**WEEK-7 Develop a simple containerized application using Docker**

Open Command Prompt and Change the Directory to Desktop

C:\Users\VAAGDEVI>**cd Desktop**

C:\Users\VAAGDEVI/Desktop>**git clone** [**https://github.com/your-username/Reg.git**](https://github.com/your-username/Reg.git)

Open Reg Folder from Desktop

Right Click in Reg Folder (Whitespace) -----> New---->New Text Document

**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.

Create a file named Dockerfile in the same folder as the file index.html with the

following contents.

----------------------------------------------------------------------------------------------------

FROM nginx:alpine

COPY index.html /usr/share/nginx/html/index.html

EXPOSE 80

**How to Save Dockerfile:**

File----->Save As ---->File Name: “Dockerfile”

Save As Type ----> All Files

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.

**FROM nginx: alpine**

Sets your base image to Nginx on Alpine Linux. Nginx is already Pre-configured to serve static files from /usr/share/ngnix/html, you only need to copy your index.html there.

You get a lightweight image that already has Nginx installed and default config that serves fles from /usr/share/nginx/html.

**COPY index.html /usr/share/nginx/html/index.html**

During the build, Docker takes the index.html file from your build context (the folder you run docker build in) and puts it inside the image at Nginx's default web root. This overwrites the default Nginx welcome page so your page is served at /.

**EXPOSE 80**

Adds metadata that the container listens on port 80.

**Then Change the Directory to Reg Folder**

C:\Users\VAAGDEVI\Desktop>**cd Reg**

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 nginx: alpine image. But, since we didn’t have that on our machine, that image needed to be downloaded.

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 registration-form, 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.

C:\Users\VAAGDEVI\Desktop\Reg>**docker build -t registration-form .**

[+] Building 1.9s (7/7) FINISHED docker:desktop-linux

=> [internal] load build definition from Dockerfile 0.0s

=> => transferring dockerfile: 115B 0.0s

=> [internal] load metadata for docker.io/library/nginx:alpine 1.6s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> [internal] load build context 0.0s

=> => transferring context: 32B 0.0s

=> [1/2] FROM docker.io/library/nginx:alpine@sha256:42a516af16b852e33b7682d5ef8acbd5d13fe08fecadc7ed98605ba5e3b2 0.0s

=> => resolve docker.io/library/nginx:alpine@sha256:42a516af16b852e33b7682d5ef8acbd5d13fe08fecadc7ed98605ba5e3b2 0.0s

=> CACHED [2/2] COPY index.html /usr/share/nginx/html/index.html 0.0s

=> exporting to image 0.1s

=> => exporting layers 0.0s

=> => exporting manifest sha256:e43000260ca92cdf371645589509bb7988c5024f0de6e568e8acdbfc43c2b194 0.0s

=> => exporting config sha256:5e0dab59c77cd598ff5520385934013d16ed2fb74ea83d4c3f773d9573c5dfd3 0.0s

=> => exporting attestation manifest sha256:1fac3703e798f97e649005ce324d33ebdcd8f7e54f655e7cf790ae667aa3ec84 0.0s

=> => exporting manifest list sha256:926c5364ffe954c5b3fd6686c7a6ad7f767673fa7decae3a079dee9a47c940c4 0.0s

