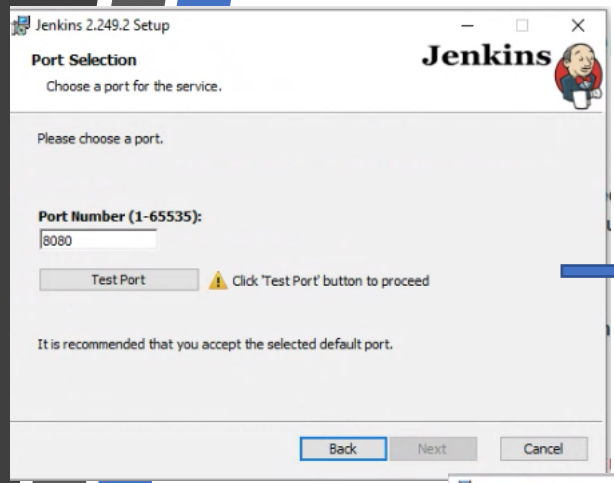
Jenkins Installation

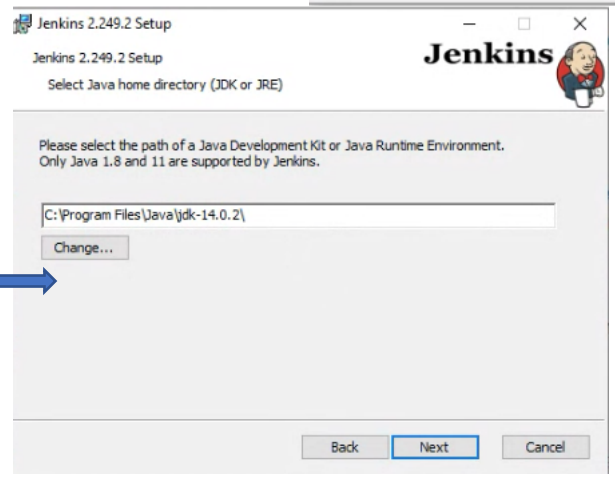


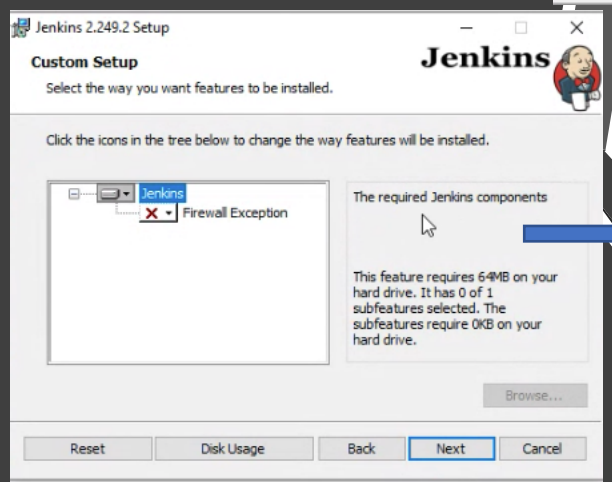


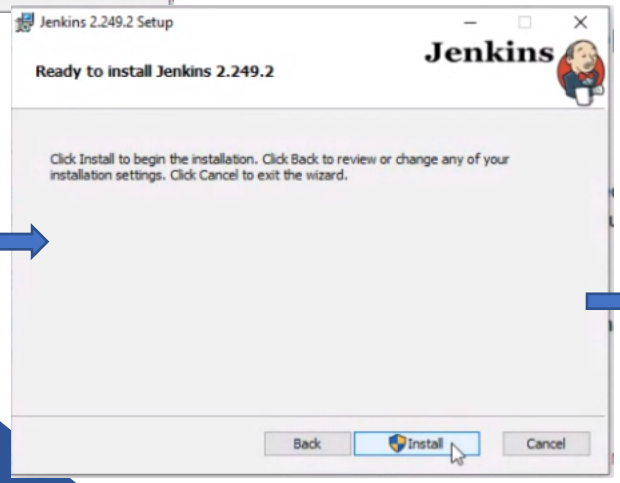


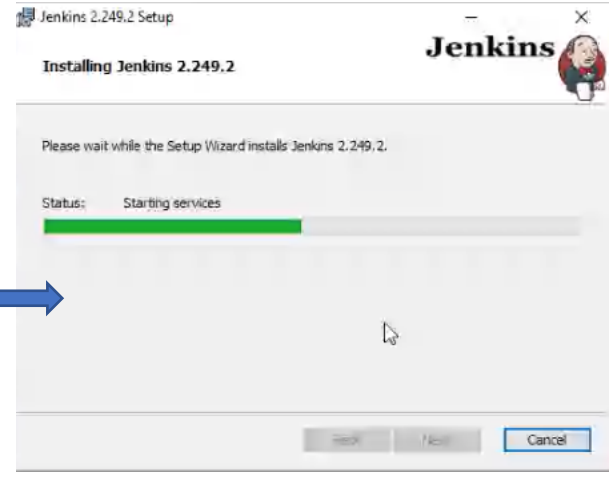




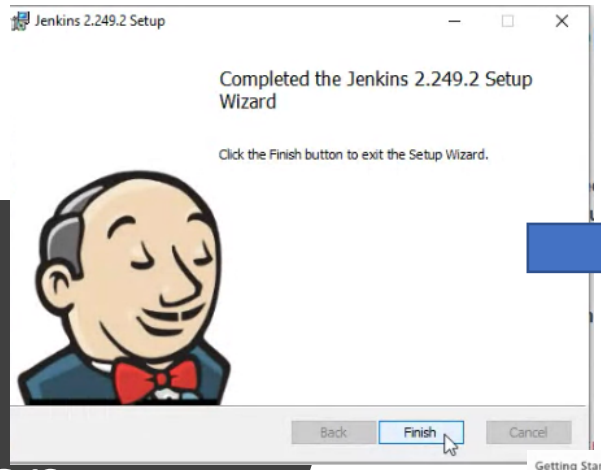


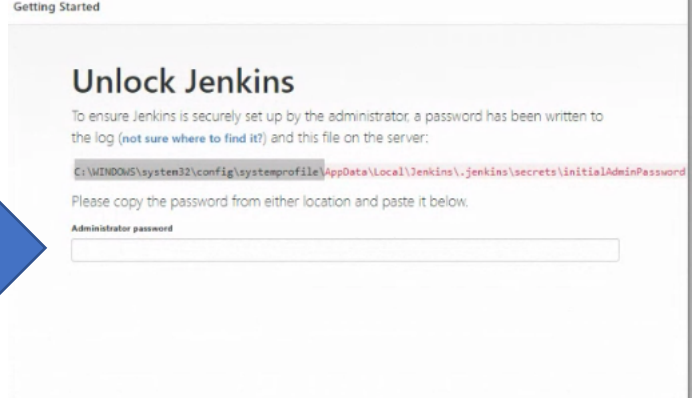


