

Department of Computer Science & Engineering

(CSM & CSD)

III-II Sem Lab Manual

DevOps Lab Manual



By

Dr.R.Naveen Kumar

T.Sravanthi

Vaagdevi Engineering College

Bollikunta, Warangal



VAAGDEVI ENGINEERING COLLEGE
COMPUTER SCIENCE & ENGINEERING
(CSM & CSD)
EVEN SEMESTER LAB MANUAL

DevOps Lab

Course Objectives:

- Able to learn the agile relationship between development and IT operations
- To understand the skill sets and high-functioning teams involved in DevOps
- To learn DevOps and related methods to reach a continuous delivery capability
- To Implement automated system update and DevOps lifecycle.

Course Outcomes:

- Able to Identify components of Devops environment
- Able to apply different project management, integration, testing and code deployment tool
- Able to investigate different DevOps Software development, models
- Able to explain continuous integration and development using Jenkins.

List of Experiments:

1. Write code for a simple user registration form for an event.
2. Explore Git and GitHub commands.
3. Practice Source code management on GitHub. Experiment with the source code written in exercise 1.
4. Jenkins installation and setup, explore the environment.
5. Demonstrate continuous integration and development using Jenkins.
6. Explore Docker commands for content management.
7. Develop a simple containerized application using Docker.
8. Integrate Kubernetes and Docker.
9. Automate the process of running containerized application developed in exercise 7 using Kubernetes.
10. Install and Explore Selenium for automated testing.
11. Write a simple program in JavaScript and perform testing using Selenium.
12. Develop test cases for the above containerized application using selenium.

1. Write a program for sample registration page.

Registration.html

```
<html>
<body>
<form action="servlet/Register" method="post">

Name:<input type="text" name="userName"/><br/><br/>
Password:<input type="password" name="userPass"/><br/><br/>
Email Id:<input type="text" name="userEmail"/><br/><br/>
Country:
<select name="userCountry">
<option>India</option>
<option>Pakistan</option>
<option>other</option>
</select>

<br/><br/>
<input type="submit" value="register"/>

</form>
</body>
</html>
```

Register.java

```
import java.io.*;
import java.sql.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class Register extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");
        String p=request.getParameter("userPass");
        String e=request.getParameter("userEmail");
        String c=request.getParameter("userCountry");

        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

            PreparedStatement ps=con.prepareStatement(
                "insert into registeruser values(?,?,?,?)");

            ps.setString(1,n);
            ps.setString(2,p);
            ps.setString(3,e);
            ps.setString(4,c);

            int i=ps.executeUpdate();
            if(i>0)
                out.print("You are successfully registered...");
```

```
}catch (Exception e2) {System.out.println(e2);}
```

```
out.close();
```

```
}
```

```
}
```

Web.xml file

```
<web-app>
```

```
<servlet>
```

```
<servlet-name>Register</servlet-name>
```

```
<servlet-class>Register</servlet-class>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
<servlet-name>Register</servlet-name>
```

```
<url-pattern>/servlet/Register</url-pattern>
```

```
</servlet-mapping>
```

```
<welcome-file-list>
```

```
<welcome-file>register.html</welcome-file>
```

```
</welcome-file-list>
```

```
</web-app>
```

Create a table in a database:

```
CREATE TABLE "REGISTERUSER"
```

```
( "NAME" VARCHAR2(4000),
```

```
"PASS" VARCHAR2(4000),
```

```
"EMAIL" VARCHAR2(4000),
```

```
"COUNTRY" VARCHAR2(4000)
```

```
)
```


What is Git?

Git is a version control system which lets you track changes you make to your files over time. With Git, you can revert to various states of your files (like a time traveling machine). You can also make a copy of your file, make changes to that copy, and then merge these changes to the original copy.

For example, you could be working on a website's landing page and discover that you do not like the navigation bar. But at the same time, you might not want to start altering its components because it might get worse.

With Git, you can create an identical copy of that file and play around with the navigation bar. Then, when you are satisfied with your changes, you can merge the copy to the original file.

You are not limited to using Git just for source code files – you can also use it to keep track of text files or even images. This means that Git is not just for developers – anyone can find it helpful.

How to install Git

In order to use Git, you have to install it on your computer. To do this, you can download the latest version on the [official website](https://git-scm.com/downloads).

<https://git-scm.com/downloads>

You can download for your operating system from the options given.

You can also install Git using the command line, but since the commands vary with each operating system, we'll focus on the more general approach.

How to configure Git

- I will assume that at this point you have installed Git.
- To verify this, you can run this command on the command line:
 - `git --version` – gives the latest version installed on the system.
- To configure **git**: - execute the below command.
git config
 - Now set your username and email address for git. Git will use this information to identify who made specific changes to files.
 - To set username:
git config --global user.name "YOUR_USERNAME"
 - To set email id:

git config --global user.email "YOUR_EMAIL"

Just make sure to replace "YOUR_USERNAME" and "YOUR_EMAIL" with the values you choose.

How to Create a Repository in Git?

Execute the below command to create your repository in git:

git init

How to Create and Initialize a Project in Git

We are finally done with installing and setting up Git. It is now time to create our project.

I have created a folder on my desktop called Git and GitHub tutorial. Using the command line, navigate to your new project's location. For me, I would run the following commands:

cd desktop

cd Git and GitHub tutorial

or

If you are new to the command line and are still learning how to use it to navigate around your PC, then I would suggest using Microsoft's Visual Studio Code. It is a code editor which has an inbuilt terminal for executing commands. You can download it [here](https://code.visualstudio.com/download).

<https://code.visualstudio.com/download>

After installing VS Code, open your project in the editor and open a new terminal for your project. This automatically points the terminal/command line to your project's path.

Now to initialize your project, simply run git init. This will tell Git to get ready to start watching your files for every change that occurs. It looks like this:

```
THECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial
$ git init
Initialized empty Git repository in C:/Users/IHECHIKARA/Desktop/Git and GitHub tutorial/.git/
```

git init

The first line has information about my PC and the path to where the folder exists. The second line is the command `git init`, and the third line is the response sent back telling me that my repository (repo) has been initialized. It is considered empty because we have not told Git what files to track.

A repository is just another way to define a project being watched/tracked by Git.

Git project files

I have created only one file called `todo.txt`. This is what the file looks like:

```
MY TO-DO LIST

1. Write an article.
2. Code.
3. Study books.
4. Attend classes on time.
5. Visit aunt.
6. Apply for remote jobs.
```

What is GitHub?

GitHub is an online hosting service for Git repositories. Imagine working on a project at home and while you are away, maybe at a friend's place, you suddenly remember the solution to a code error that has kept you restless for days.

You cannot make these changes because your PC is not with you. But if you have your project hosted on GitHub, you can access and download that project with a command on whatever computer you have access to. Then you can make your changes and push the latest version back to GitHub.

In summary, GitHub lets you store your repo on their platform. Another awesome feature that comes with GitHub is the ability to collaborate with other developers from any location.

Now that we have created and initialized our project locally, let's push it to GitHub.

If you are a beginner, you will come across some new terms like push, commit, add, and so on – but do not be overwhelmed by them. With some practice you will be able to remember these terms and what they do.

How to push a repository to GitHub

I will divide this section into steps to help you understand the process more clearly.

Step 1 – Create a GitHub account

To be able to use GitHub, you will have to create an account first. You can do that on [website](https://github.com).

<https://github.com>

Step 2 – Create a repository

You can click on the + symbol on the top right corner of the page then choose "New repository". Give your repo a name then scroll down and click on "Create repository".

Step 3 – Add and commit file(s)

Before we "add" and "commit" our files, you need to understand the stages of a file being tracked by Git.

Committed state

A file is in the **committed** state when all the changes made to the file have been saved in the local repo. Files in the committed stage are files ready to be pushed to the remote repo (on GitHub).

Modified state

A file in the **modified** state has some changes made to it but it's not yet saved. This means that the state of the file has been altered from its previous state in the committed state.

Staged state

A file in the **staged** state means it is ready to be committed. In this state, all necessary changes have been made so the next step is to move the file to the commit state.

You can understand this better by imagining Git as a camera. The camera will only take a snapshot when the file reaches the commit state. After this state, the camera starts comparing changes being made to the same file with the last snapshot (this is the modified state). And when the required changes have been made, the file is staged and moved to the commit state for a new snapshot.

This might be a lot of information to take in at the moment, but do not be discouraged – it gets easier with practice.

How to add files in Git

```
git add .    #(dot)
git status
```

How to commit files in Git

```
git commit -m "first commit" command.
```

The first part of the command `git commit` tells Git that all the files staged are ready to be committed so it is time to take a snapshot. The second part `-m "first commit"` is the commit message. `-m` is shorthand for message while the text inside the parenthesis is the commit message.

After executing this command, you should get a response similar to this:

```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git commit -m "first commit"
[main (root-commit) 48195f0] first commit
1 file changed, 8 insertions(+)
create mode 100644 todo.txt
```

`git commit`

Now our file is in the committed state.

Step 4 – Push the repository to GitHub

After you create the repo, you should be redirected to a page that tells you how to create a repo locally or push an existing one.

In our case, the project already exists locally so we will use commands in the "...or push an existing repository from the command line" section. These are the commands:

```
git remote add origin https://github.com/ihechikara/git-and-github-tutorial.git
git branch -M main
git push -u origin main
```

The first command

`git remote add origin`

creates a connection between your local repo and the remote repo on Github.

The URL for your remote project should be entirely different from the one above. So to follow along, make sure you are following the steps and working with your own remote repo. You won't usually get a response after executing this command but make sure you have an internet connection.

The second command

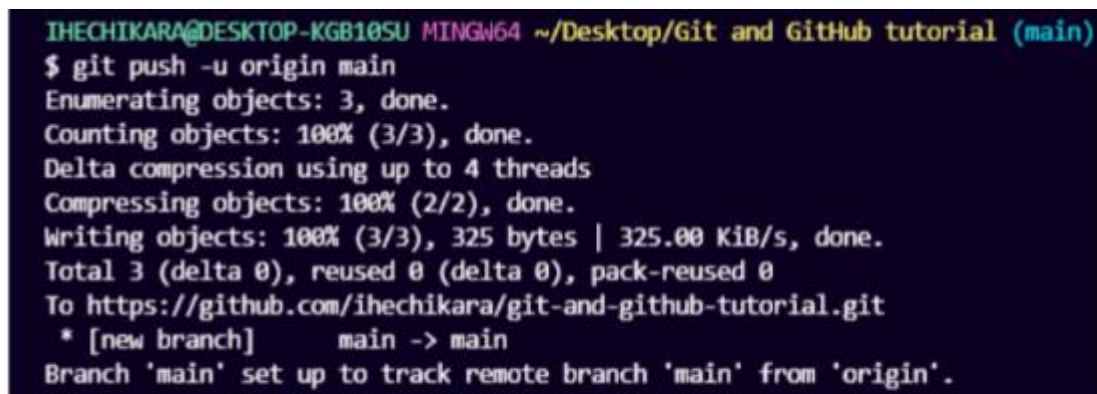
`git branch -M main`

changes your main branch's name to "main". The default branch might be created as "master", but "main" is the standard name for this repo now. There is usually no response here.

The last command

```
git push -u origin main
```

pushes your repo from your local device to GitHub. You should get a response similar to this:

A terminal window with a dark background and light-colored text. The prompt is 'IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)'. The command entered is '\$ git push -u origin main'. The output shows the progress of pushing the main branch to the remote repository, including object enumeration, counting, compression, and writing. It ends with a message indicating that the branch 'main' is now set up to track the remote branch 'main' from 'origin'.

```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 325 bytes | 325.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ihechikara/git-and-github-tutorial.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

git push

To help you deepen your understanding of file stages, I will make changes to the file and then push the new version to GitHub.

Recall that our file is now in the committed state. Let's make changes to the file and take note of the states.

I am going to add a new task to the to-do list:

MY TO-DO LIST

1. Write an article.
2. Code.
3. Study books.
4. Attend classes on time.
5. Visit aunt.
6. Apply for remote jobs.
7. Practice code

After adding the new task, run the git status command. This is what you should see:

```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   todo.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

git status

After making changes to the file, it moved to the modified state – but it's not yet staged for commit, so you can't push it to GitHub yet. Git has not taken a final snapshot of this current state as it's only comparing the changes we have made now with the last snapshot.

Now we are going to add (stage) this file and then commit and push it. This is the same as in the last section.

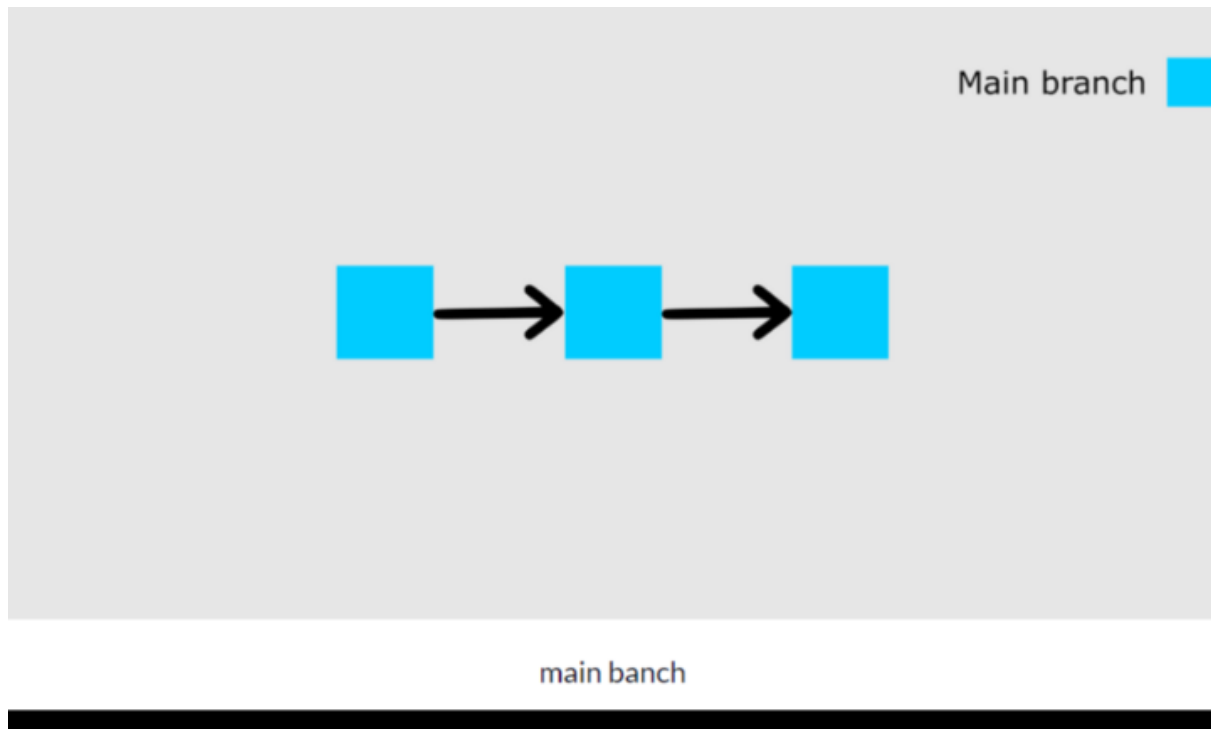
We first add the file by using git add . which adds all the files in the folder (one file in our case). Then we commit the file by running git commit -m "added new task" followed by git push -u origin main.

Those are the three steps to pushing your modified files to GitHub. You add, commit, and then push. I hope you now understand file stages and the commands associated with them.

How to Use Branches in Git

With branches, you can create a copy of a file you would like to work on without messing up the original copy. You can either merge these changes to the original copy or just let the branch remain independent.

Before we go into using branches, I want to show you a visual representation of our repo which looks like this:



The image above shows our main branch with the last two commits (the first commit and the added new task commit).

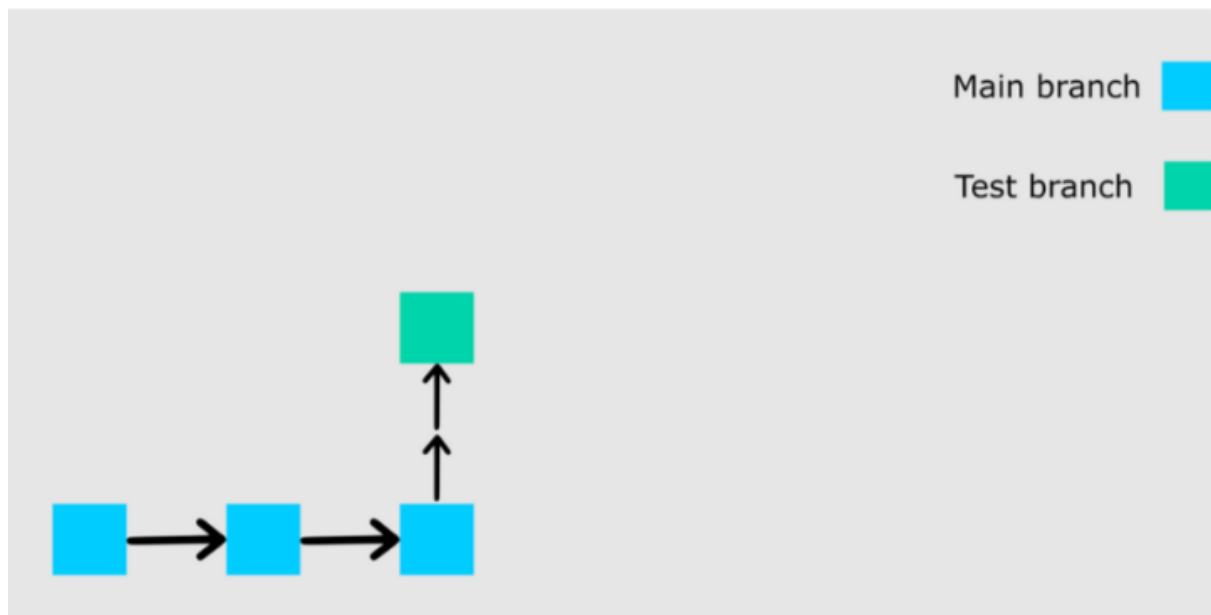
At this point, I want to add more tasks to the list but I am not yet sure whether I want them on my main list. So I will create a new branch called test to see what my list would look like with more tasks included.