=> => naming to docker.io/library/registration-form:latest 0.0s

=> => unpacking to docker.io/library/registration-form:latest 0.0s

**docker build:** Create a new image using the Dockerfile in the current Directory.

**-t registration-form:** Tags the image with the name registration-form.

**. Dot:** the build Context- i.e, “Use the files in the folder"

C:\Users\VAAGDEVI\Desktop\Reg>**docker images**

REPOSITORY TAG IMAGE ID CREATED SIZE

registration-form latest fd217567bb59 36 seconds ago 79.4MB

**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:

C:\Users\VAAGDEVI\Desktop\Reg>**docker run -dp 8080:80 registration-form**

**1ad2c8096263f92f2ed11f5c8dc40b5e28395df649810e20050217de7c4c2cf1**

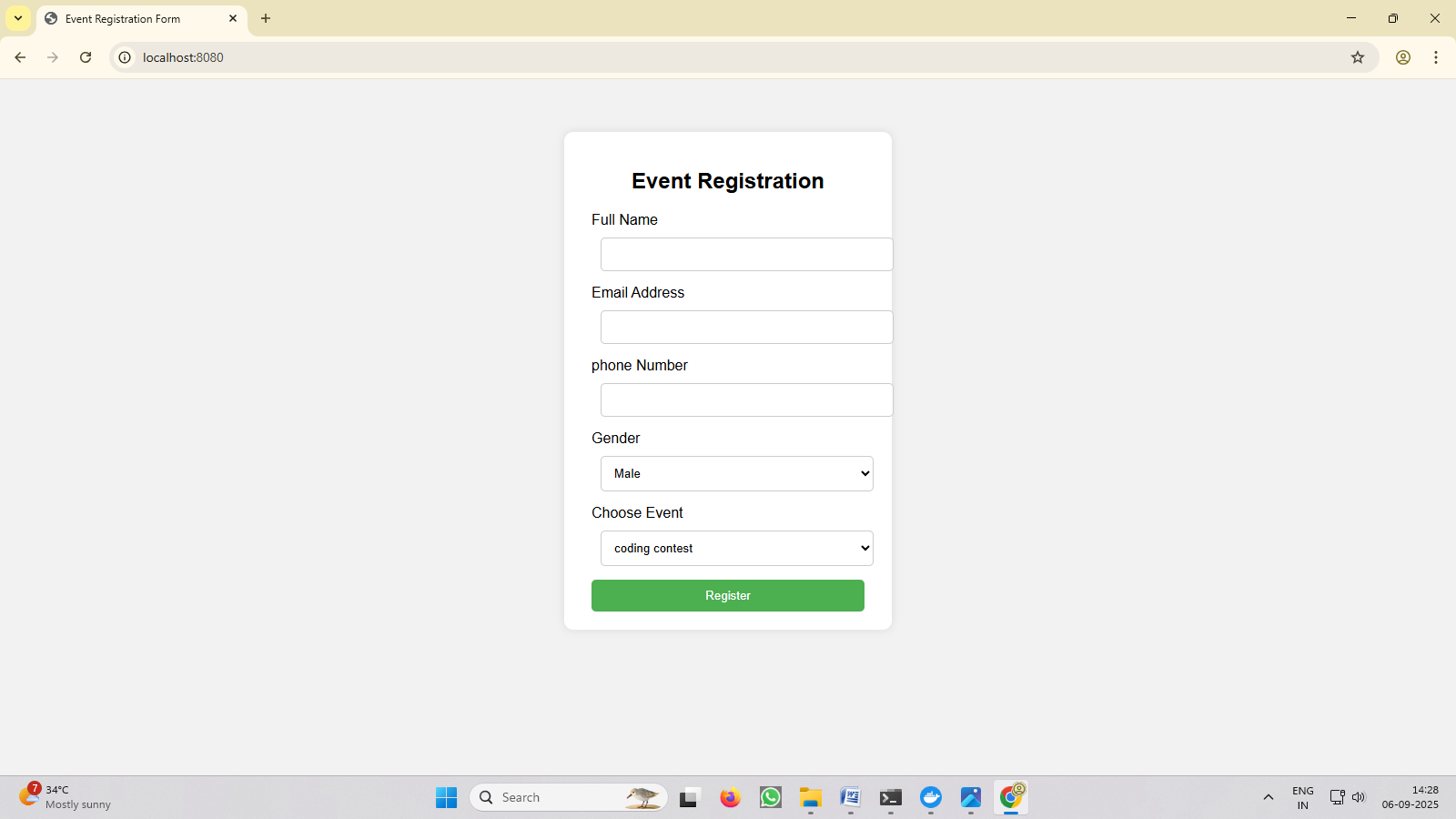
Remember the -d and -p flags? We’re running the new container in “detached” mode (in the background) and creating a mapping between the host’s port 8080 to the container’s port 80. Without the port mapping, we wouldn’t be able to access the application**.**

**docker run:** starts a container (running instance) from an image.

**-d:** Detached mode (Runs in the background; prints the container ID.

**-P 8080:80** --> Port Mapping--> host 8080--> Container 80(Nginx listens on 80inside)

After a few seconds, open your web browser to **http://localhost:8080**. You should see our app.