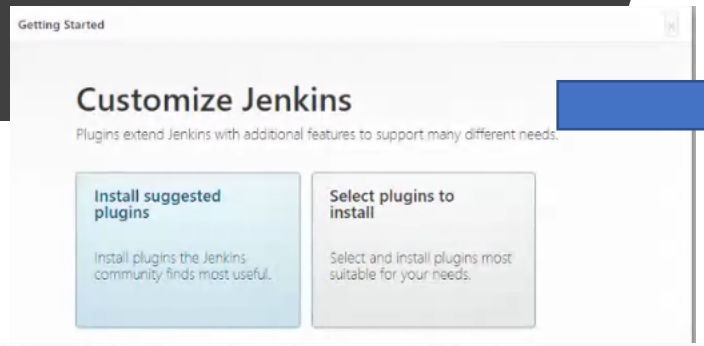


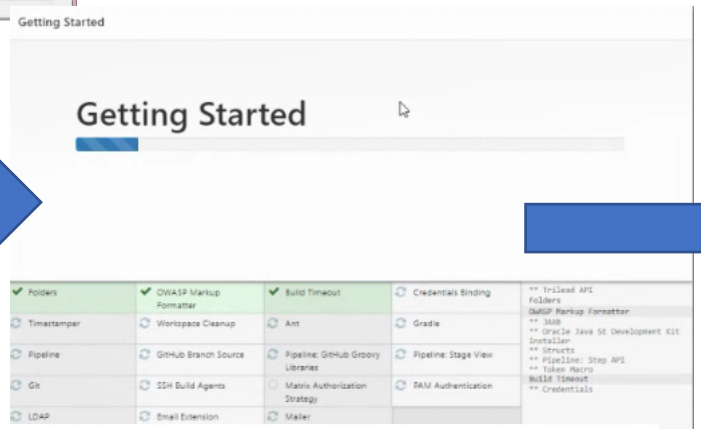


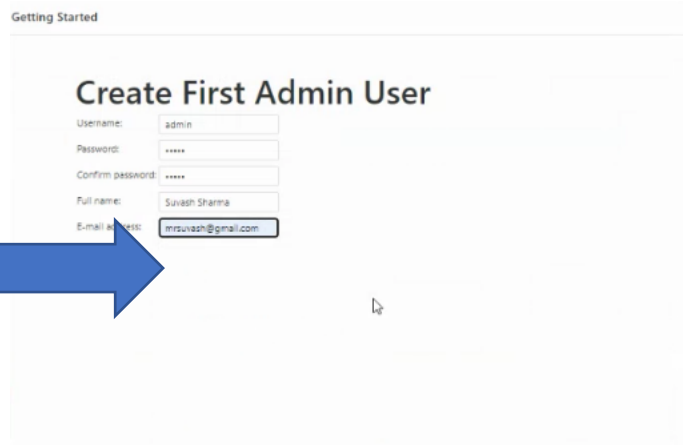
Jenkins Installation

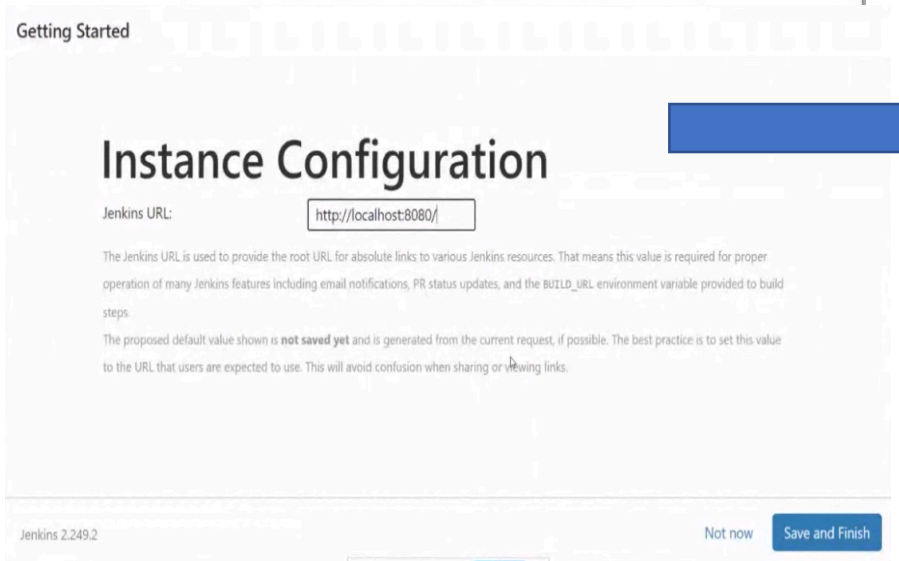


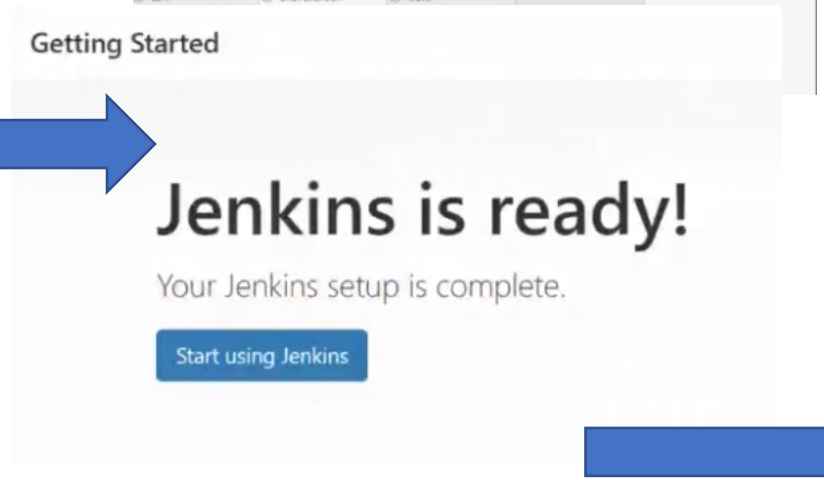


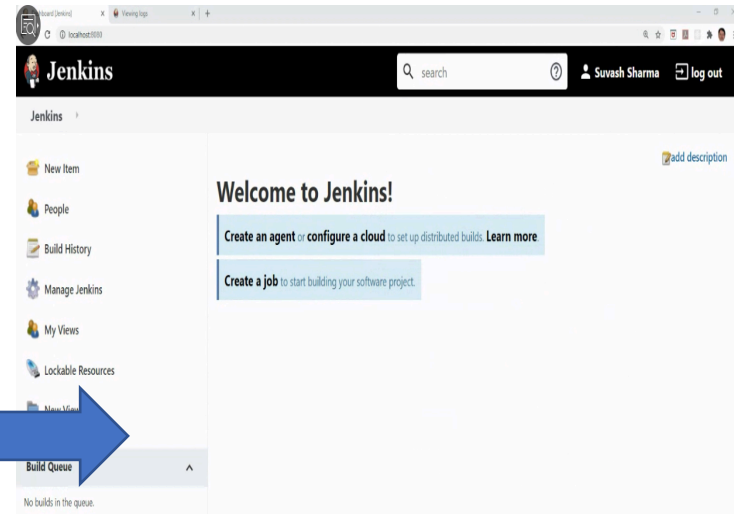


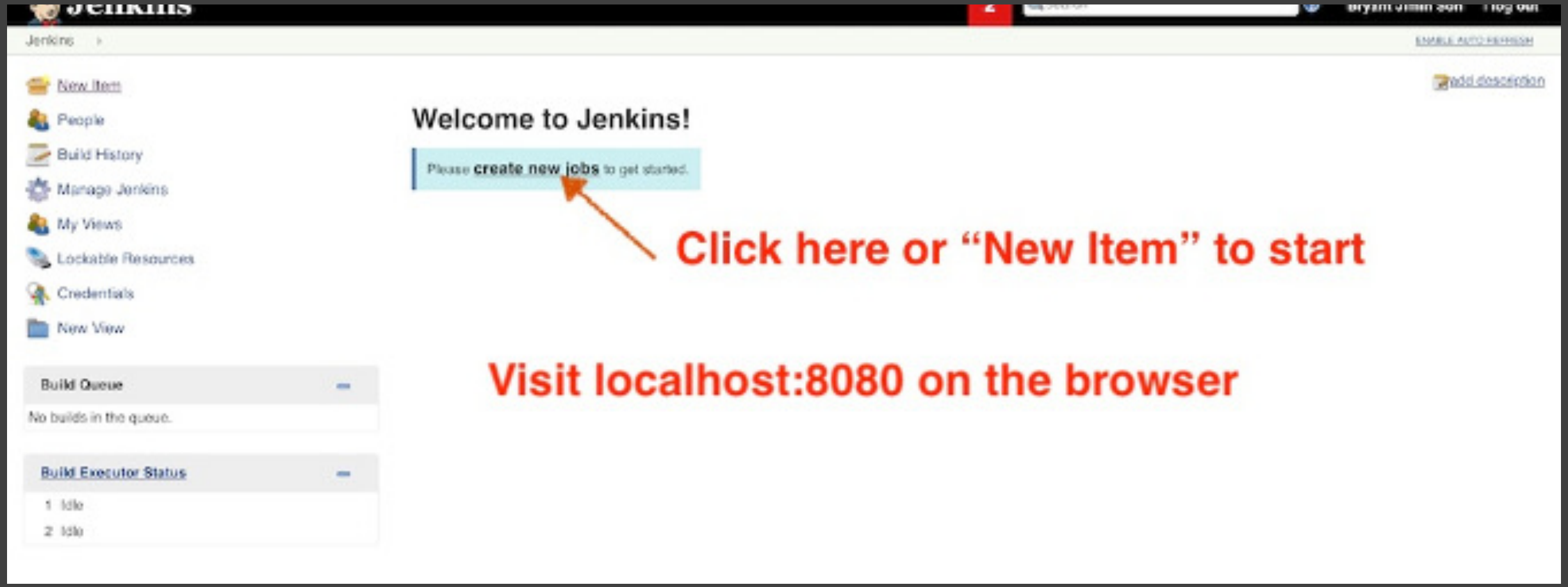