To create a new branch, run this command: `git checkout -b test`. I will break it down. `checkout` tells Git it is supposed to switch to a new branch. `-b` tells Git to create a new branch. `test` is the name of the branch to be created and switched to. Here is the response you should get:

```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git checkout -b test
Switched to a new branch 'test'
```

`git checkout -b`

Now that we have a new branch created, this is what our repo will look like:



git branch

We created the new branch from the state of our last commit. Let's now add more tasks to this new branch.

MY TO-DO LIST

1. Write an article.
2. Code.
3. Study books.
4. Attend classes on time.
5. Visit aunt.
6. Apply for remote jobs.
7. Practice code
8. Complete internship task.
9. Practice chess openings.
10. Solve chess puzzles.
11. Check exam schedule.

I have added four new tasks. To merge the new state to the main branch, you have to first stage and commit this branch. I will not go into details about how to do this as we did it twice in the last section.

You should try doing it yourself so you understand how it works. As a hint, add the file and then commit with a message (refer to the previous section for details showing you how to do that).

After committing your test branch, switch back to the main branch by running this command: `git checkout main`.

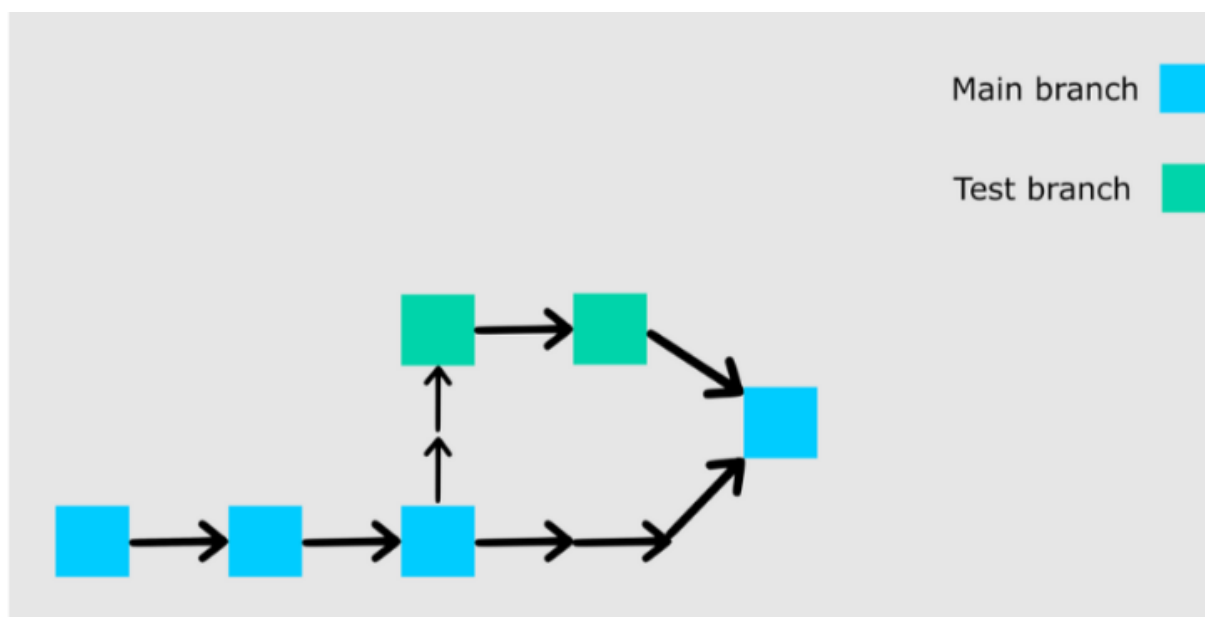
Did you notice that we did not add -b ? This is because we are not creating a new branch but rather switching to an existing one. You can check all the branches that exist in your repo by running the git branch command.

Now we can merge the changes we made in the test branch into the main branch by running git merge test. At this point, you will see all the changes made in the test branch reflected on the main branch. You should also receive a response similar to this:

```
IHECHIKARA@DESKTOP-KGB18SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git merge test
Updating 61dad8f..33a2410
Fast-forward
 todo.txt | 6 +++++-
 1 file changed, 5 insertions(+), 1 deletion(-)
```

git merge

Here is a visual representation of our repo:



git merge

If you go on to push your repo to GitHub, you'll see that the test branch will not be pushed. It will only remain in your local repo. If you would like to push your test branch, switch to the branch using git checkout test and then run git push -u origin test.

How to Pull a Repository in Git

To pull in Git means to clone a remote repository's current state into your computer/repository. This comes in handy when you want to work on your repo from a different computer or when you are contributing to an open source project online.

To test this, don't worry about switching to a new computer. Just run `cd ..` to leave the current directory and go back one step. In my own case, I have navigated back to my desktop. Go to GitHub, and on your repository's main page you should see a green button that says "Code". When you click on the button, you should see some options in a dropdown menu. Go on and copy the HTTPS URL.

After that, run `git clone YOUR_HTTPS_URL`. This command pulls the remote repository into your local computer in a folder called `git-and-git-tutorial`. That is:

```
IHECHIKARA@DESKTOP-KGB18SU MINGW64 ~/Desktop
$ git clone https://github.com/ihechikara/git-and-github-tutorial.git
Cloning into 'git-and-github-tutorial'...
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 14 (delta 2), reused 10 (delta 1), pack-reused 0
Receiving objects: 100% (14/14), done.
Resolving deltas: 100% (2/2), done.
```

git clone

Overview of GIT:

Basic Command to perform operations in GIT follow:

SETUP Configuring user information used across all local repositories :

SETUP

Configuring user information used across all local repositories

git config --global user.name "[firstname lastname]"

set a name that is identifiable for credit when review version history
--

git config --global user.email "[valid-email]"

set an email address that will be associated with each history marker

git config --global color.ui auto
--

set automatic command line coloring for Git for easy reviewing
--

SETUP & INIT

Configuring user information, initializing and cloning repositories

git init

initialize an existing directory as a Git repository
--

git clone [url]

retrieve an entire repository from a hosted location via URL
--

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

git status

show modified files in working directory, staged for your next commit

git add [file]

add a file as it looks now to your next commit (stage)
--

git reset [file]

unstage a file while retaining the changes in working directory

git diff

diff of what is changed but not staged
--

git diff --staged

diff of what is staged but not yet committed
--

git commit -m "[descriptive message]"
--

commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

git branch
list your branches. a * will appear next to the currently active branch
git branch [branch-name]
create a new branch at the current commit
git checkout
switch to another branch and check it out into your working directory
git merge [branch]
merge the specified branch's history into the current one
git log
show all commits in the current branch's history

INSPECT & COMPARE

Examining logs, diffs and object information

git log
show the commit history for the currently active branch
git log branchB..branchA
show the commits on branchA that are not on branchB
git log --follow [file]
show the commits that changed file, even across renames
git diff branchB..branchA
show the diff of what is in branchA that is not in branchB
git show [SHA]
show any object in Git in human-readable format

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

git remote add [alias] [url]
add a git URL as an alias
git fetch [alias]
fetch down all the branches from that Git remote
git merge [alias]/[branch]
merge a remote branch into your current branch to bring it up to date
git push [alias] [branch]
Transmit local branch commits to the remote repository branch
git pull
fetch and merge any commits from the tracking remote branch

TRACKING PATH CHANGES

Versioning file removes and path changes

git rm [file]
delete the file from project and stage the removal for commit
git mv [existing-path] [new-path]
change an existing file path and stage the move
git log --stat -M
show all commit logs with indication of any paths that moved

IGNORING PATTERNS

Preventing unintentional staging or committing of files

logs/ *.notes pattern*/
Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.
git config --global core.excludesfile [file]
system wide ignore pattern for all local repositories

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

git rebase [branch]
apply any commits of current branch ahead of specified one
git reset --hard [commit]
clear staging area, rewrite working tree from specified commit

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

git stash
Save modified and staged changes
git stash list
list stack-order of stashed file changes
git stash pop
write working from top of stash stack
git stash drop
discard the changes from top of stash stack

4. Jenkins installation and setup, explore the environment.

There are 6 Steps to Install Jenkins on Windows

1. Install Java Development Kit (JDK)

- Download [JDK 8](#) and choose windows 32-bit or 64-bit according to your system configuration. Click on "accept the license agreement."

2. Set the Path for the Environmental Variable for JDK

- Go to System Properties. Under the "Advanced" tab, select "Environment Variables."
- Under system variables, select "new." Then copy the path of the JDK folder and paste it in the corresponding value field. Similarly, do this for JRE.
- Under system variables, set up a bin folder for JDK in PATH variables.
- Go to command prompt and type the following to check if [Java](#) has been successfully installed:

```
C:\Users\Simplilearn>java -version
```

3. Download and Install Jenkins

- [Download Jenkins](#).
- <https://www.jenkins.io/download/>
- Under LTS, click on windows.
- After the file is downloaded, unzip it. Click on the folder and install it. Select "finish" once done.

4. Run Jenkins on Localhost 8080

- Once Jenkins is installed, explore it. Open the web browser and type "localhost:8080".
- Enter the credentials and log in. If you install Jenkins for the first time, the dashboard will ask you to install the recommended plugins. Install all the recommended plugins.

5. Jenkins Server Interface

- New Item allows you to create a new project.
- Build History shows the status of your builds.
- Manage System deals with the various configurations of the system.

6. Build and Run a Job on Jenkins

- Select a new item (Name - Jenkins_demo). Choose a freestyle project and click Ok.
- Under the General tab, give a description like "This is my first Jenkins job." Under the "Build Triggers" tab, select add built step and then click on the "Execute Windows" batch command.
- In the command box, type the following: `echo "Hello... This is my first Jenkins Demo: %date%: %time% "`. Click on apply and then save.
- Select build now. You can see a building history has been created. Click on that. In the console output, you can see the output of the first Jenkins job with time and date.

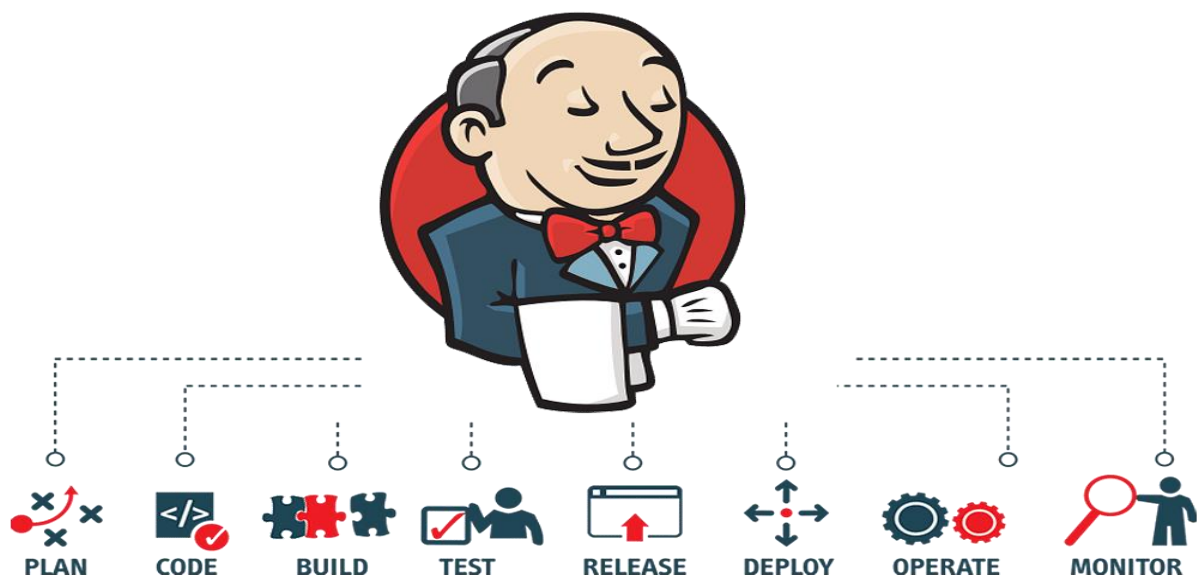
Congratulations, you've just installed Jenkins on your Windows system!

5. Demonstrate continuous integration and development using Jenkins.

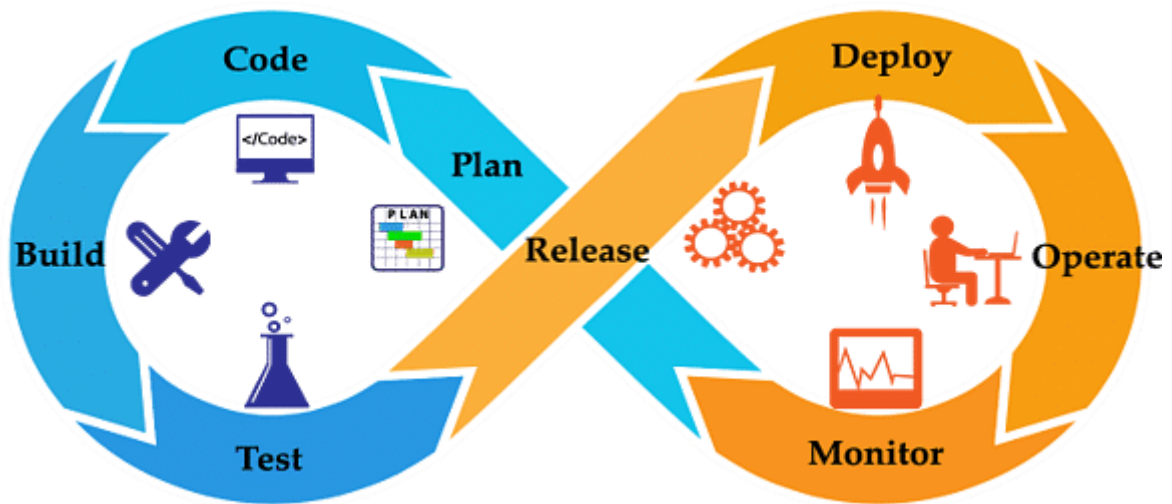
What is a CI/CD Pipeline?

The term “CI/CD pipeline” refers to the Continuous Integration/Continuous Delivery pipeline. Before we dive deep into this segment, let's first understand what is meant by the term ‘pipeline.’

A pipeline is a concept that introduces a series of events or tasks that are connected in a sequence to make quick software releases. For example, there is a task, that task has got five different stages, and each stage has got several steps. All the steps in phase one must be completed, to mark the latter stage to be complete.



Now, consider the CI/CD pipeline as the backbone of the DevOps approach. This Pipeline is responsible for building codes, running tests, and deploying new software versions. The Pipeline executes the job in a defined manner by first [coding](#) it and then structuring it inside several blocks that may include several steps or tasks.



Before we tell you how to build this CI/CD pipeline using Jenkins, let us investigate a brief meaning of these two terms: Continuous Integration and Continuous Delivery.

What is Continuous Integration (CI)?

[Continuous Integration](#) is a practice that integrates code into a shared repository. It uses automated verifications for the early detection of problems. Continuous Integration doesn't eliminate bugs but helps in finding and removing them quickly.

What is Continuous Delivery (CD)?

Continuous Delivery is the phase where the changes are made in the code before deploying. The team in this phase decides what is to be deployed to the customers and when. The final goal of the pipeline is to make deployments.

When both these practices come together, all the steps are considered automated, resulting in the process we know as CI/CD. Implementation of CI/CD enables the team to deploy codes quickly and efficiently. The process makes the team more agile, productive, and confident.

Jenkins is the DevOps tool that is most used for CI/CD pipelines. Therefore, we must have a look at the basics of Jenkins and understand why it is the most sought-after tool to build this pipeline.

Building CI/CD Pipeline with Jenkins

There are six steps to building a pipeline with Jenkins. But, before you begin those six steps, make sure you have the following in your system.

- Java Development Kit
- Knowledge to execute some basic Linux commands

The steps to build CI/CD pipeline with Jenkins are:

1. Download Jenkins

- [Download Jenkins](https://www.jenkins.io/download/) from the Jenkins downloads page '<https://www.jenkins.io/download/>'.
- Download the file 'Generic Java package (.war)'.

Download Jenkins 2.250 for:

Generic Java package (.war)

SHA-256: cdd74f5a7e08499e40480a866adabc8429c81f2ab6f1255cba1be2bd94e2f500


Docker

Ubuntu/Debian


CentOS/Fedora/Red Hat

Windows

openSUSE

Arch Linux 

FreeBSD 

Gentoo 

macOS 

2. Execute Jenkins as a Java binary

- Open the terminal window and enter `cd <your path>`.
- Use the command `java -jar ./Jenkins.war` to run the WAR file.

3. Create a Jenkins Job

- Open the web browser and open `localhost:8080`.
- The Jenkins dashboard opens creates new jobs there.



4. Create a Pipeline Job

- Select and define what Jenkins job that is to be created.
- Select Pipeline, give it a name and click OK.
- Scroll down and find the pipeline section.
- Either directly write a pipeline script or retrieve the Jenkins file from SCM (Source Code Management).

5. Configure and Execute a Pipeline Job With a Direct Script

- Choose Pipeline script as the Destination and paste the Jenkins file content in the Script from the GitHub.
- Click on Save to keep the changes.
- Now click on the Build Now to process the build.

- To check the output, click on any stage and click Log; a message will appear on the screen.