**WEEK-8 Integrate Kubernetes with Docker**

**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)

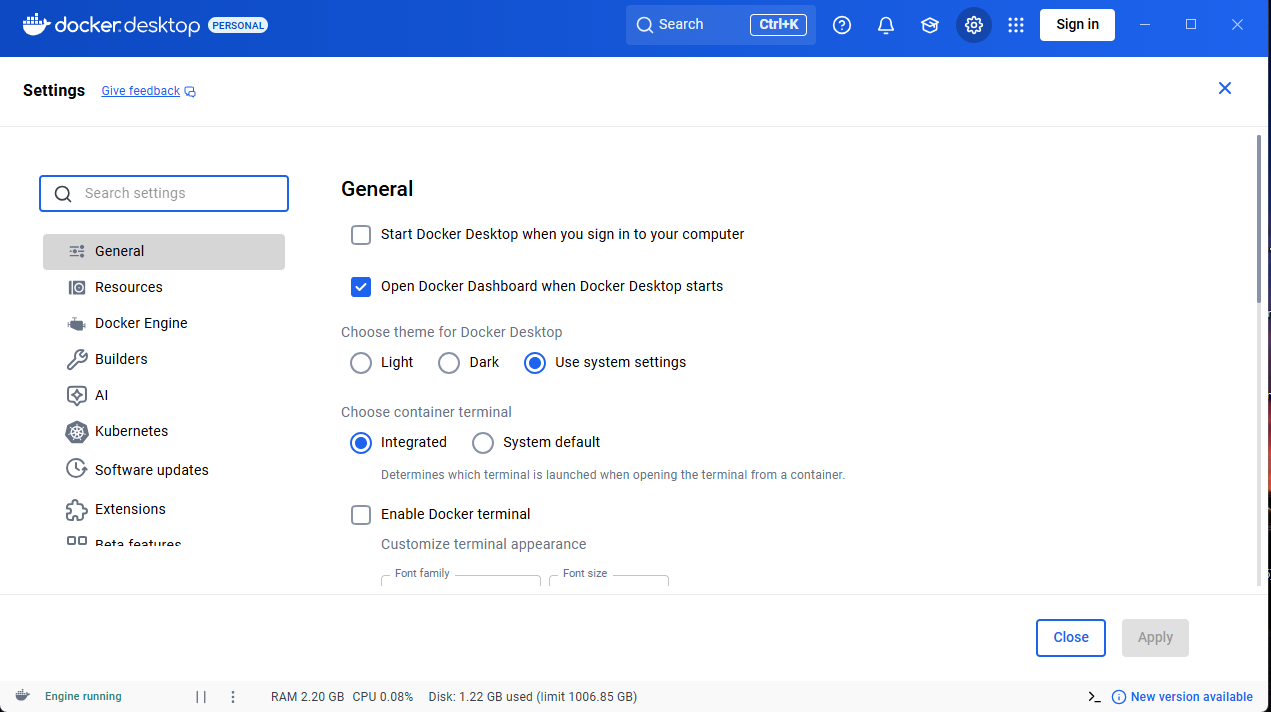
Older operating systems can’t use Docker Desktop :( You can use Docker Toolbox instead, but that doesn’t come with Kubernetes - so you’ll need to run Kubernetes in Docker.