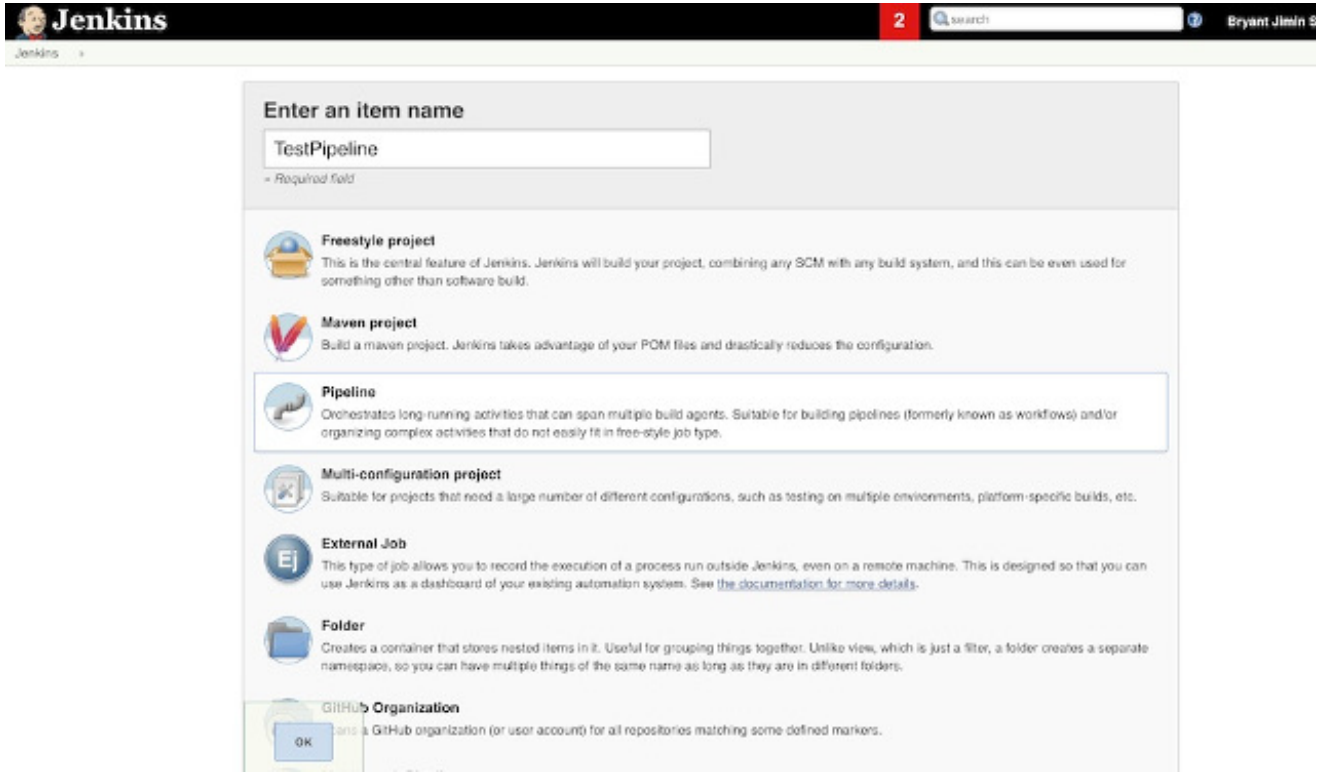






- **Step 3: Create a new Jenkins job**
- Open a web browser and navigate to **localhost:8080**. Unless you have a previous Jenkins installation, it should go straight to the Jenkins dashboard. Click **Create New Jobs**. You can also click **New Item** on the left.