6. Configure and Execute a Pipeline With SCM

- Copy the [GitHub repository](#) URL by clicking on Clone or download.
- Now, click on Configure to modify the existing job.
- Scroll to the Advanced Project Options setting and select Pipeline script from the SCM option.
- Paste the GitHub repository URL here.
- Type Jenkinsfile in the Script, and then click on the Save button.
- Next, click on Build Now to execute the job again.
- There will be an additional stage, in this case, i.e., Declaration: Checkout SCM.
- Click on any stage and click on Log.

After you have grasped all the essential steps to build a CI/CD pipeline using Jenkins, a hands-on demonstration will serve as the icing on the cake.

Demo - To Build a CI/CD Pipeline With Jenkins


Go to your Jenkins Portal:

- Click on 'Create a job'.
- In the item name dialog box, you may enter the 'pipeline'.
- Select the pipeline job type in the list below.
- Click on OK.

Enter an item name


Pipeline

» Required field




Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.




Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

A configuration related to the pipeline opens on the screen.

- Scroll down on that page.
- There in the dialog box, choose GitHub+Maven.

Pipeline

Definition Pipeline script

Script

```
1 pipeline {
2   agent any
3
4   tools {
5     // Install the Maven version configured as "M3" and add it to the path.
6     maven "M3"
7   }
8
9   stages {
10    stage('Build') {
11      steps {
12        // Get some code from a GitHub repository
13        git 'https://github.com/jglick/simple-maven-project-with-tests.git'
14
15        // Run Maven on a Unix agent.
16        sh "mvn -Dmaven.test.failure.ignore=true clean package"
17      }
18    }
19  }
20 }
```

☒ Use Groovy Sandbox

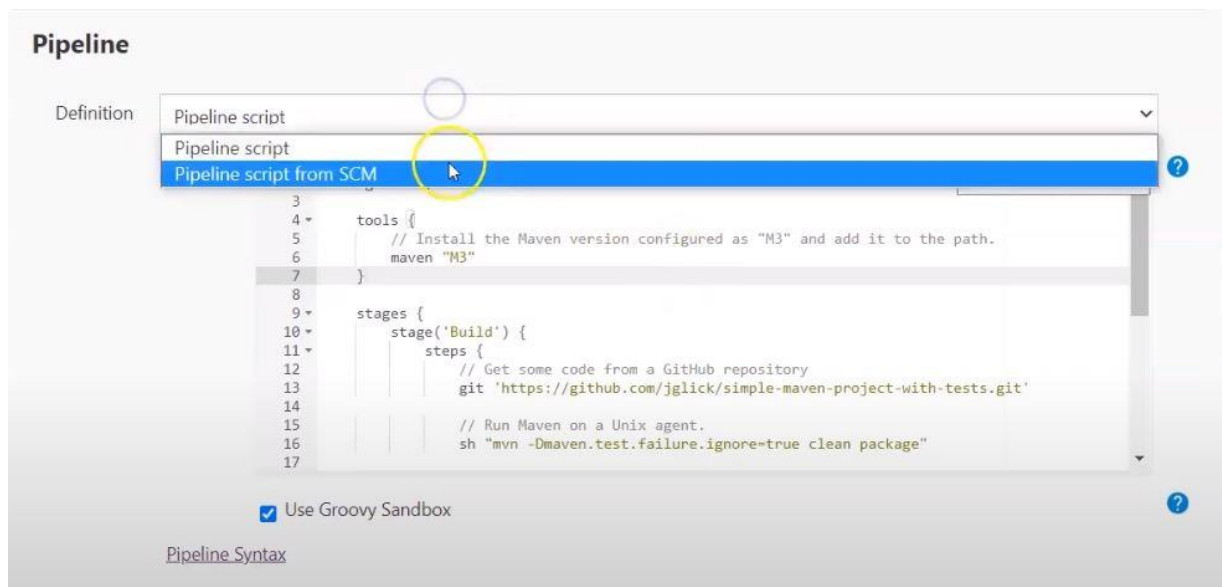
[Pipeline Syntax](#)

GitHub + Maven

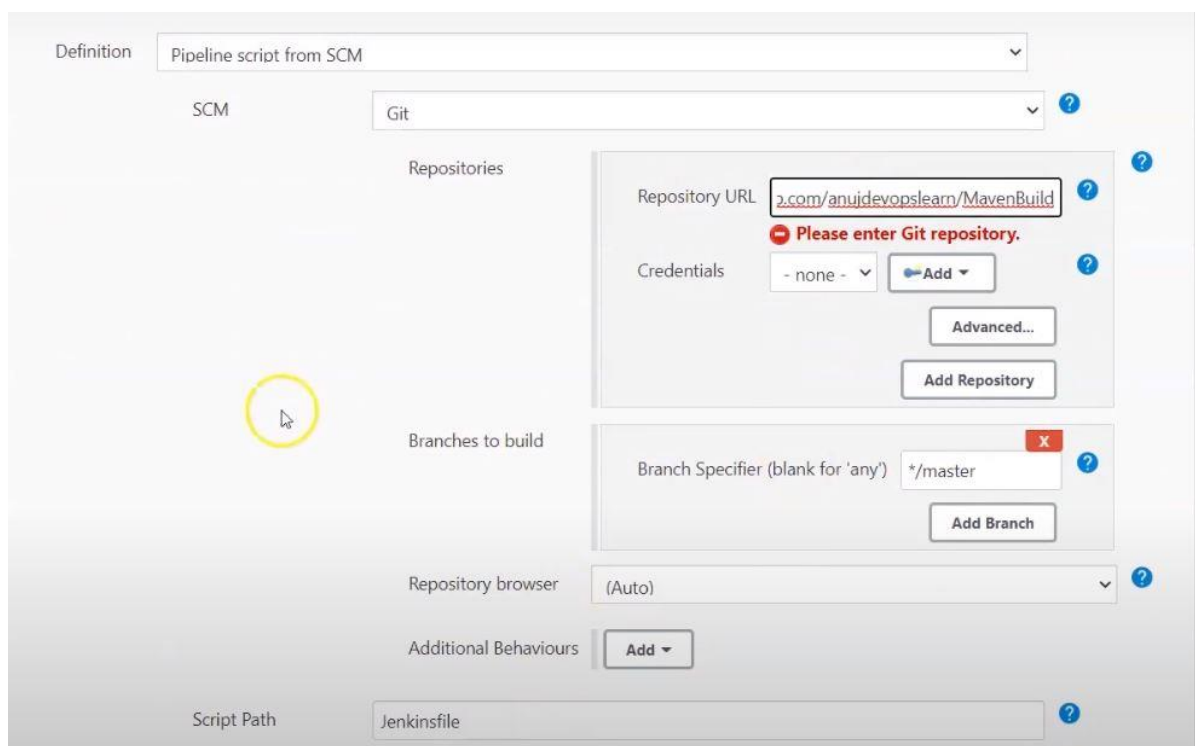
Some steps will appear on the screen. The next step is to integrate the Jenkins file into the Version Control system.

So, to do that, you must:

- Select 'Pipeline script from SCM'.



- Then in the SCM dialog box, select Git.
- 'Jenkins file' is the name of the Script.
- Add the Git repository URL.
- You can add the credentials if any.



The credentials can be added with the help of the 'Add' option.

- Then save the configuration

A page now appears on the screen that gives you various options like 'Build Now', 'Delete Pipeline', 'Configure', etc.

- Click on the Build Now option.

The pipeline will start downloading. The checkout will be visible on the screen and you can see the build being complete on the screen.

checkout code	Build	Test Cases Execution	Sonar Analysis	Archive Artifacts	Deployment	Notification
1s	15s	12s	164ms	324ms	106ms	8s

You can go to the console output option to check the log that is taking place.

You will soon be able to see that all the segments of the pipeline are completed. The artifact will be present to download. The war file can be downloaded using that link.

The entire process helps us understand how the whole Pipeline is configured. Using similar types of steps, different kinds of automation pipelines can be configured.

Using CI/ CD Pipelines and IndexedDB for a jQuery UI Web Application

Using a simple [PostgreSQL](#) database, we'll be storing some client-side data in IndexedDB storage. This technique can be a convenient and robust solution for storing some data in the browser.

Using IndexedDB is a much better alternative to the Document Object Model (DOM) for storing and querying client-side data because of how easy it is to manage and query data. Let's get started with our project.

Creating a New Project

First, create a new directory for our project in a new directory and run this command:

```
git init
```

You should see something like the following:

```
$ mkdir jquery-ui-pipeline-demo $ cd jquery-ui-pipeline-demo $ bin/pipeline create
```

The "Pipeline create" command lets you specify the template where your code will be built and tested. This method is like how a knitter uses yarn but in our case, we're using the web toolbelt instead of the yarn.

The only difference is that you won't need to use your own build environment to run tests. The web toolbelt will do this for you automatically.

That's all you need to do to create a new project.

Building a jQuery UI Web App Using a Pipeline

To use the pipeline to build your sample app, we will open a new terminal window and type:

```
$ bin/pipeline build
```

This command will take a while because it's building and testing the app on our local development machine.

After the build process is complete, you can see that the completed project is stored in our `index.html` file. We can use these files in other apps or even deploy the app to a server to test it out.

6. Explore Docker commands for content management.

- Docker image is made of a Dockerfile + any necessary dependencies
- Docker container is a Docker image brought to life.
- To work with Docker commands, we first need to know whether we are dealing with an image or a container.
 - **A Docker image either exists or it doesn't.**
 - **A Docker container either exists or it doesn't.**
 - **A Docker container that exists is either running or it isn't.**

Command Commonalities

Here are a few things to know about Docker commands:

- Docker CLI management commands start with `docker`, then a space, then the management category, then a space, and then the command. For example, `docker container stop` stops a container.
- A command referring to a specific container or image requires the name or id of that container or image.

For example, `docker container run my_app` is the command to build and run the container named *my_app*. I'll use the name *my_container* to refer to a generic container throughout the examples. Same goes for *my_image*, *my_tag*, etc.

I'll provide the command alone and then with common flags, if applicable. A flag with two dashes in front is the full name of the flag. A flag with one dash is a shortcut for the full flag name. For example, `-p` is short for the `--port` flag.

Containers

Use `docker container my_command`

`create` — Create a container from an image.

`start` — Start an existing container.

`run` — Create a new container and start it.

`ls` — List running containers.

`inspect` — See lots of info about a container.

`logs` — Print logs.

`stop` — Gracefully stop running container.
`kill` — Stop main process in container abruptly.
`rm` — Delete a stopped container.

Images

Use `docker image my_command`

`build` — Build an image.
`push` — Push an image to a remote registry.
`ls` — List images.
`history` — See intermediate image info.
`inspect` — See lots of info about an image, including the layers.
`rm` — Delete an image.

Misc

`docker version` — List info about your Docker Client and Server versions.
`docker login` — Log in to a Docker registry.
`docker system prune` — Delete all unused containers, unused networks, and dangling images.

Containers

Container Beginnings

The terms create, start, and run all have similar semantics in everyday life. But each is a separate Docker command that creates and/or starts a container. Let's look at creating a container first.

`docker container create my_repo/my_image:my_tag` — Create a container from an image.

I'll shorten `my_repo/my_image:my_tag` to `my_image` for the rest of the article.

There are [a lot of possible flags](#) you could pass to create.

`docker container create -a STDIN my_image`

`-a` is short for `--attach`. Attach the container to STDIN, STDOUT or STDERR.

Now that we've created a container let's start it.

docker container start my_container — Start an existing container.

Note that the container can be referred to by either the container's ID or the container's name.

docker container start my_container

Images

Here are seven commands for working with Docker images.

Developing Images

docker image build -t my_repo/my_image:my_tag . — Build a Docker image named *my_image* from the Dockerfile located at the specified path or URL.

-t is short for tag. Tells docker to tag the image with the provided tag. In this case *my_tag* .

The **.** (period) at the end of the command tells Docker to build the image according to the Dockerfile in the current working directory.

Removing Images

docker image rm my_image — Delete the specified image. If the image is stored in a remote repository, the image will still be available there.

docker image rm \$(docker images -a -q) — Delete all images.

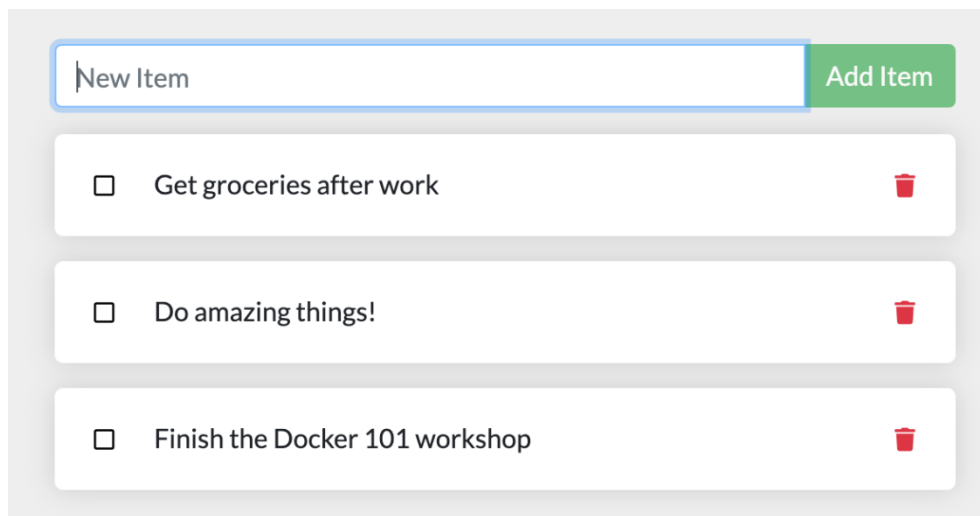
7. Develop a simple containerized application using Docker.

Sample application

Estimated reading time: 5 minutes

For the rest of this tutorial, we will be working with a simple todo list manager that is running in Node.js. If you're not familiar with Node.js, don't worry. No real JavaScript experience is needed.

At this point, your development team is quite small and you're simply building an app to prove out your MVP (minimum viable product). You want to show how it works and what it's capable of doing without needing to think about how it will work for a large team, multiple developers, etc.

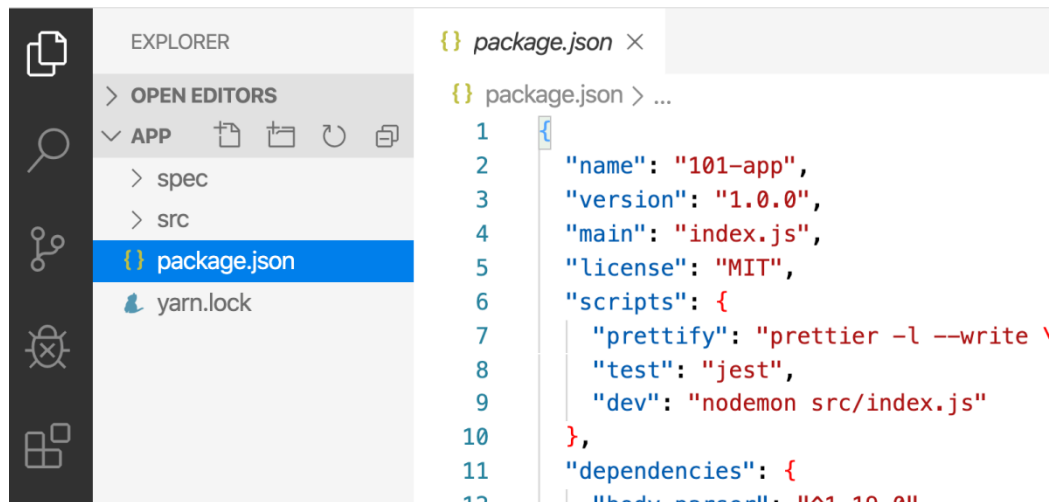
A screenshot of a web application interface for a todo list. At the top, there is a text input field with the placeholder text "New Item" and a green button labeled "Add Item". Below this, there is a list of three items, each in a white box with a light gray border. Each item has a checkbox on the left, the text of the item in the middle, and a red trash icon on the right. The items are: "Get groceries after work", "Do amazing things!", and "Finish the Docker 101 workshop".

Input	Action
New Item	Add Item
<input type="checkbox"/> Get groceries after work	
<input type="checkbox"/> Do amazing things!	
<input type="checkbox"/> Finish the Docker 101 workshop	

Get the app

Before we can run the application, we need to get the application source code onto our machine. For real projects, you will typically clone the repo. But, for this tutorial, we have created a ZIP file containing the application.

1. [Download the App contents](#). You can either pull the entire project or download it as a zip and extract the app folder out to get started with.
2. Once extracted, use your favorite code editor to open the project. If you're in need of an editor, you can use [Visual Studio Code](#). You should see the package.json and two subdirectories (src and spec).



Build the app's container image

In order to build the application, we need to use a Dockerfile. A Dockerfile is simply a text-based script of instructions that is used to create a container image. If you've created Dockerfiles before, you might see a few flaws in the Dockerfile below. But, don't worry. We'll go over them.

1. Create a file named Dockerfile in the same folder as the file package.json with the following contents.
2. `# syntax=docker/dockerfile:1`
3. `FROM node:12-alpine`
4. `RUN apk add --no-cache python2 g++ make`
5. `WORKDIR /app`
6. `COPY . .`
7. `RUN yarn install --production`
8. `CMD ["node", "src/index.js"]`
9. `EXPOSE 3000`

Please check that the file Dockerfile has no file extension like .txt. Some editors may append this file extension automatically and this would result in an error in the next step.

10. If you haven't already done so, open a terminal and go to the app directory with the Dockerfile. Now build the container image using the docker build command.
11. `$ docker build -t getting-started .`

This command used the Dockerfile to build a new container image. You might have noticed that a lot of "layers" were downloaded. This is because we instructed the builder that we wanted to start from the node:12-alpine image. But, since we didn't have that on our machine, that image needed to be downloaded.

After the image was downloaded, we copied in our application and used yarn to install our application's dependencies. The CMD directive specifies the default command to run when starting a container from this image.