**2. Enable Kubernetes**

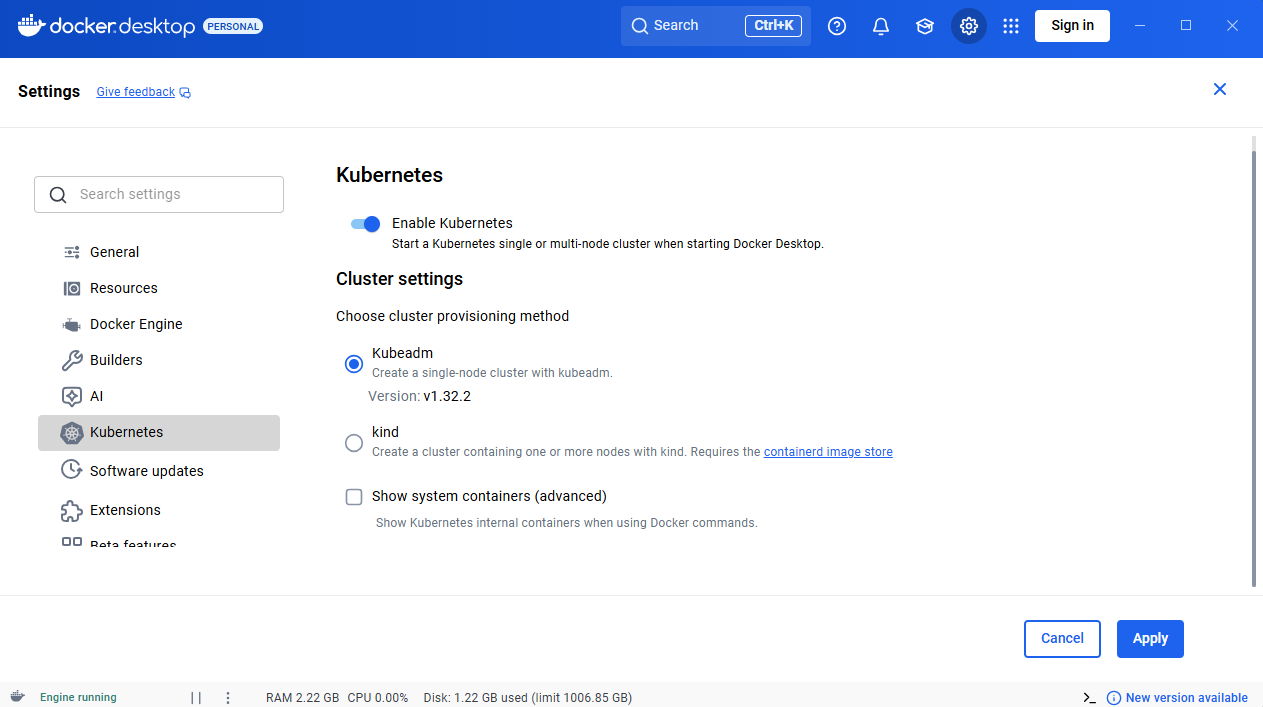
Kubernetes itself runs in containers. When you deploy a Kubenetes cluster you first install Docker (or another container runtime like containerd) and then use tools like kubeadm which starts all the Kubernetes components in containers. Docker Desktop does all that for you.

Make sure you have Docker Desktop running - in the taskbar in Windows and search Docker Desktop, Open Docker-Desktop and Click on Settings:

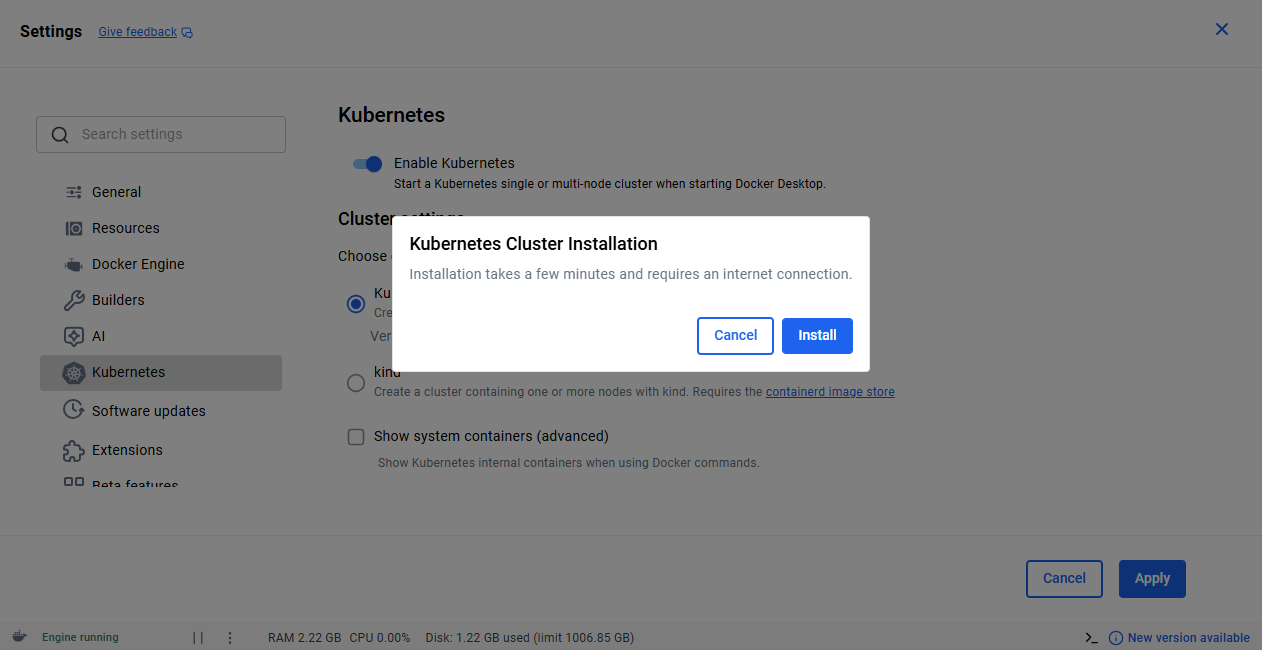
A new screen opens with all of Docker Desktop’s configuration options.