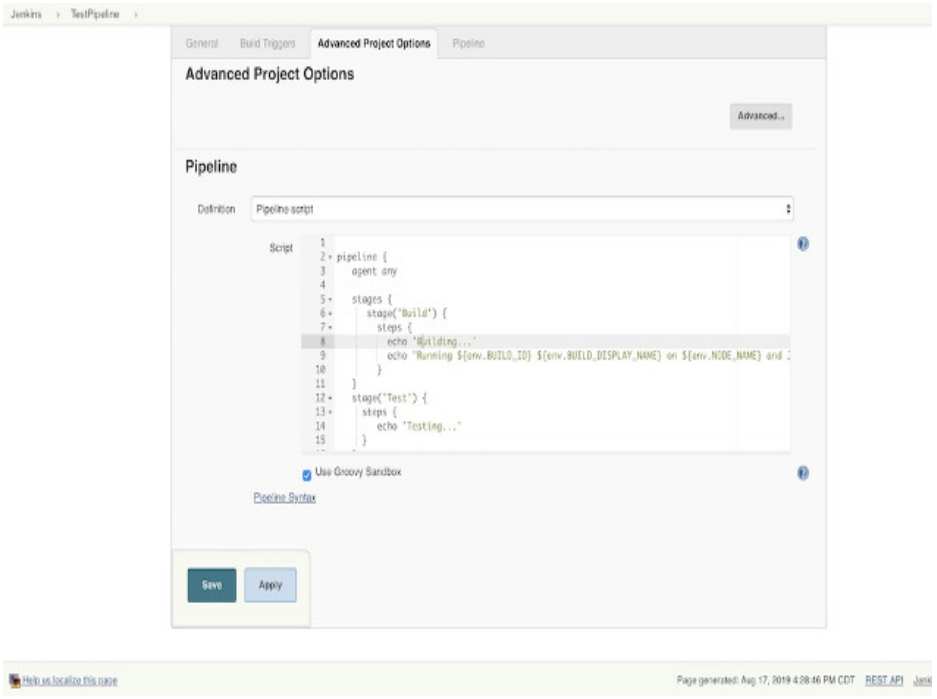
Creating Pipeline



- **Step 4: Create a pipeline job**
- In this step, you can select and define what type of Jenkins job you want to create. Select **Pipeline** and give it a name (e.g., Test Pipeline). Click **OK** to create a pipeline job.

You will see a Jenkins job configuration page. Scroll down to find **Pipeline section**. There are two ways to execute a Jenkins pipeline. One way is by *directly writing a pipeline script* on Jenkins, and the other way is by retrieving the *Jenkins file from SCM* (source control management). We will go through both ways in the next two steps.