Finally, the -t flag tags our image. Think of this simply as a human-readable name for the final image. Since we named the image getting-started, we can refer to that image when we run a container.

The . at the end of the docker build command tells Docker that it should look for the Dockerfile in the current directory.

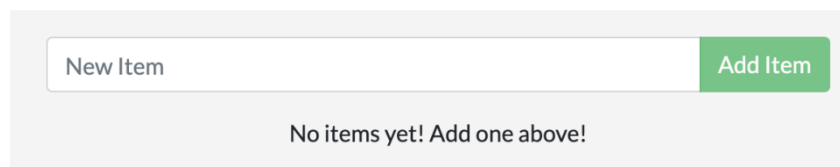
Start an app container

Now that we have an image, let's run the application. To do so, we will use the docker run command (remember that from earlier?).

1. Start your container using the docker run command and specify the name of the image we just created:
2. `$ docker run -dp 3000:3000 getting-started`

Remember the -d and -p flags? We're running the new container in "detached" mode (in the background) and creating a mapping between the host's port 3000 to the container's port 3000. Without the port mapping, we wouldn't be able to access the application.

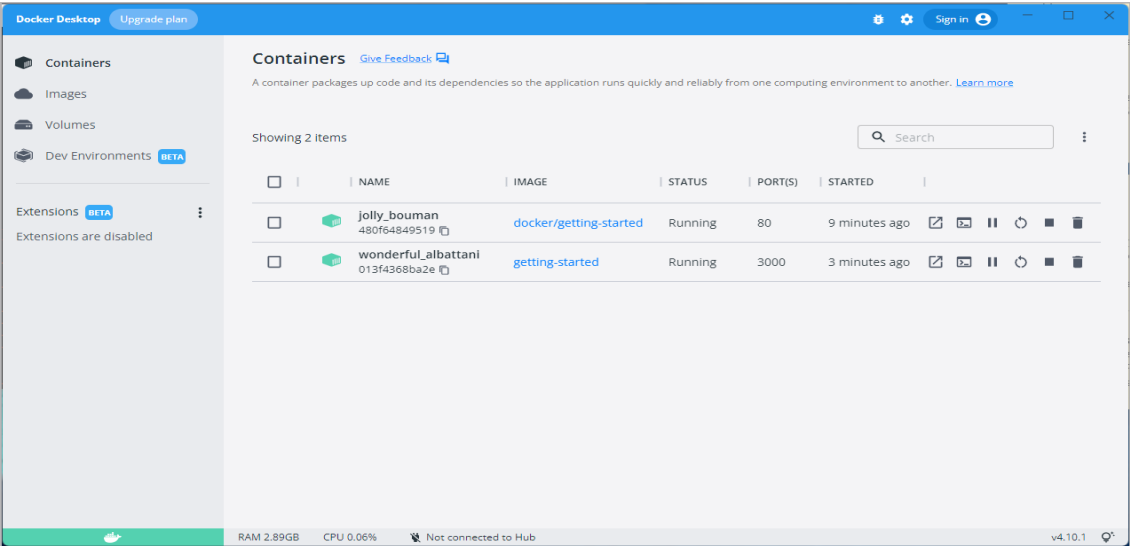
3. After a few seconds, open your web browser to <http://localhost:3000>. You should see our app.



4. Go ahead and add an item or two and see that it works as you expect. You can mark items as complete and remove items. Your frontend is successfully storing items in the backend. Pretty quick and easy, huh?

At this point, you should have a running todo list manager with a few items, all built by you. Now, let's make a few changes and learn about managing our containers.

If you take a quick look at the Docker Dashboard, you should see your two containers running now (this tutorial and your freshly launched app container).



8. Integrating Kubernetes with Dockers.

1. Install Docker Desktop

Docker Desktop is freely available in a community edition, for Windows and Mac. Start by downloading and installing the right version for you:

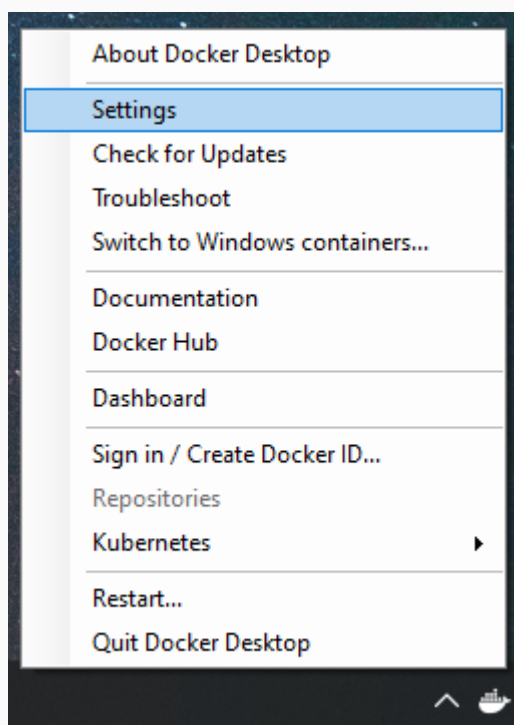
- [Windows 10 \(Professional or Enterprise\)](#)
- [Mac OS X \(Sierra 10.12 minimum\)](#)

Older operating systems can't use Docker Desktop :(You can use [Docker Toolbox](#) instead, but that doesn't come with Kubernetes - so you'll need to run [Kubernetes in Docker](#).

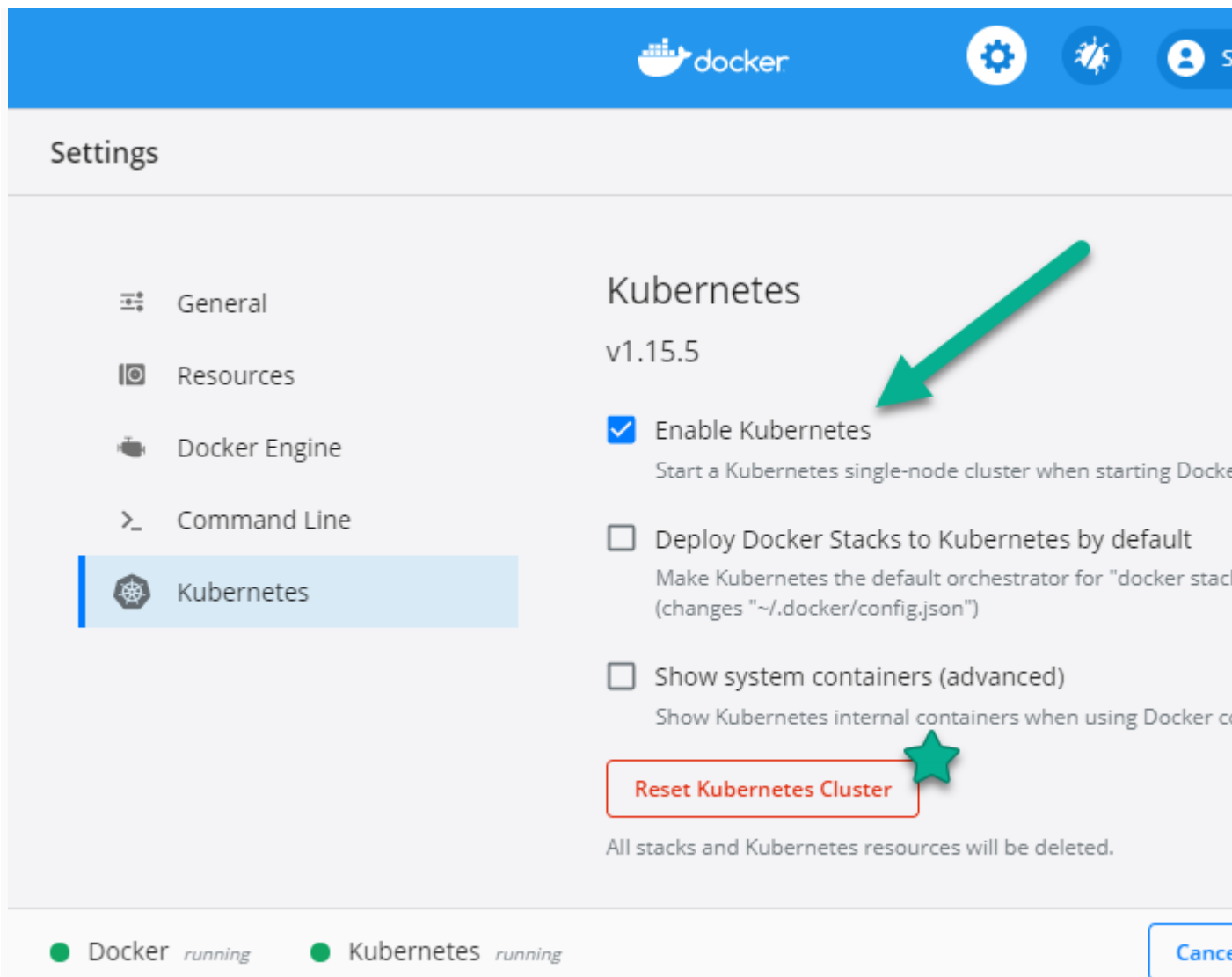
2. Enable Kubernetes

Kubernetes itself runs in containers. When you deploy a Kubernetes cluster you first install Docker (or another container runtime like [containerd](#)) and then use tools like [kubeadm](#) which starts all the Kubernetes components in containers. Docker Desktop does all that for you.

Make sure you have Docker Desktop running - in the taskbar in Windows and the menu bar on the Mac you'll see Docker's whale logo. Click the whale and select *Settings*:



A new screen opens with all of Docker Desktop's configuration options. Click on *Kubernetes* and check the *Enable Kubernetes* checkbox:



That's it! Docker Desktop will download all the Kubernetes images in the background and get everything started up. When it's ready you'll see two green lights in the bottom of the settings screen saying *Docker running* and *Kubernetes running*.

*The star in the screenshot shows the **Reset Kubernetes Cluster** button, which is one of the reasons why Docker Desktop is the best of the local Kubernetes options. Click that and it will reset your cluster back to a fresh install of Kubernetes.*

3. Verify your Kubernetes cluster

If you've worked with Docker before, you're used to managing containers with the **docker** and **docker-compose** command lines. Kubernetes uses a different tool called **kubectl** to manage apps - Docker Desktop installs **kubectl** for you too.

Check the state of your Docker Desktop cluster:

```
kubectl get nodes
```

You should see a single node in the output called **docker-desktop**. That's a full Kubernetes cluster, with a single node that runs the Kubernetes API and your own applications.

The Kubernetes components are running in Docker containers, but Docker Desktop doesn't show them by default to keep things simple when you're running **docker** commands.

Try:

```
docker container ls
```

and you'll see zero containers (unless you have some of your own running).

But try:

```
docker info
```

And you'll see a whole bunch of containers in the running state (18 on my machine), which are the various parts of Kubernetes.

4. Run a familiar application

Let's run the classic [Docker sample voting app](#)! It's a distributed application which uses a Postgres database and Redis message queue, with application components running in Python, .NET and Node.js containers.

All the components of the app are published in public images on Docker Hub. All you need to run it is a Kubernetes manifest - a YAML file which describes all the components of the app. Here's the [voting app definition in Kubernetes](#) which you'll be deploying (compare it to the [voting app definition in Docker Compose](#) if you want to see how Kubernetes is different):

```
kubectl apply -f https://raw.githubusercontent.com/docker/docker-birthday/master/resources/kubernetes-docker-desktop/vote.yaml
```

It'll take a couple of minutes for all the container images to download from Docker Hub and start up.

5. Check the app components

Kubernetes runs containers for you, so instead of explicitly running them with **docker container run**, you describe the desired outcome in a YAML file and when you run **kubectl apply** Kubernetes starts all the containers.

Containers in Kubernetes are wrapped in another object called a [pod](#). Have a look at the pods for the voting app:

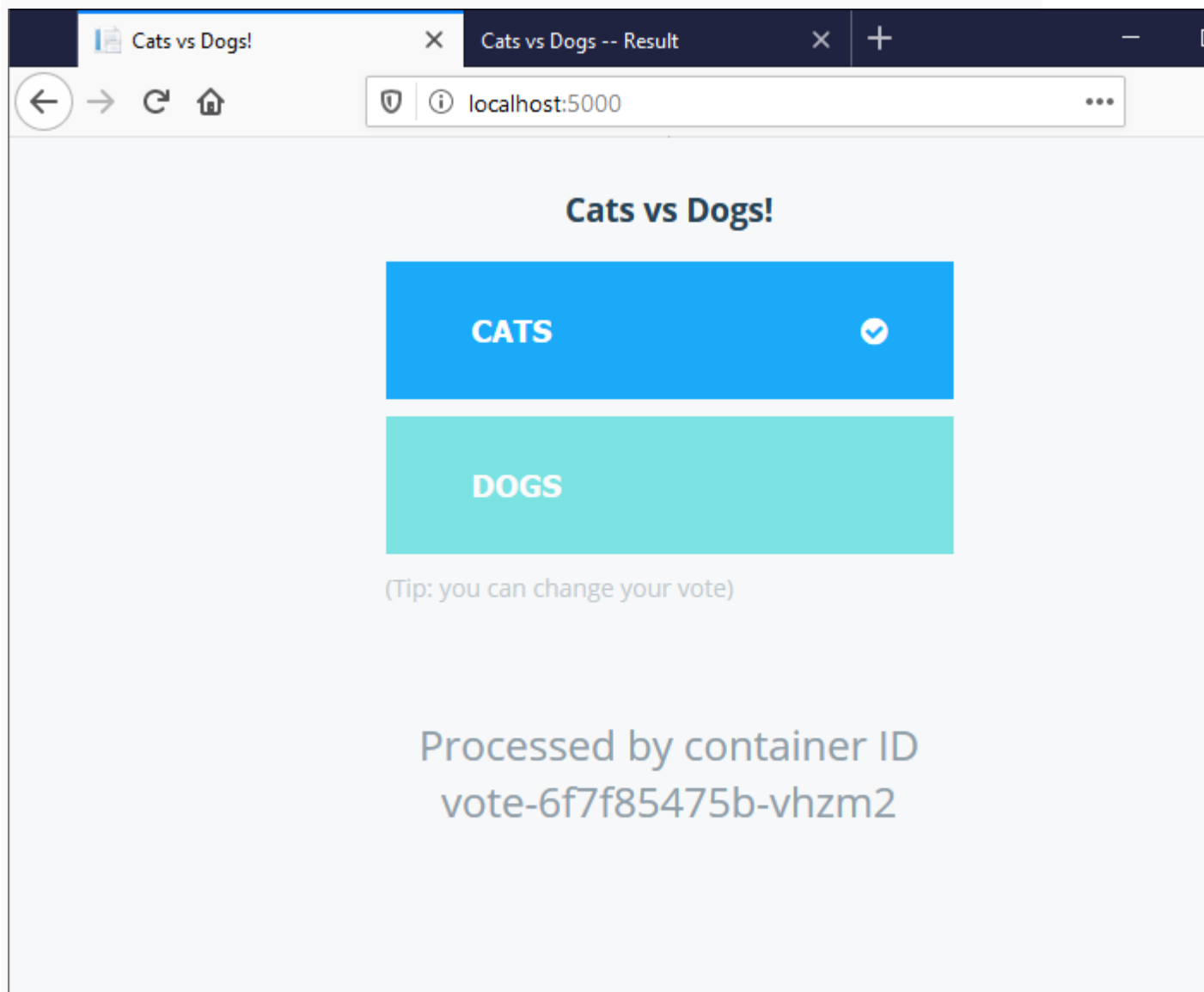
```
kubectl -n vote get pods
```

You should see lots of pods, with names starting `db-`, `redis-` etc. When the `READY` column says `1/1` for every pod, that means all the containers are running.

You can have many containers in one pod in Kubernetes, and they share the same network and compute environment. That lets you do very cool things with the [sidecar pattern](#).

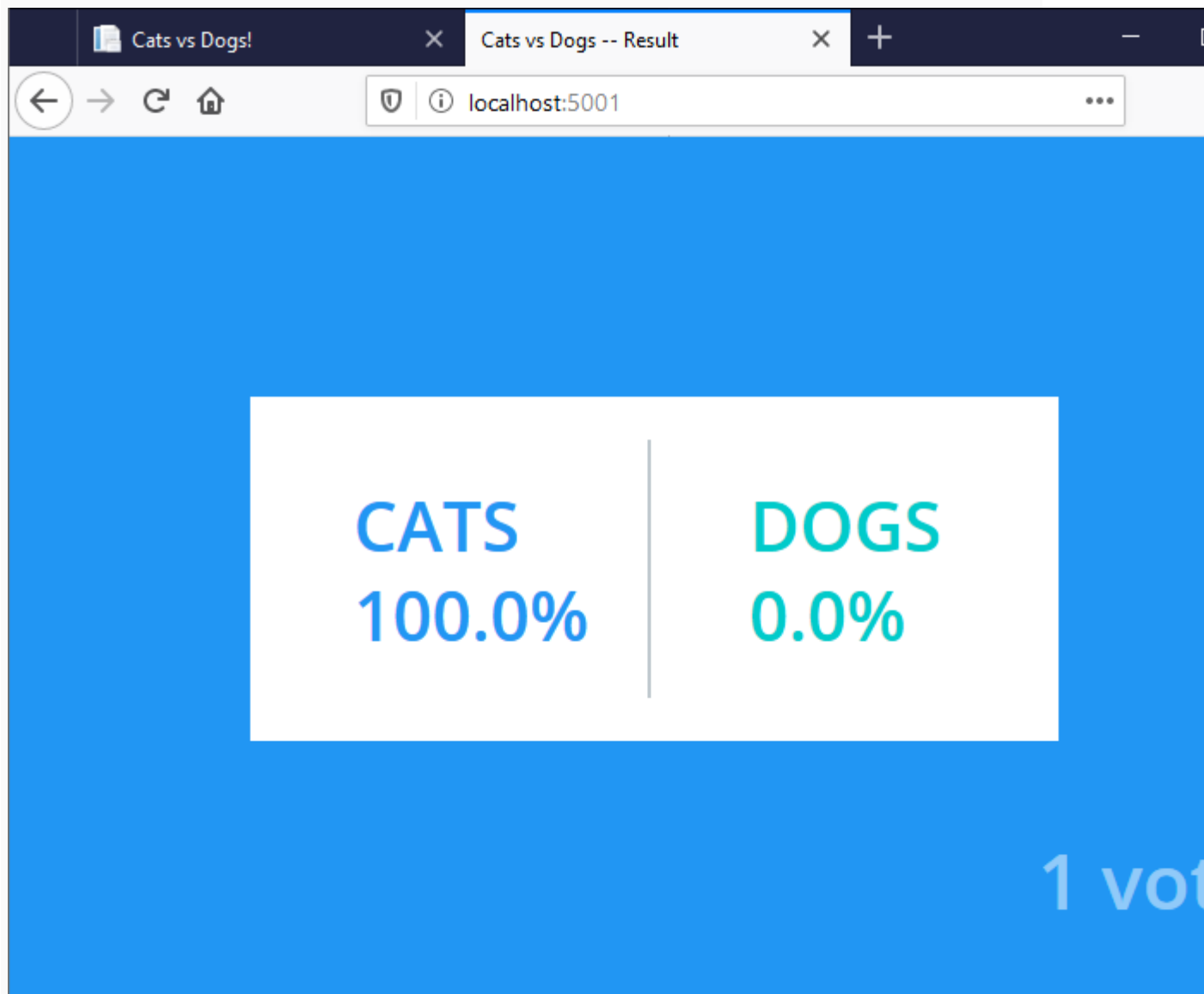
6. Use the app

Browse to `http://localhost:5000` and you should see the classic voting application:



That's a Python application running in a Docker container, being managed by Kubernetes. Click on *Cats* (or *Dogs*) and the app sends a message to the Redis message queue. That message gets picked up by a .NET worker application, which updates a Postgres database.