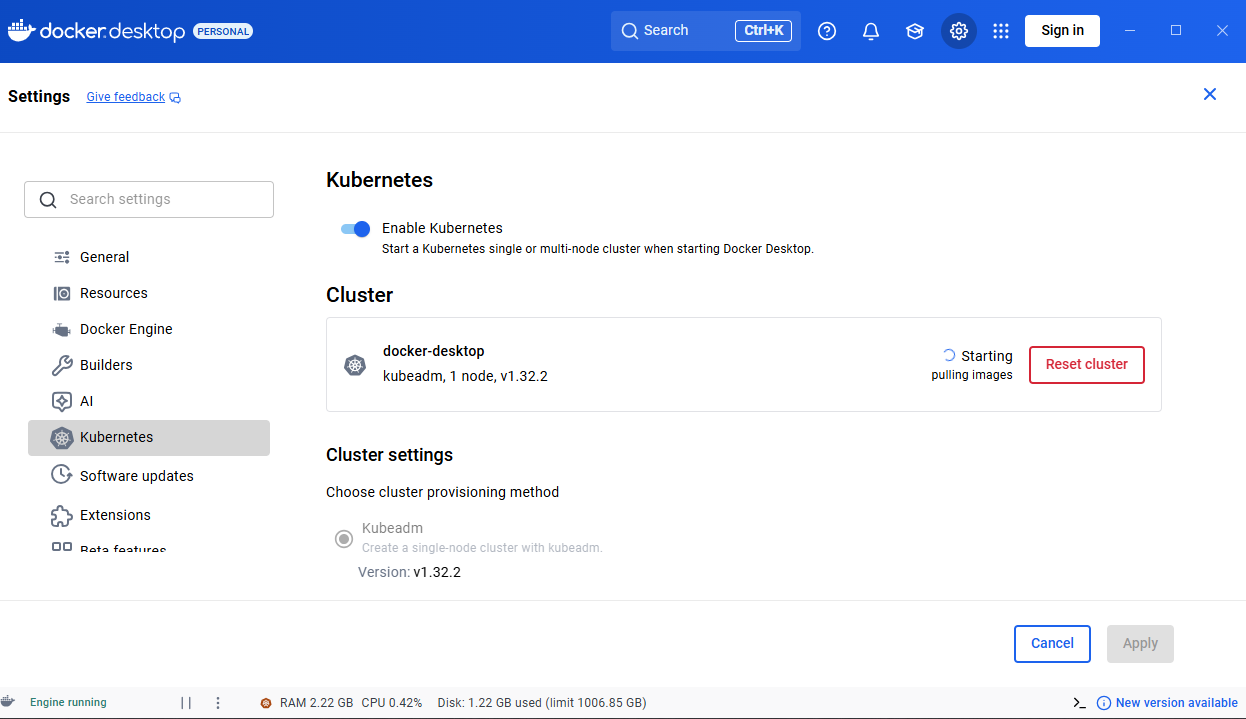


Click on Kubernetes and Enable Kubernetes Toggle Button.