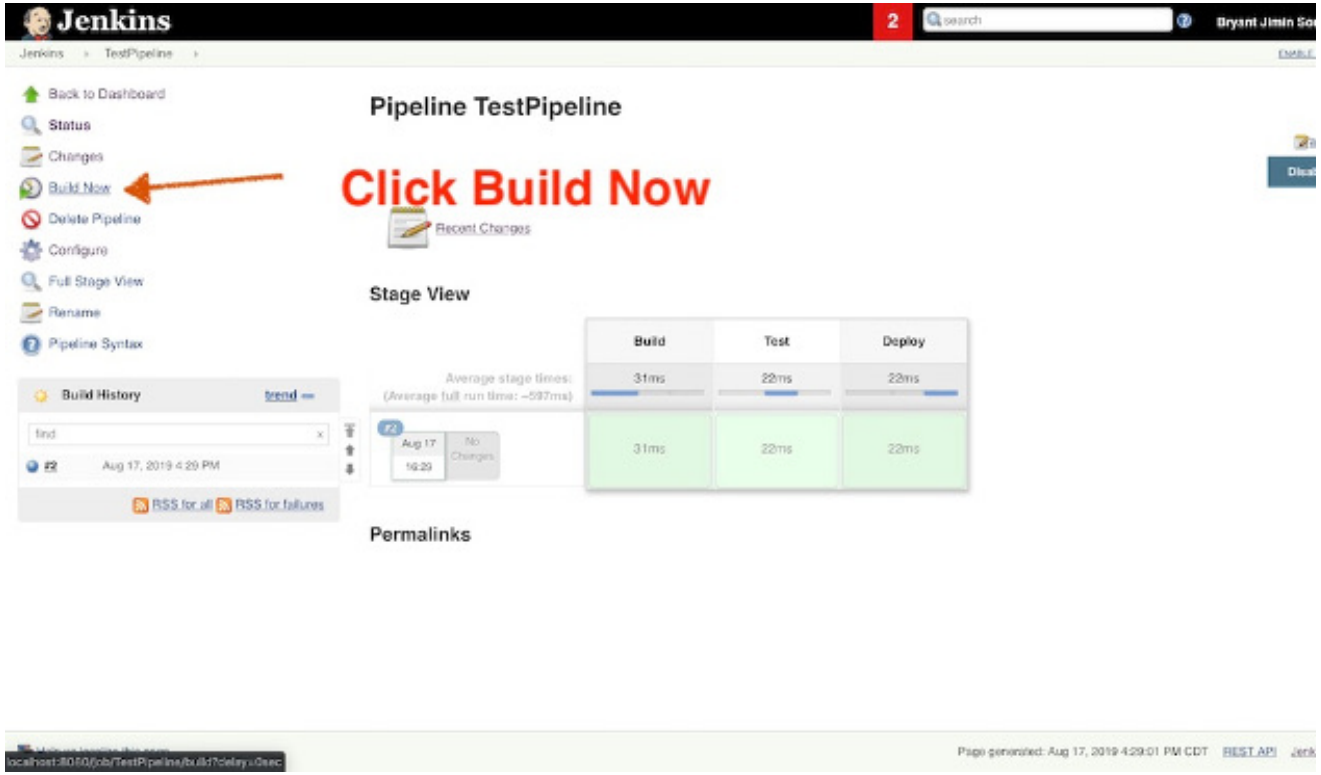
Configure Pipeline




```
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        echo 'Building...'
        echo "Running ${env.BUILD_ID}
${env.BUILD_DISPLAY_NAME} on
${env.NODE_NAME} and JOB
${env.JOB_NAME}"
      }
    }
    stage('Test') {
      steps {
        echo 'Testing...'
      }
    }
    stage('Deploy') {
      steps {
        echo 'Deploying...'
      }
    }
  }
}
```