Browse to `http://localhost:5001` and you'll see the results:



That's a Node.js app which reads the data from Postgres. Everything is running in containers through Kubernetes.

7. Check the resilience

Kubernetes makes sure containers are running to keep your app at the service level you requested in the YAML file. They're all Docker containers which you can also manage with the `docker` command line.

Print the container ID for the result app:

```
docker container ls -f name='k8s_result*' --format '{{.ID}}'
```

And now remove that container:

```
docker container rm -f $(docker container ls -f name='k8s_result*' --format '{{.ID}}')
```

Check back on the result app in your browser at <http://localhost:5001> and you'll see it's still working. Kubernetes saw that the container had been removed and started a replacement straight away.

Print the result container ID again:

```
docker container ls -f name='k8s_result*' --format '{{.ID}}'
```

And you'll see it's a new container. Kubernetes makes sure the running app always matches the desired state in the application YAML file.

This was a simple introduction to Kubernetes, and there's an awful lot more to learn. The example app is a good place to explore, you can read up on [services](#), [volumes](#) and [namespaces](#) - they're all used in this app.

If you want to learn more, check out:

- [The Kubernetes Book](#) by Docker Captain [Nigel Poulton](#)
- the Pluralsight course [Kubernetes for Developers: Core Concepts](#) by Docker Captain [Dan Wahlin](#).

8. Teardown your environment

Docker Desktop has a feature to tear down your whole Kubernetes cluster and reset it to the original state. Open the settings from the Docker whale icon and click *Kubernetes*. Click the *Reset Kubernetes Cluster* button and the demo app will be gone.

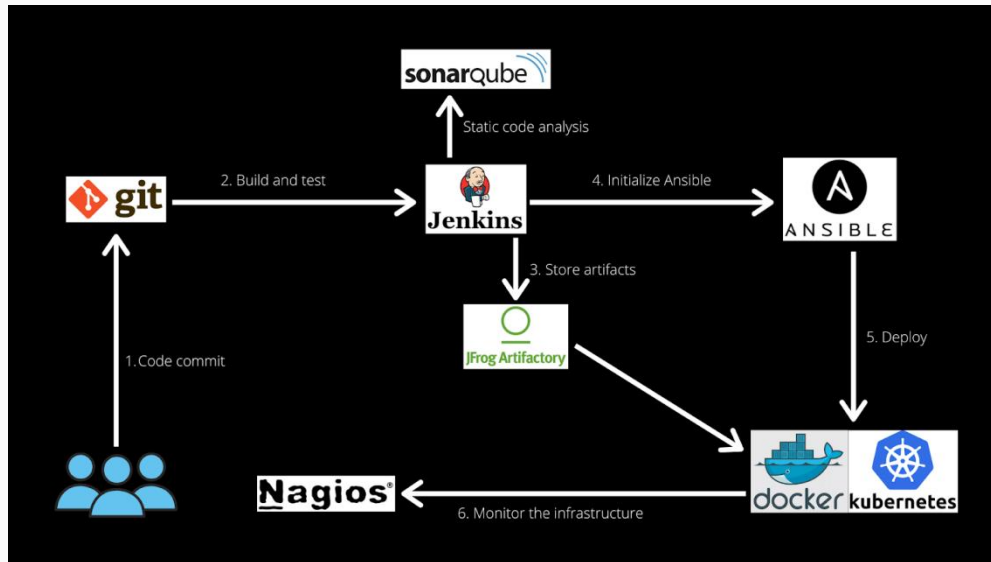
Docker and Kubernetes Work Together:

Docker helps to “create” containers, and Kubernetes allows you to “manage” them at runtime. Use Docker for packaging and shipping the app. Employ Kubernetes to deploy and scale your app. Startups or small companies with fewer containers usually can manage them without having to use Kubernetes, but as the companies grow, their infrastructure needs will rise; hence, the number of containers will increase, which can be difficult to manage. This is where Kubernetes comes into play.

When used together, Docker and Kubernetes serve as digital transformation enablers and tools for modern cloud architecture. Using both has become a new norm in the industry for faster application deployments and releases. While building your stack, it is highly recommended to understand the high-level differences between Docker and Kubernetes.

Let containers help to unhitch the mysteries of cloud computing regardless of the cloud journey you choose.

Let's take a simple scenario of a CI/CD setup using Docker and Kubernetes:



1. The developers' code is pushed into the Git.
2. The build and test happen with Maven in Jenkins for C.
3. Using Ansible as a deployment tool, we will write Ansible playbooks to deploy on AWS.
4. We will introduce JFrog Artifactory as the repository manager after the build process from Jenkins; the artifacts will be stored in Artifactory.
5. Ansible can communicate with Artifactory, take the artifacts and deploy them onto the Amazon EC2 instance.
6. The SonarQube can help in reviewing the code and will give static code analysis.
7. We then introduce Docker as a containerization tool. Just like the way we did on Amazon EC2, we will deploy the app on Docker container by creating a Docker file and Docker images.
8. Once this above setup is done, we will introduce Kubernetes to create a Kubernetes cluster, and by using Docker images, we will be able to deploy.
9. Finally, we will use Nagios to monitor the infrastructure.

9. Automate the process of running containerized application developed in exercise 7 using Kubernetes.

Building Docker images for Kubernetes

The first step to deploying your application to Kubernetes is to build your Docker images. In this guide, I will assume you already have created Docker images in development to create your application, and we will focus on tagging and storing production-ready Docker images in an image repository.

The first step is to run **docker image build**. We pass in `.` as the only argument to specify that it should build using the current directory. This command looks for a **Dockerfile** in your current directory and attempts to build a docker image as described in the Dockerfile.

```
docker image build .
```

If your Dockerfile takes arguments such as **ARG app_name**, you can pass those arguments into the build command:

```
docker image build --build-arg "app_name=MyApp" .
```

You may run into a situation where you want to build your app from a different directory than the current one. This is especially useful if you are managing multiple Dockerfiles in separate directories for different applications which share some common files, and can help you write build scripts to handle more complex builds. Use the **-f** flag, to specify which dockerfile to build with:

```
docker image build -f "MyApp/Dockerfile" .
```

When using this method, be mindful that the paths referenced in your Dockerfile will be relative to the directory passed as the final argument, not the directory the Dockerfile is located in. So in this example, we will build the Dockerfile located at **MyApp/Dockerfile** but all paths referenced in that Dockerfile for COPY and other operations will actually be relative to the current working directory, not **MyApp**.

After your docker image has been built, you will then need to tag your image. Tagging is very important in a docker build and release pipeline since it is the only way to differentiate versions of your application. It is common practice to tag your newest images with the **latest** tag, but this will be insufficient for deploying to Kubernetes since you have to change the tag in your Kubernetes configuration to signal that a new image should be ran. Because of this, I recommend tagging your images with the git commit hash of the current commit. This way you can tie your docker images back to version control to see what has actually been deployed, and you have a unique identifier for each build.

To get the current commit hash programmatically and then tag your image, run:

```
git rev-parse --verify HEAD
```

You can then tag your image like so:


```
docker image tag $IMAGE MyApp:$COMMIT
```

Tagging your image after it is built can be useful for fixing up old images, but you can and should tag them as part of the build command using the **-t** argument. With everything put together, you could write a simple bash script to build and tag your image:

```
#!/bin/bash
COMMIT=$(git rev-parse --verify HEAD)
docker image build -f "MyApp/Dockerfile" . \
  --build-arg "app_name=MyApp" \
  -t "MyApp:latest" \
  -t "MyApp:${COMMIT}"
```

Where to store Docker images

Now that you have your Docker images built and tagged, you need to store them somewhere besides on your laptop. Your Kubernetes cluster needs a fast and reliable Docker repository from which to pull your images, and there are many options for this.

One of the most popular Docker image repositories is [dockerhub](#). For open source projects or public repositories, dockerhub is completely free. For private repositories, dockerhub has very reasonable [pricing](#).

To push images to dockerhub, you must tag your images with the name of the dockerhub repository you created, and then push each tag. Here is an example of tagging and pushing the latest image built above:

```
docker image tag MyApp:latest myrepo/MyApp:latest
docker login
docker push myrepo/MyApp:latest
```

For anyone already using Amazon Web Services, [Amazon Elastic Container Registry](#) provides cheap and private docker repositories. You can similarly tag and push docker images to your ECR repository if you have the AWS CLI installed. Just replace **ECR_URL** in the following example with the actual URL for your ECR repository, which can be viewed in the AWS Web Console.

```
docker image tag MyApp:latest ECR_URL/MyApp:latest
eval $(aws ecr get-login --no-include-email)
docker push ECR_URL/MyApp:latest
```

Deploying Docker images with kubectl

Now that you have built and pushed your Docker images, you can deploy them to your Kubernetes cluster. The quickest way to get started is by using **kubectl**.

create a Deployment in your cluster

kubectl edit

Managing Kubernetes configuration files

Manage Kubernetes deployments using

kubectl edit

```
kubectl apply -f my_app_deployment.yaml
```

Then you can commit the changes you've made and have a history of every change in your Kubernetes cluster, and an easy way to recreate your entire setup if needed.

10. Install and Explore Selenium for automated testing.

Selenium installation is a 3 step process:

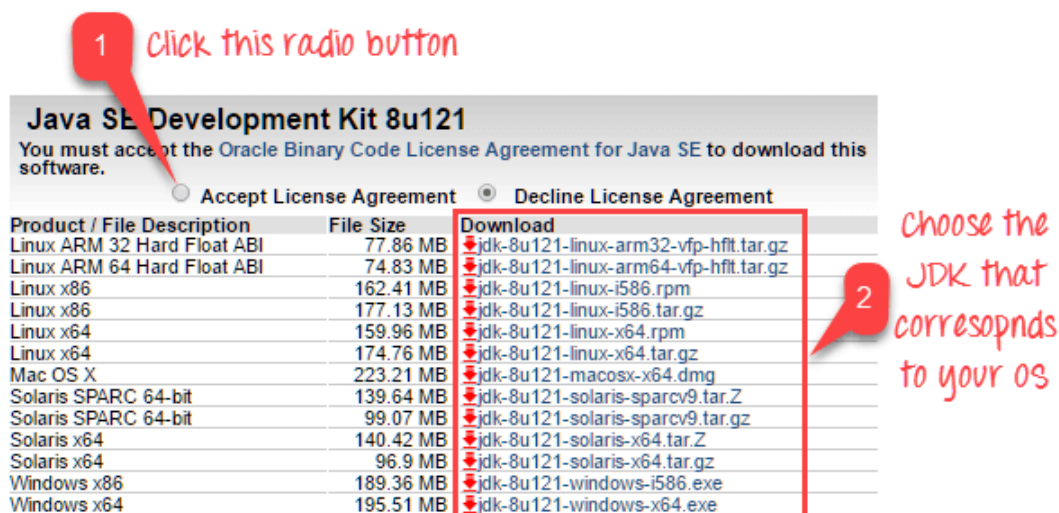
1. Install Java SDK
2. Install Eclipse
3. Install Selenium Webdriver Files

Step 1 – Install Java on your computer

Download and install the **Java Software Development Kit (JDK)** [here](#).

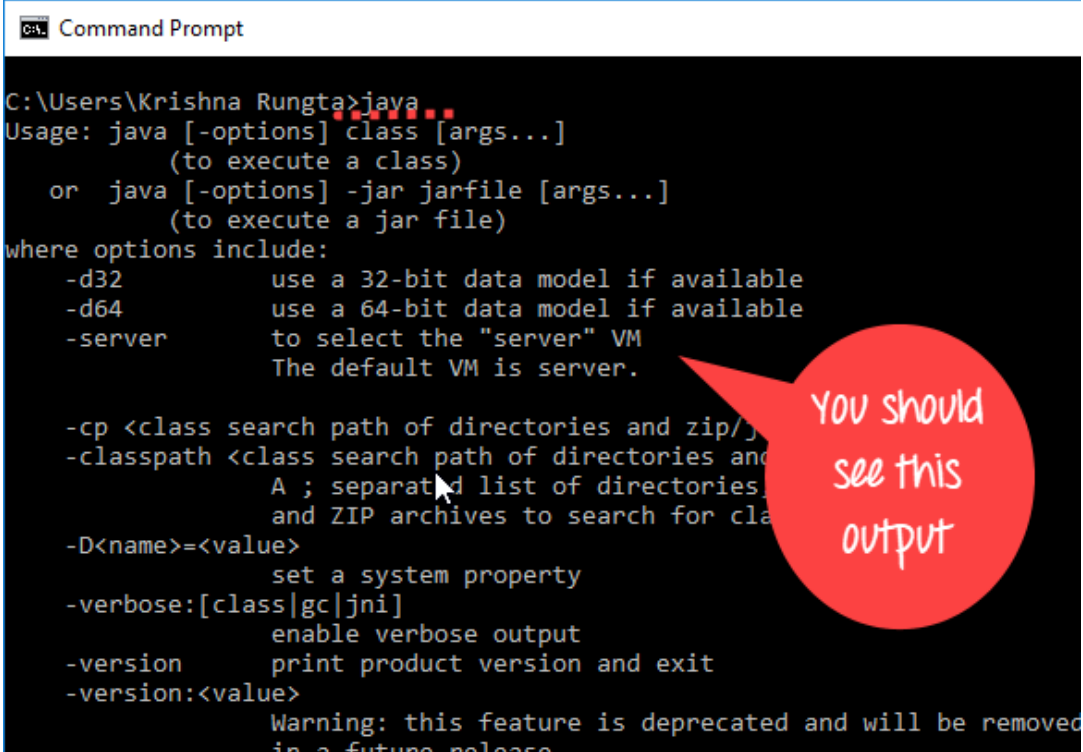


Next –



This JDK version comes bundled with Java Runtime Environment (JRE), so you do not need to download and install the JRE separately.

Once installation is complete, open command prompt and type “java”. If you see the following screen you are good to move to the next step



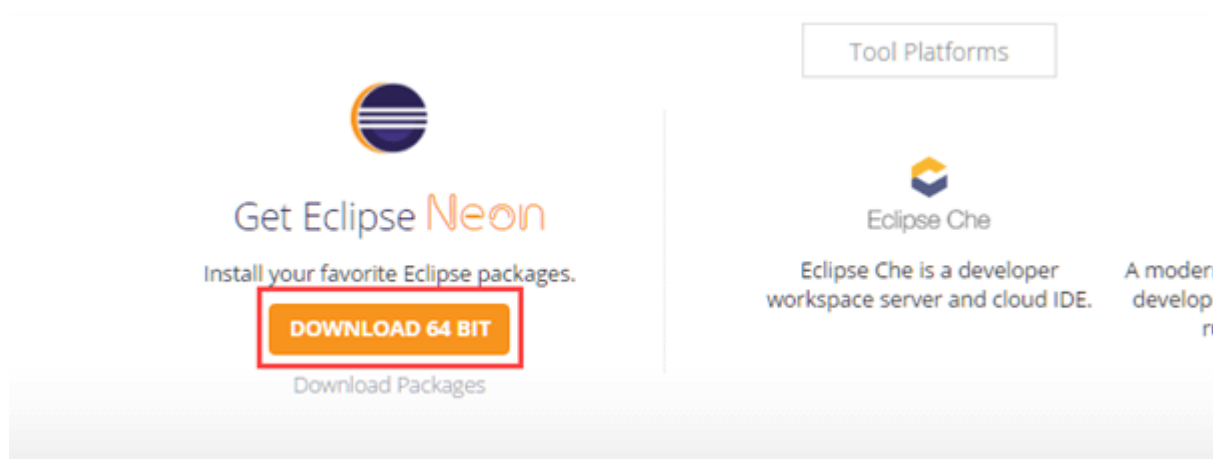
```
C:\Users\Krishna Rungta>java
Usage: java [-options] class [args...]
           (to execute a class)
   or java [-options] -jar jarfile [args...]
           (to execute a jar file)
where options include:
    -d32          use a 32-bit data model if available
    -d64          use a 64-bit data model if available
    -server       to select the "server" VM
                  The default VM is server.

    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A ; separated list of directories, zip archives, and
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose:[class|gc|jni]
                  enable verbose output
    -version       print product version and exit
    -version:<value>
                  Warning: this feature is deprecated and will be removed
                  in a future release.
```

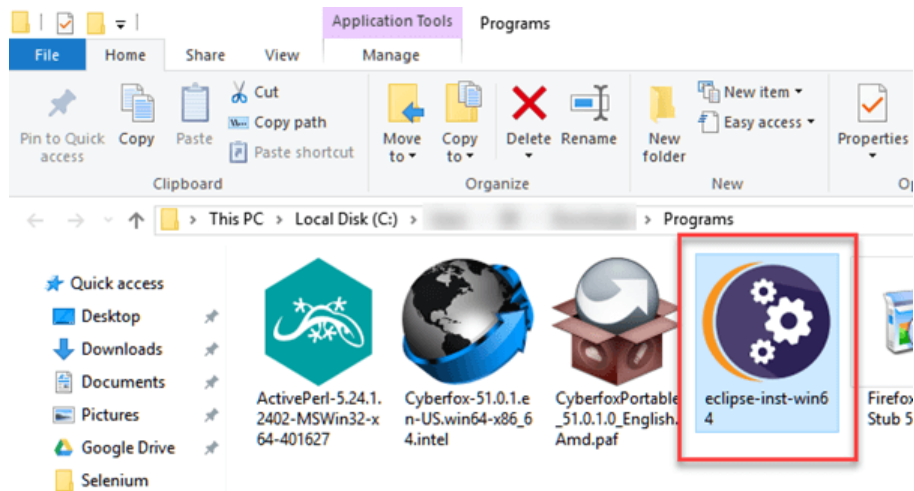
You should see this output

Step 2 – Install Eclipse IDE

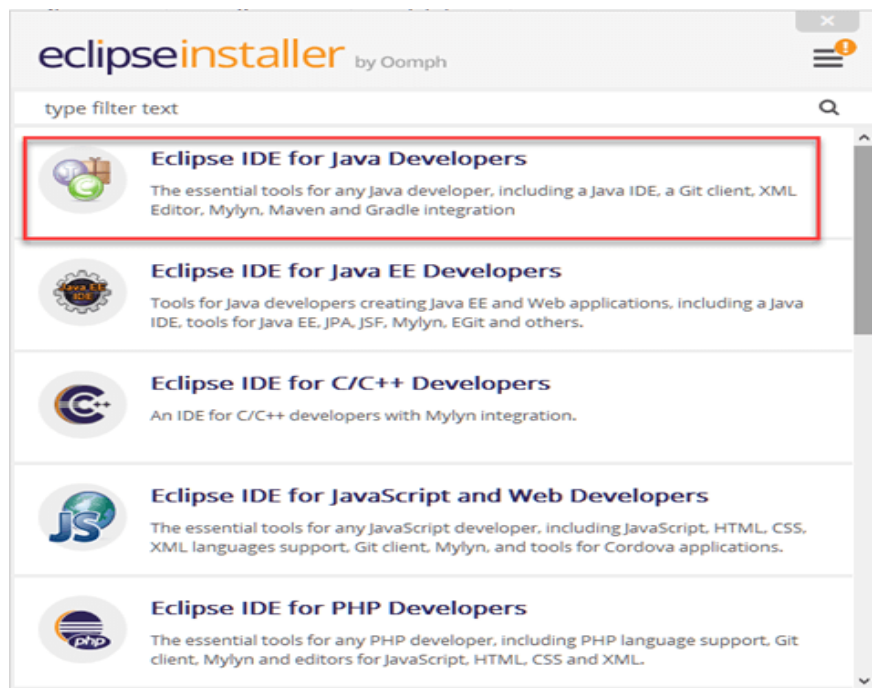
Download latest version of “Eclipse IDE for Java Developers” [here](#). Be sure to choose correctly between Windows 32 Bit and 64 Bit versions.



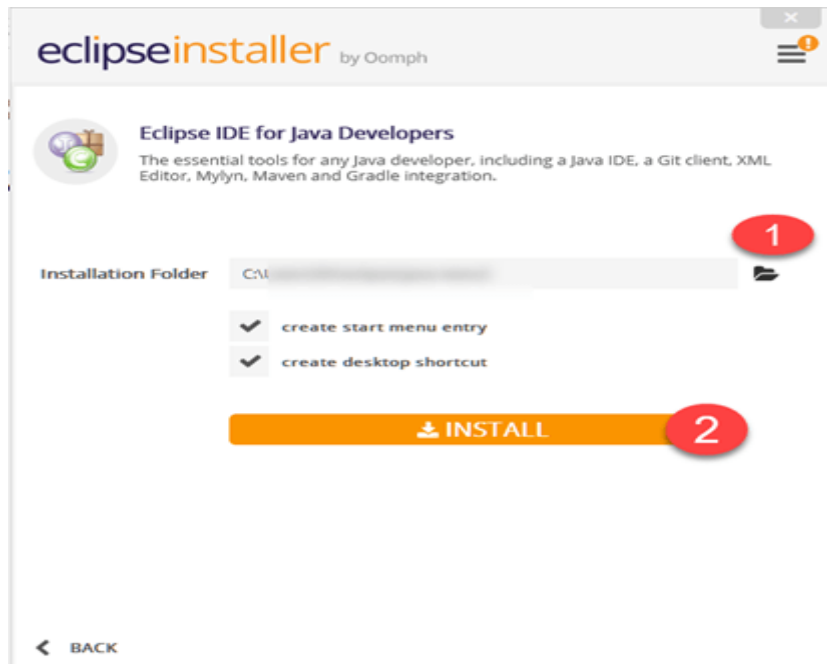
You should be able to download an exe file named “eclipse-inst-win64” for Setup.



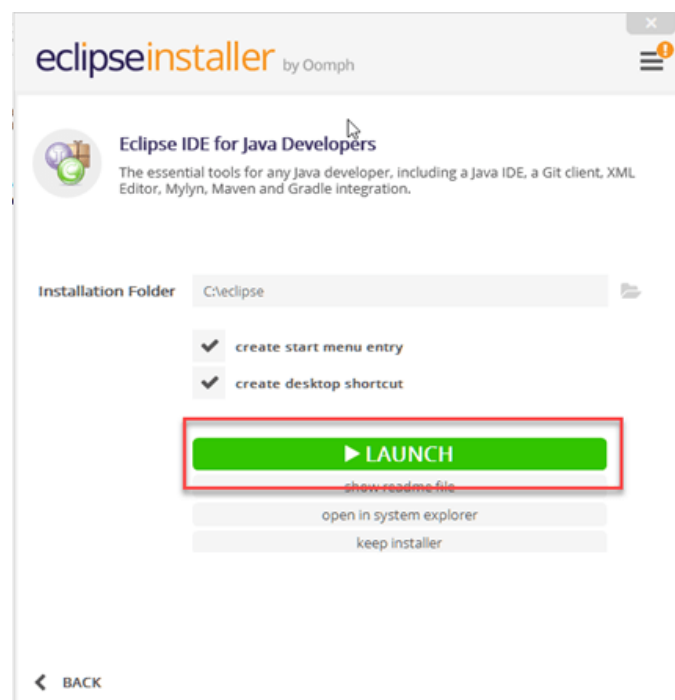
Double-click on file to Install the Eclipse. A new window will open. Click Eclipse IDE for Java Developers.



After that, a new window will open which click button marked 1 and change path to “C:\eclipse”. Post that Click on Install button marked 2



After successful completion of the installation procedure, a window will appear. On that window click on Launch



This will start eclipse neon IDE for you.

Step 3 – Download the Selenium Java Client Driver

You can download **Selenium Webdriver for Java Client Driver** [here](#). You will find client drivers for other languages there, but only choose the one for Java.

Selenium Client & WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers.

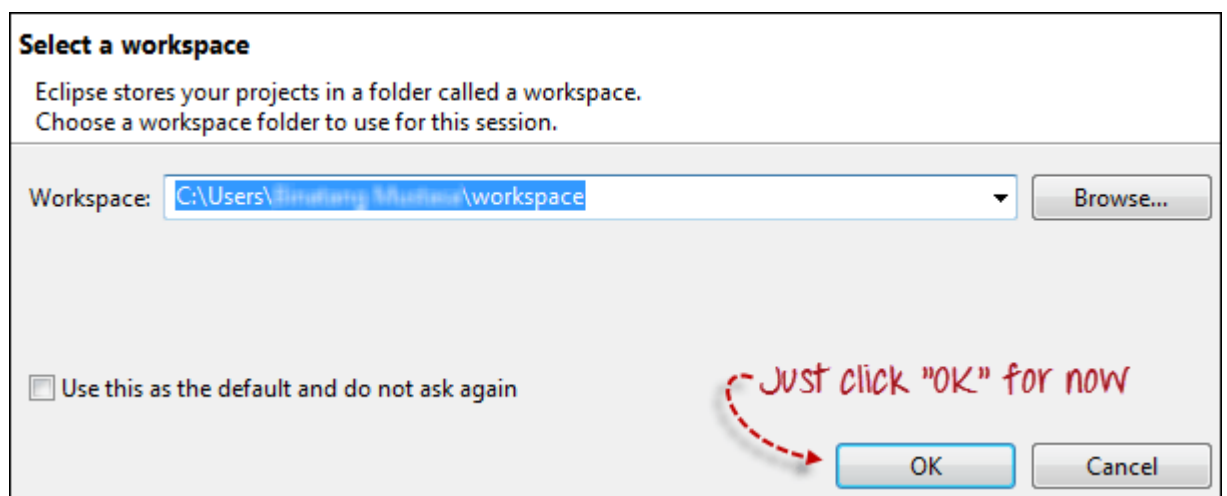
While language bindings for [other languages exist](#), these are the core ones that are supported by the main project hosted on GitHub.

LANGUAGE	VERSION	RELEASE DATE	
Ruby	3.142.6	October 04, 2019	Download
JavaScript	4.0.0-alpha.5	September 08, 2019	Download
Java	3.141.59	November 14, 2018	Download
Python	3.141.0	November 01, 2018	Download
C#	3.14.0	August 02, 2018	Download

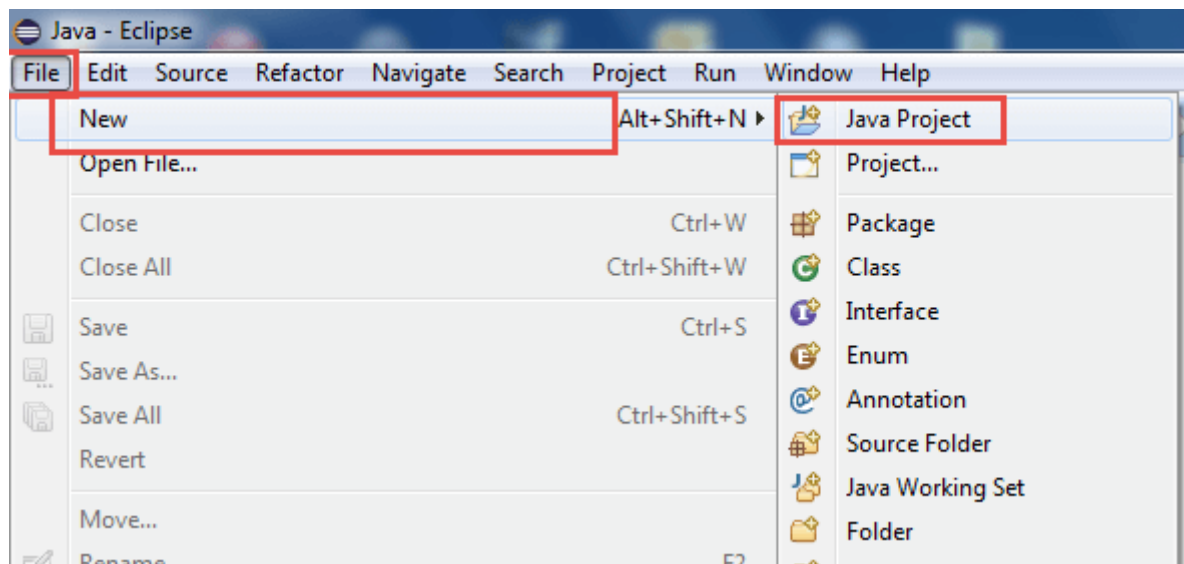
This download comes as a ZIP file named “selenium-3.14.0.zip”. For simplicity of Selenium installation on Windows 10, extract the contents of this ZIP file on your C drive so that you would have the directory “C:\selenium-3.14.0\”. This directory contains all the JAR files that we would later import on Eclipse for Selenium setup.

Step 4 – Configure Eclipse IDE with WebDriver

1. Launch the “eclipse.exe” file inside the “eclipse” folder that we extracted in step 2. If you followed step 2 correctly, the executable should be located on C:\eclipse\eclipse.exe.
2. When asked to select for a workspace, just accept the default location.

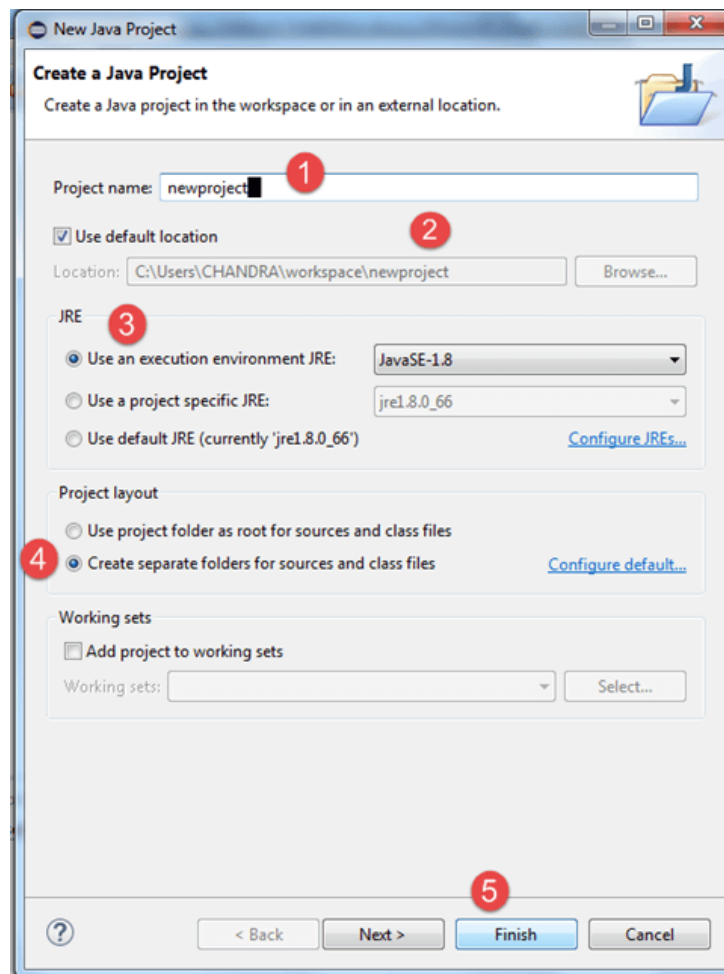


3. Create a new project through File > New > Java Project. Name the project as “newproject”.



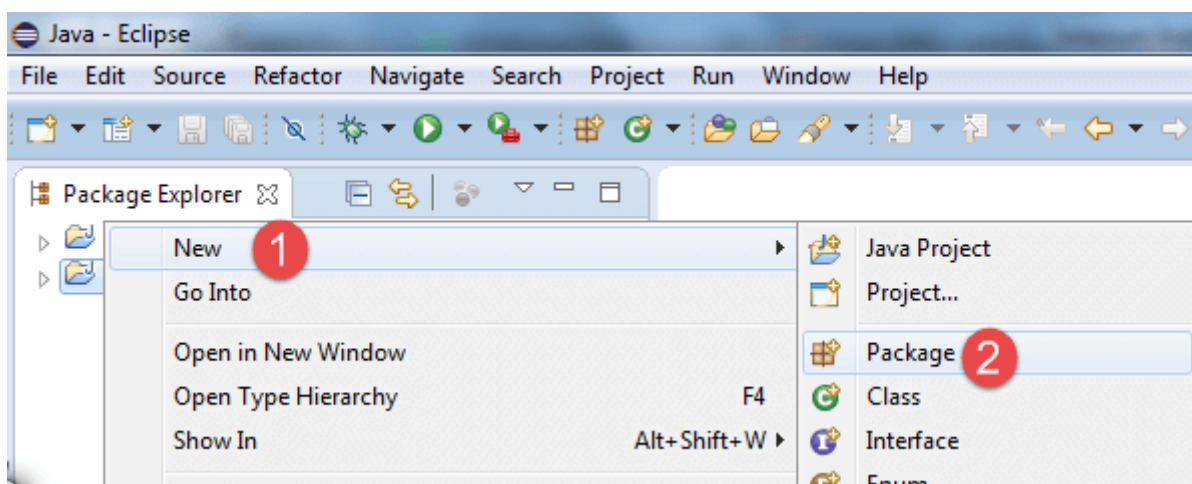
A new pop-up window will open enter details as follow

1. Project Name
2. Location to save project
3. Select an execution JRE
4. Select layout project option
5. Click on Finish button



4. In this step,

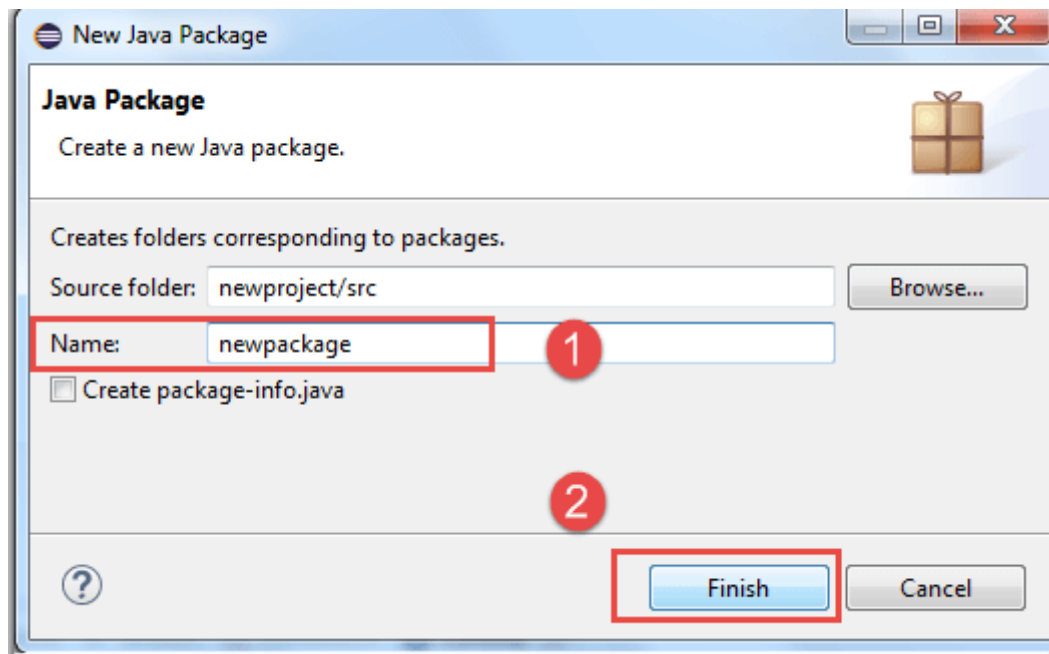
1. Right-click on the newly created project and
2. Select New > Package, and name that package as “newpackage”.