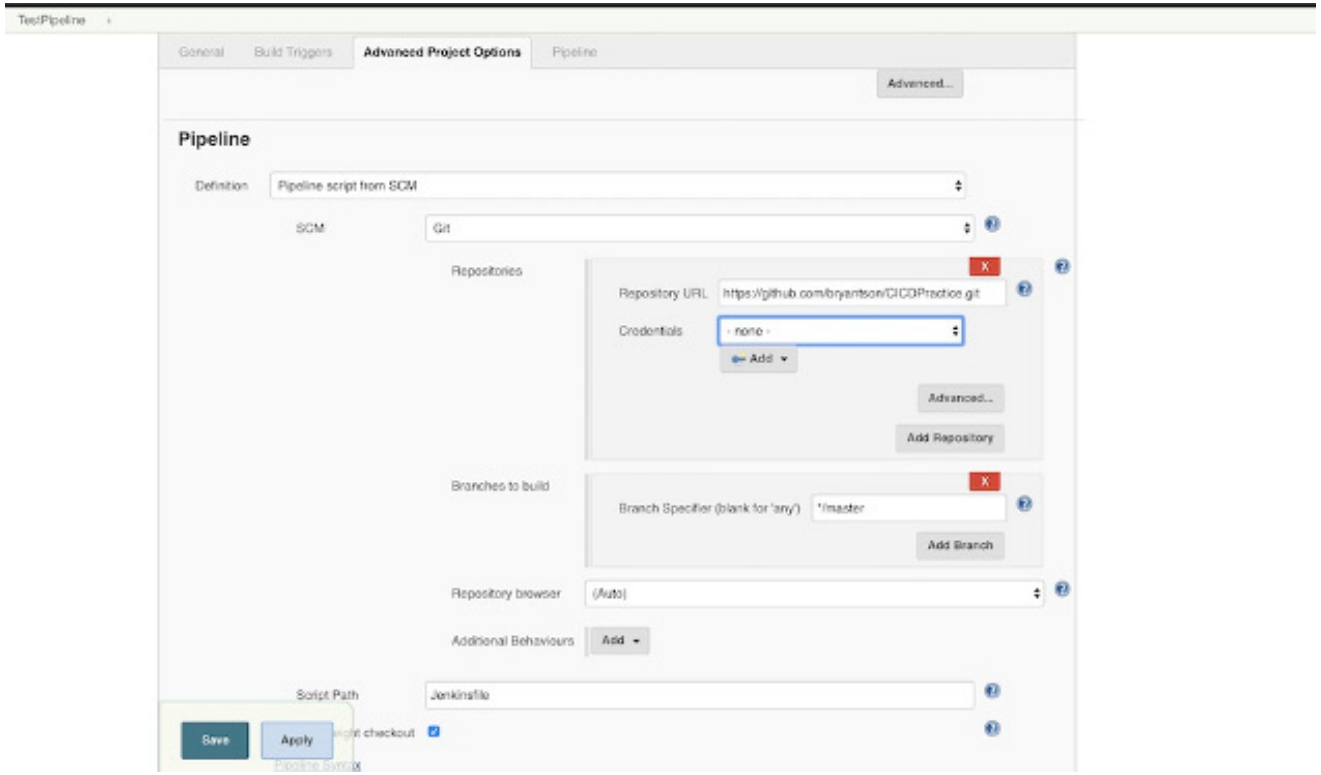
- **Step 5: Configure and execute a pipeline job through a direct script**
- To execute the pipeline with a direct script, begin by copying the contents
- .Choose **Pipeline script** as the **Destination** and paste the **Jenkinsfile** contents in **Script**. Spend a little time studying how the Jenkins file is structured. Notice that there are three Stages: Build, Test, and Deploy, which are arbitrary and can be anything. Inside each Stage, there are Steps; in this example, they just print some random messages.
- Click **Save** to keep the changes, and it should automatically take you back to the Job Overview.

Start the Pipeline



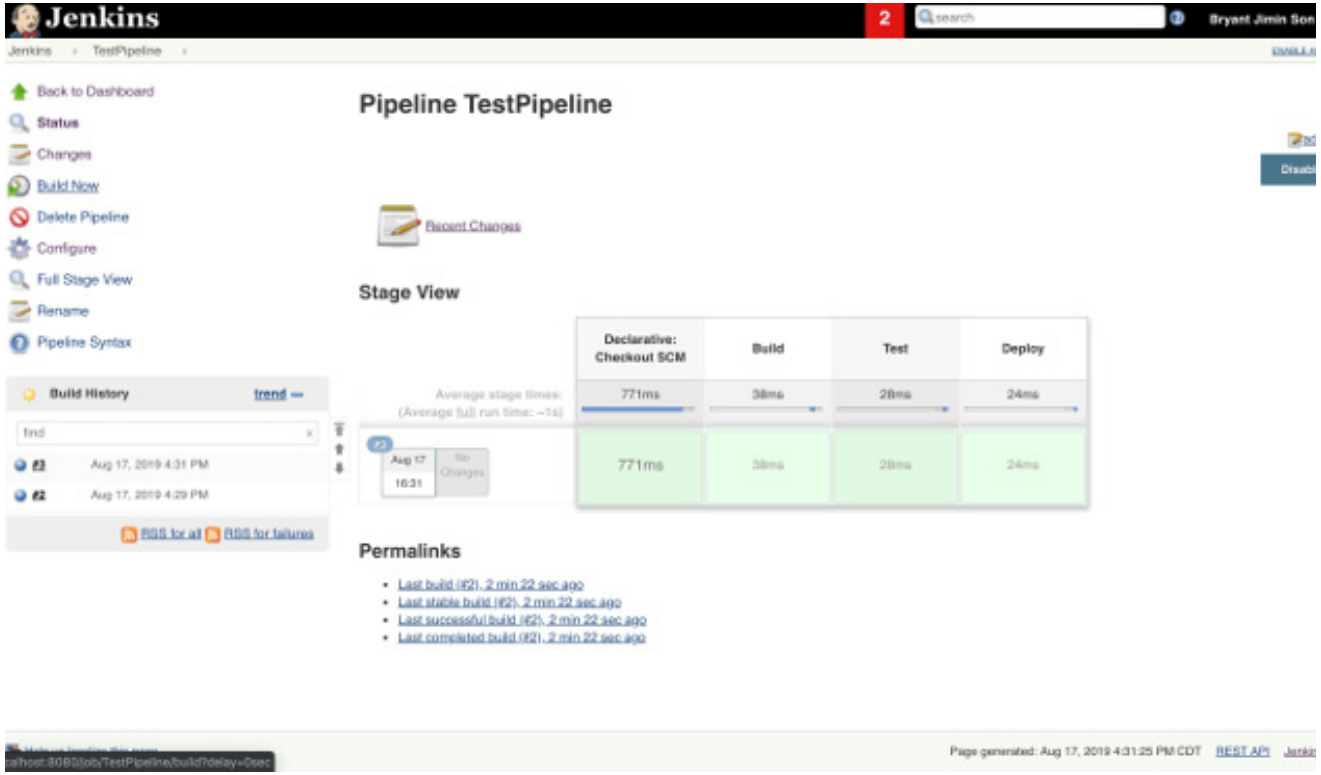
- To start the process to build the pipeline, click **Build Now**. If everything works, you will see your first pipeline (like the one below).