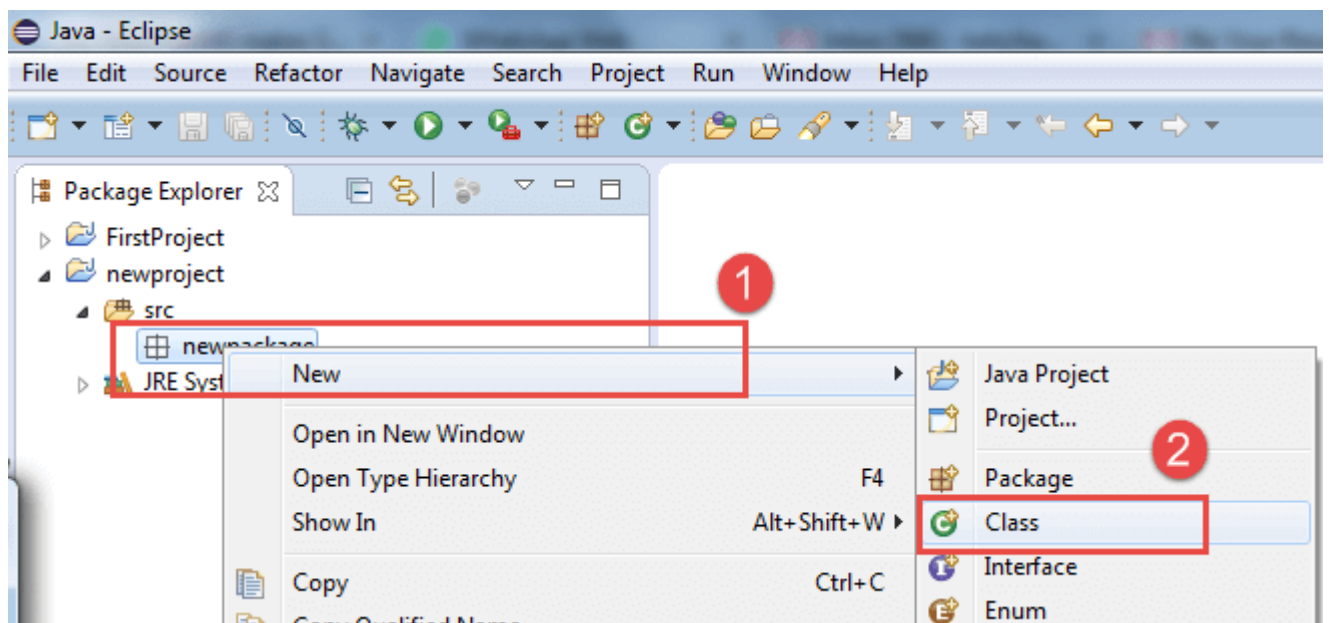
A pop-up window will open to name the package,

1. Enter the name of the package

2. Click on Finish button

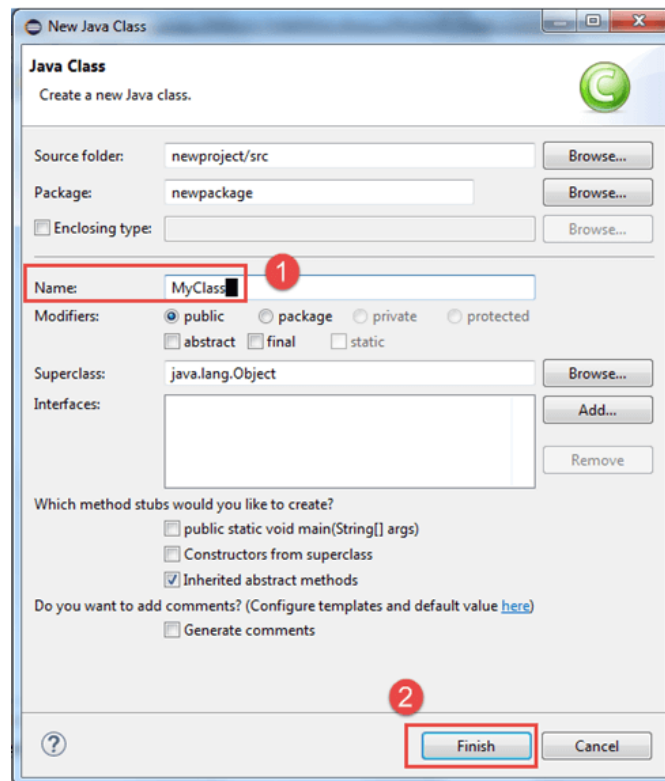


5. Create a new Java class under newpackage by right-clicking on it and then selecting- New > Class, and then name it as “MyClass”. Your Eclipse IDE should look like the image below.

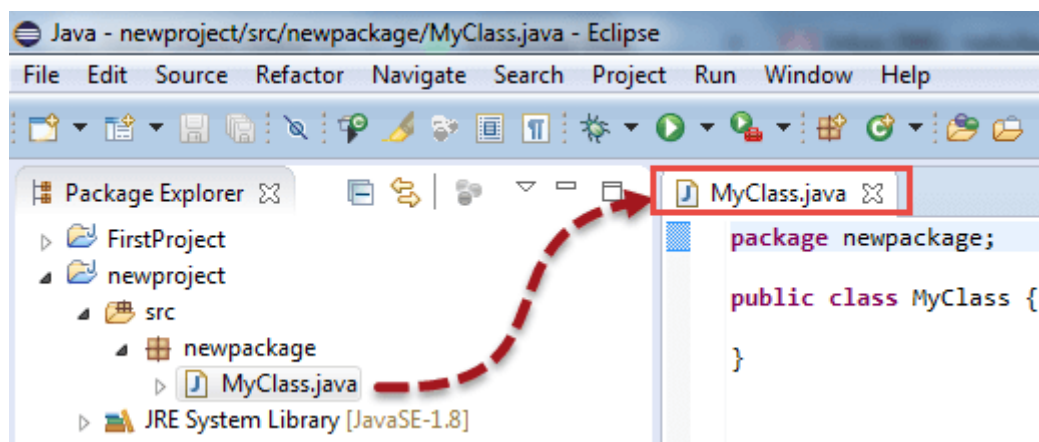


When you click on Class, a pop-up window will open, enter details as

1. Name of the class
2. Click on Finish button



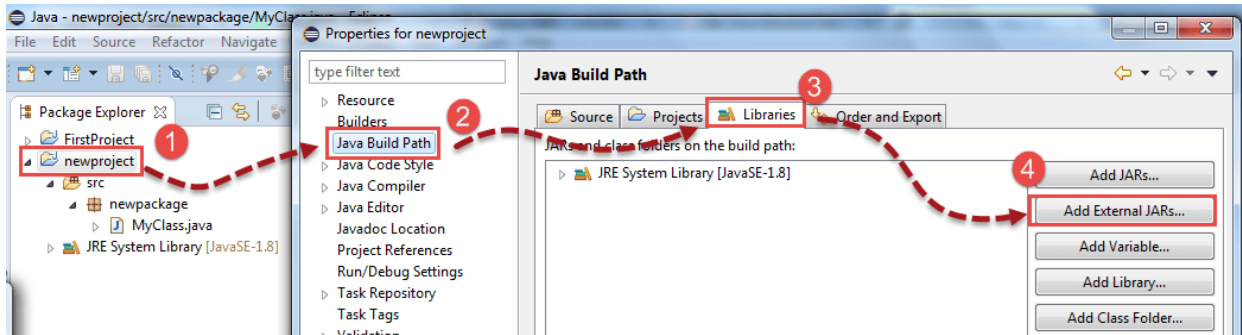
This is how it looks like after creating class.



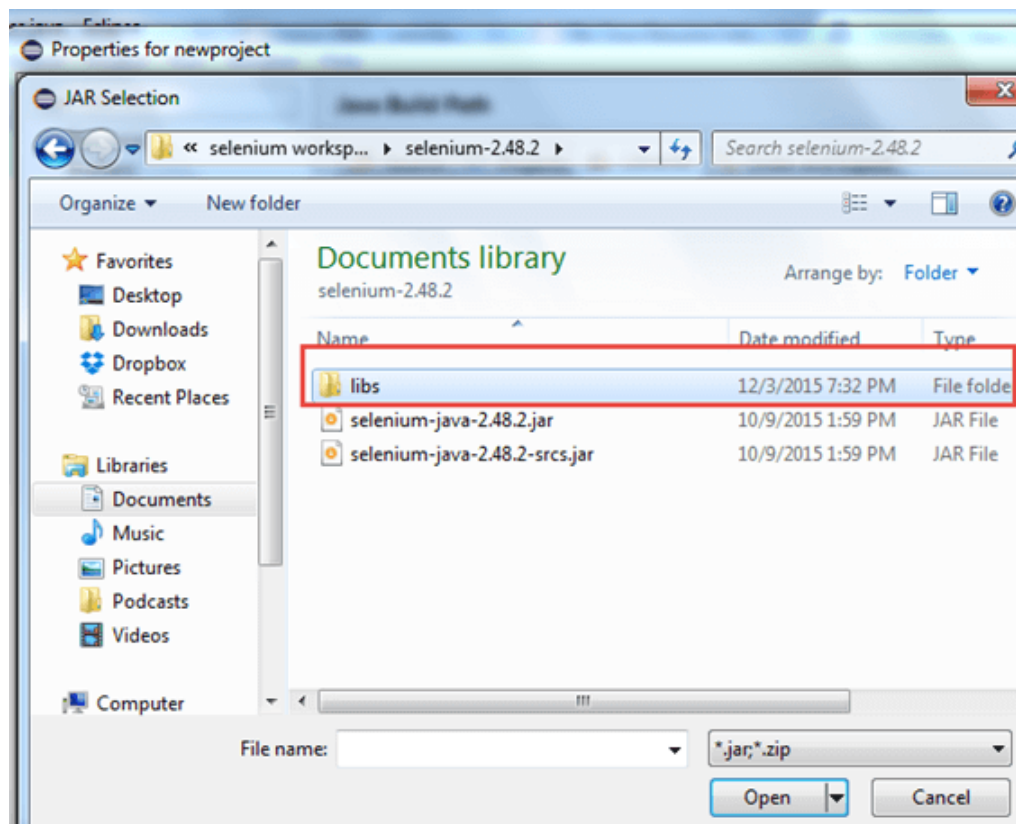
Now selenium WebDriver's into Java Build Path

In this step,

1. Right-click on "newproject" and select **Properties**.
2. On the Properties dialog, click on "Java Build Path".
3. Click on the **Libraries** tab, and then
4. Click on "Add External JARs.."

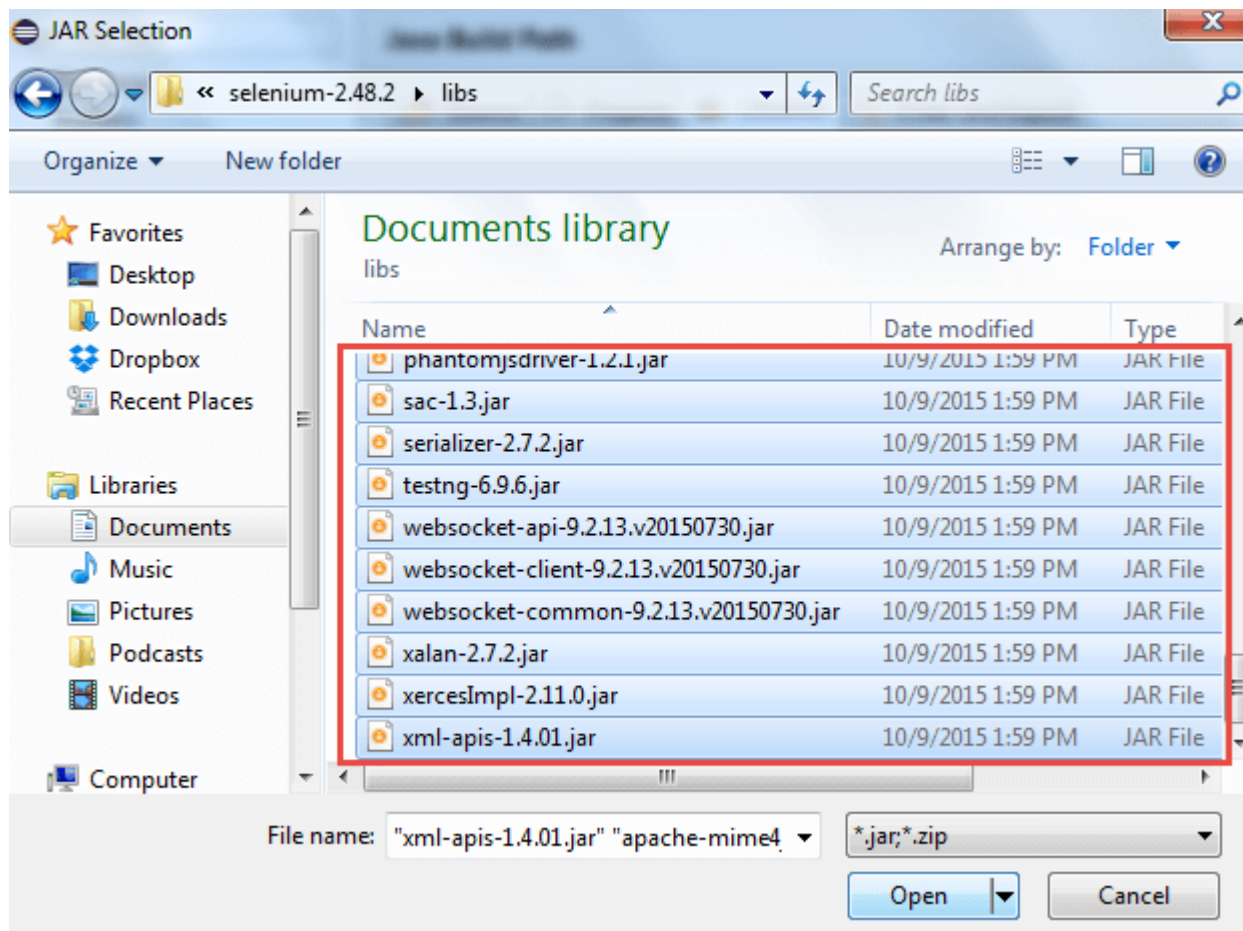


When you click on “Add External JARs..” It will open a pop-up window. Select the JAR files you want to add.

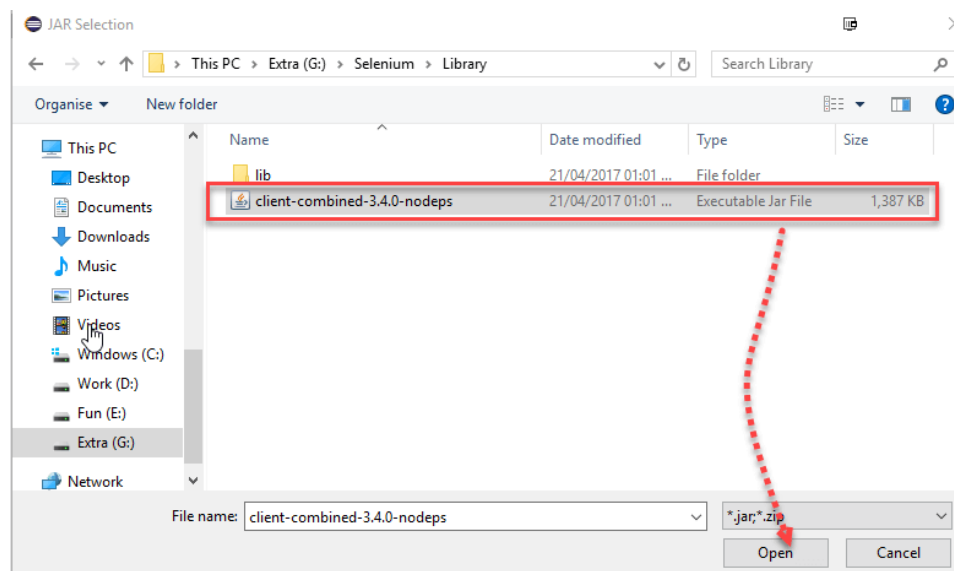


After selecting jar files, click on OK button.

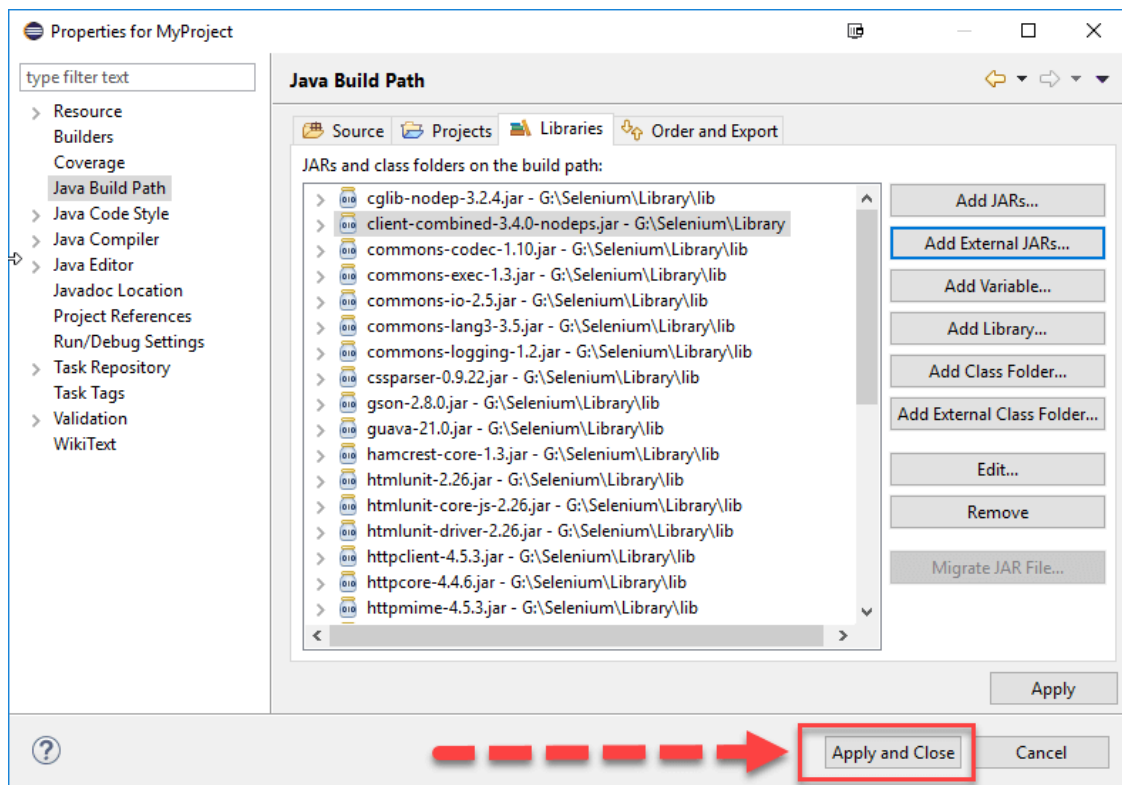
Select all files inside the lib folder.



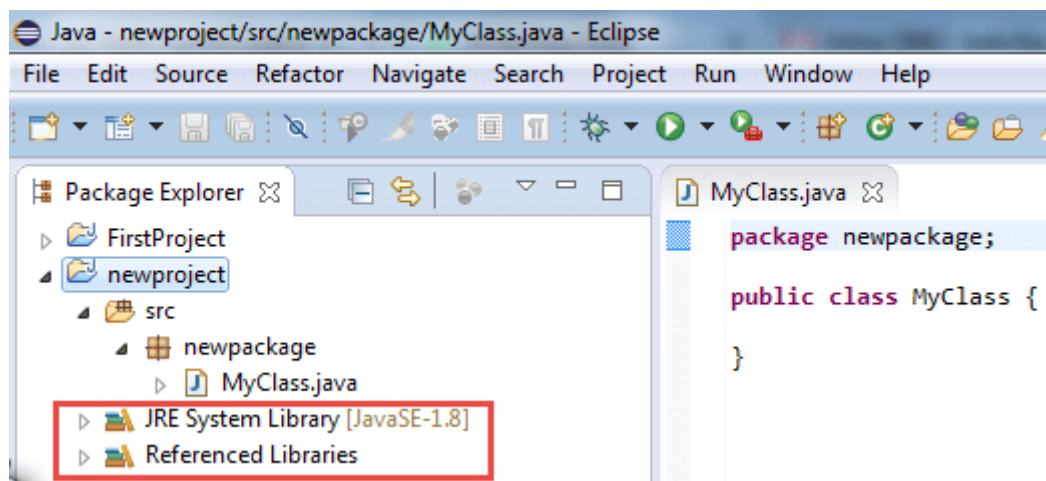
Select files outside lib folder



Once done, click “Apply and Close” button



6. Add all the JAR files inside and outside the “libs” folder. Your Properties dialog should now look similar to the image below.



7. Finally, click OK and we are done importing Selenium libraries into our project.

11. Write a simple program in JavaScript and perform testing using Selenium.

Problem Statement: Generate an alert window using JavascriptExecutor.

Objective: Create a login automation script using Selenium that generates an alert window using JavascriptExecutor methods.

Code:

JAVA Program

```
package newpackage;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.Test;

import org.openqa.selenium.JavascriptExecutor;

public class LoginAutomation {

    @Test

    public void login() {

        System.setProperty("webdriver.chrome.driver", "path");

        WebDriver driver = new ChromeDriver();

        JavascriptExecutor js = (JavascriptExecutor)driver;

        driver.manage().window().maximize();

        driver.get("https://www.browserstack.com/users/sign_in");

        js.executeScript("document.getElementById('user_email_login').value='rbc@xyz.com'");

        js.executeScript("document.getElementById('user_password').value='password'");

        js.executeScript("document.getElementById('user_submit').click()");
```

```
js.executeScript("alert('enter correct login credentials to continue');");
```

```
sleep(2000);
```

```
}
```

```
public static void sleep(int ms)
```

```
{
```

```
try
```

```
{
```

```
Thread.sleep(ms);
```

```
}
```

```
catch(InterruptedException e)
```

```
{
```

```
e.printStackTrace();
```

```
}
```

```
}
```

```
}
```

```
[/java]
```

<https://www.lambdatest.com/blog/automation-testing-with-selenium-javascript/>

12. Develop test cases for the above containerized application using selenium.

Running Selenium Tests in Docker

Once the Docker Desktop is installed, it can run a few docker images. You can either create a Docker image from scratch or pull an already configured base image from the Docker hub and then add on to it.

Docker hub is the central registry hosted by Docker, where developers and organizations build and host numerous containers for the community.

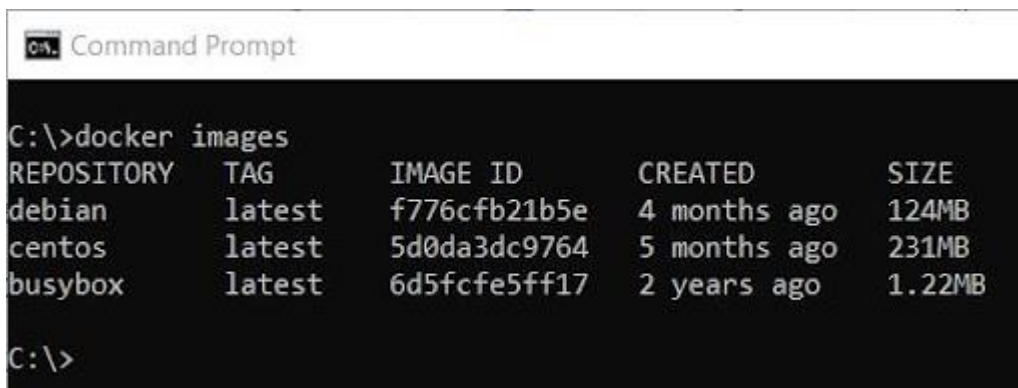
Did you know the difference between Docker and BrowserStack Cloud Infrastructure to meet your Selenium testing needs? [Find out.](#)

This tutorial uses the [selenium/standalone-chrome](#) image hosted by selenium on DockerHub.

Step 1: Pull the docker image

To get a list of all the already existing images on the system, run the following command in the command prompt:

```
docker images
```



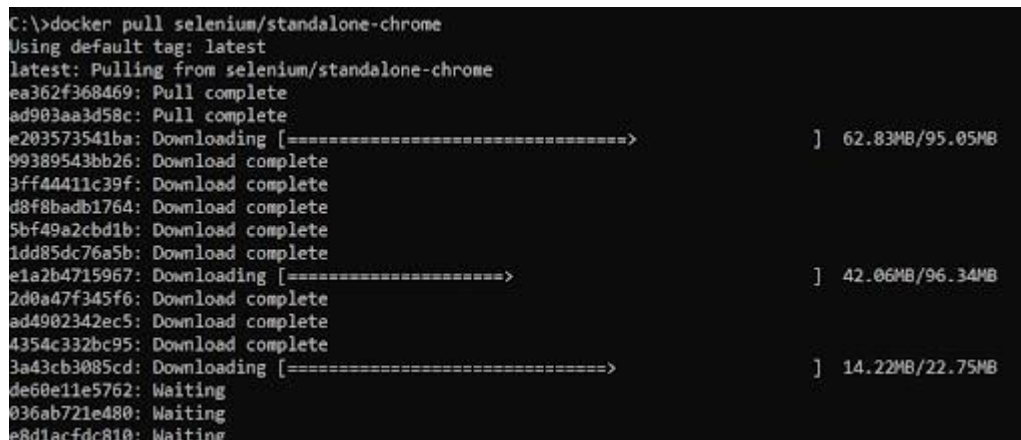
```
Command Prompt

C:\>docker images
REPOSITORY      TAG              IMAGE ID         CREATED          SIZE
debian          latest          f776cfb21b5e    4 months ago    124MB
centos          latest          5d0da3dc9764    5 months ago    231MB
busybox         latest          6d5fcfe5ff17    2 years ago     1.22MB

C:\>
```

If you do not already have the selenium standalone-chrome docker image, run the following command to download a copy of the image onto the system.

```
docker pull selenium/standalone-chrome
```



```
C:\>docker pull selenium/standalone-chrome
Using default tag: latest
latest: Pulling from selenium/standalone-chrome
ea362f368469: Pull complete
ad903aa3d58c: Pull complete
e203573541ba: Downloading [=====>] 62.83MB/95.05MB
99389543bb26: Download complete
3ff44411c39f: Download complete
d8f8badb1764: Download complete
5bf49a2cbd1b: Download complete
1dd85dc76a5b: Download complete
e1a2b4715967: Downloading [=====>] 42.06MB/96.34MB
2d0a47f345f6: Download complete
ad4902342ec5: Download complete
4354c332bc95: Download complete
3a43cb3085cd: Downloading [=====>] 14.22MB/22.75MB
de60e11e5762: Waiting
036ab721e480: Waiting
e8d1acfdc810: Waiting
```

Upon rerunning the command docker images, selenium/standalone-chrome image appears in the list.

```
C:\>docker images
REPOSITORY          TAG          IMAGE ID      CREATED      SIZE
selenium/standalone-chrome  latest      d0b6caec4485  9 days ago  1.2GB
debian              latest      f776cfb21b5e  4 months ago 124MB
centos              latest      5d0da3dc9764  5 months ago 231MB
busybox            latest      6d5fcf85ff17  2 years ago  1.22MB
```

Step 2: Running the Selenium Webdriver Docker container

Upon pulling the selenium/standalone-chrome image onto the system, start the container by running the following command:

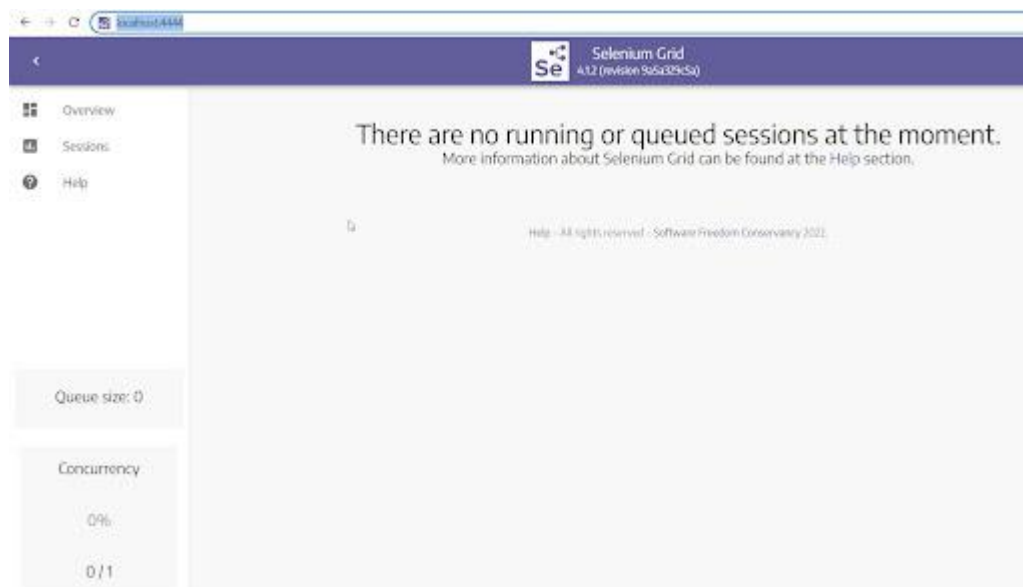
```
docker run -d -p 4444:4444 -v /dev/shm:/dev/shm selenium/standalone-chrome
```

The above command starts a container from the image specified in detached mode (background mode). It also maps Port 4444 on the container to Port 4444 on your local browser.

```
C:\Users\bshrikanth>docker run -d -p 4444:4444 selenium/standalone-chrome
837f375519d620fb2f53f70ded60eaabf87756e5819757a736df1aca62fe1bf4
```

The command, when run, will return the ContainerID.

Open the browser and navigate to <http://localhost:4444/>. It reflects Selenium Grid UI, as shown below.



Step 3: Creating a sample test file

[Selenium](#) supports tests written in different languages of which Java and Python are most popularly used. In this example, using Python to create Selenium Test.

```
from selenium import webdriver
```

To open file in chrome browser, [Chromedriver](#) is necessary. Therefore, initializing ChromeOptions to declare any connection and driver options necessary while instantiating a browser instance.

```
options = webdriver.ChromeOptions()
```

```
options.add_argument('--ignore-ssl-errors=yes')

options.add_argument('--ignore-certificate-errors')
```

Creating an instance of the Remote webdriver and passing the selenium endpoint and chrome options defined in the previous step.

```
driver = webdriver.Remote(

command_executor='http://localhost:4444/wd/hub',

options=options

)
```

Navigate to the [BrowserStack website](https://www.browserstack.com/) and click on the Get Started for Free button. Inducing wait time between the two actions allows viewing the execution of tests.

```
driver.get("https://www.browserstack.com/")

driver.find_element_by_link_text("Get started free").click()

seleniumDockerTest.py

from selenium import webdriver

import time


print("Test Execution Started")

options = webdriver.ChromeOptions()

options.add_argument('--ignore-ssl-errors=yes')

options.add_argument('--ignore-certificate-errors')

driver = webdriver.Remote(

command_executor='http://localhost:4444/wd/hub',

options=options

)
```

```
#maximize the window size
```

```
driver.maximize_window()
```

```
time.sleep(10)
```

```
#navigate to browserstack.com
```

```
driver.get("https://www.browserstack.com/")
```

```
time.sleep(10)
```

```
#click on the Get started for free button
```

```
driver.find_element_by_link_text("Get started free").click()
```

```
time.sleep(10)
```

```
#close the browser
```

```
driver.close()
```

```
driver.quit()
```

```
print("Test Execution Successfully Completed!")
```

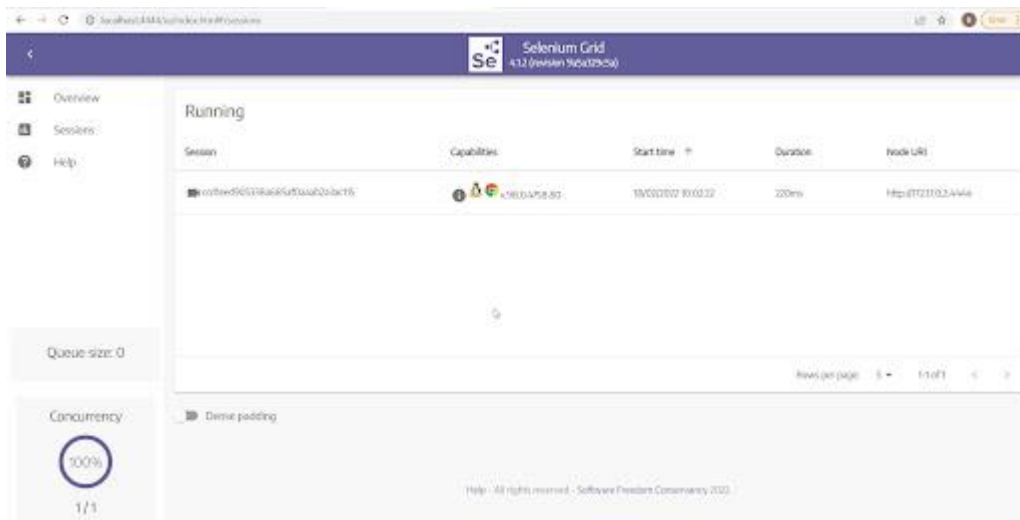
Step 4: Executing the test case

Python test case file can be run using either an IDE or command prompt. To run it from the command prompt, open a command prompt and run the following command:

```
python <filename>
```

Navigate to the sessions tab on the Selenium Grid UI upon running the command. It would reflect an active session. Click on the video icon to check how automation works.

Note: If prompted for a password while opening the video, “secret” should be the default password.



The above steps allow running Selenium tests in Docker seamlessly.