Execute a Pipeline



- **Step 6: Configure and execute a pipeline job with SCM**
- Now, switch gears: In this step, you will Deploy the same Jenkins job by copying the **Jenkinsfile** from a source-controlled GitHub.
- Click **Configure** to modify the existing job. Scroll to the **Advanced Project Options** setting, but this time, select the **Pipeline script from SCM** option in the **Destination** dropdown. Paste the GitHub repo's URL in the **Repository URL**, and type **Jenkinsfile** in the **Script Path**. Save by clicking the **Save** button.

Building Pipeline



- To build the pipeline, once you are back to the Task Overview page, click **Build Now** to execute the job again. The result will be the same as before, except you have one additional stage called **Declaration: Checkout SCM**.

5 CI/CD Best Practices for Better Code Quality

1. Decide What to Test and When

- In CI/CD first we have to decide which part of your code you are testing, and when you should be testing it.
- Testing is important to be sure that an already existing functionality was not broken by code and will not break from future code changes.
- Only manual testing isn't enough. Test automatization is required to make the testing process faster/cheaper and to be able integrate it with the CI/CD pipeline. This makes the development process faster/cheaper and helps support code in high quality.
- For automation, you can use tools like open source Taurus, which automates the performance testing process.

5 CI/CD Best Practices for Better Code Quality

2. Adapt Your CI/CD Pipeline to Your Development Process

- The CI/CD pipeline is the automatization of your development process, so it needs to fit your development process strategy. The first step is formalization of the development process, to ensure you know which branches you have and what they do. Then, you can create the correct pipeline.
- For example, if you have multi-branches with one feature per branch and a pull request review process, then your CI/CD pipeline shouldn't enable merging pull requests if the build for the branch with the feature failed. But, if you have only one branch (in case of Feature Toggle for example) or if instead of the review process you use pair-programming, then the pipeline should not care about pull requests.
- If you need to take care of code quality, the CI/CD pipeline can use Checkstyle or any tool for static code analysing, or CI/CD may require an integration with SonarQuibe to collect information about the code quality status that also shows code coverage report by tests.

5 CI/CD Best Practices for Better Code Quality

3. Make the Build as Fast as Possible

- A shorter code-change-result cycle makes the code easier to fix and update, as changes are still fresh in developers' minds. You can achieve a quicker build by triggering it as soon as code is pushed to the repository (webhooks are the best way to do this). Another way to shorten the process is to split the build into parts, and run them in Parallel. For example, tests can be split into chunks and run in parallel.
- If your CI/CD tool supports horizontal scaling (for example Jenkins CI does) you can add more machines. If you see you have a build queue that is waiting for a free CI/CD machine it is good time to add more machines.
- Finally, vertical scaling is also helpful. If an application build requires intensive work with HDD, you can use SSD on CI/CD machines or build on memory based partitions.

5 CI/CD Best Practices for Better Code Quality

4. Build in a Containerized Environment

- If you are developing an application and want to create a CI/CD infrastructure, you probably need to install additional software on CI/CD machines (e.g. Git for checkout code, Maven/Ant/Gradle to build the code, Ansible to deploy it and so on). Note that it is required to use a particular version for each software component. If you install any version it most probably will not work.
- The solution can be any container solution using tools like Docker. Don't install any additional software that is required for building your application. Instead you can install additional applications inside the Docker container and run the build process inside the container. Then it will be much more easy to support the CI/CD infrastructure.

5 CI/CD Best Practices for Better Code Quality

5. Install the Infrastructure Automatically and as Code

- To prepare your CI/CD infrastructure you need to perform some actions:
- Create virtual machines - because you may need more than one machine to install CI/CD to increase its performance, or you might want to install additional tools like SonarQube for code quality monitoring, a Selenium server for UI tests, machines for applications' deployment (DEV, STAGING, PROD environments for example) or a monitoring tool (e.g. Graphana)
- Install all the required software on it
- Configure the software properly

What is Markdown?

- Markdown is a lightweight markup language that you can use to add formatting elements to plaintext text documents.
- Created by John Gruber in 2004, Markdown is now one of the world's most popular markup languages.

https://www.markdownguide.org/getting-started/

https://www.markdownguide.org/cheat-sheet/

Thank You