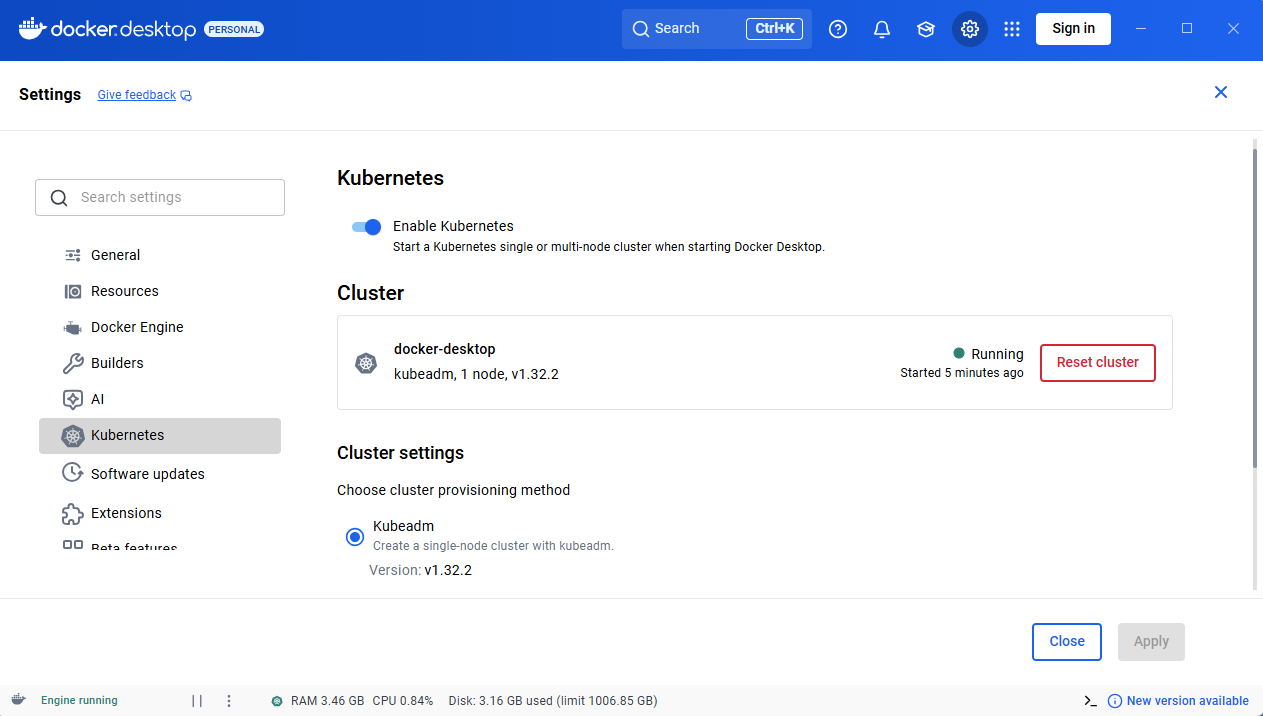
Click Apply & Click Install





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:

C:\Users\VAAGDEVI>**kubectl version --client**

Client Version: v1.32.2

Kustomize Version: v5.5.0

C:\Users\VAAGDEVI>**kubectl get nodes**

NAME STATUS ROLES AGE VERSION

docker-desktop Ready control-plane 67m v1.32.2

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.

**WEEK-9 Automate the process of running containerized application exercise 7 using kubernetes.**

**Step-1 Create a Kubernetes Deployment YAML**

Open Reg Folder from Desktop

Right Click in Reg Folder (Whitespace) -----> New---->New Text Document

Let’s create a file named **registration-form-deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: registration-deployment

labels:

app: registration

spec:

replicas: 2

selector:

matchLabels:

app: registration

template:

metadata:

labels:

app: registration

spec:

containers:

- name: registration-container

image: registration-form:latest

imagePullPolicy: IfNotPresent

ports:

- containerPort: 80

How to save **registration-form-deployment.yaml**

File----->Save As ---->File Name: **registration-form-deployment.yaml**

Save As Type ----> All Files

**Step-2 Create a Kubernetes Service YAML**

Let’s create a file named **registration-form-service.yaml**

apiVersion: v1

kind: Service

metadata:

name: registration-service

labels:

app: registration

spec:

type: NodePort

selector:

app: registration

ports:

- port: 80

targetPort: 80

How to save **registration-form-service.yaml**

File----->Save As ---->File Name: **registration-form-service.yaml**

Save As Type ----> All Files

**Step 3: Apply YAMLs using kubectl**

Open terminal and run:

C:\Users\VAAGDEVI\Desktop\registration-form3> **kubectl apply -f registration-form-deployment.yaml**

deployment.apps/registration-form-deployment created

C:\Users\VAAGDEVI\Desktop\registration-form1>**kubectl apply -f registration-form-service.yaml**

service/registration-form-service created

C:\Users\VAAGDEVI\Desktop\registration-form1>**kubectl get pods**

NAME READY STATUS RESTARTS AGE

registration-form-deployment-dcc489987-ztd2s 1/1 Running 0 63s

C:\Users\VAAGDEVI\Desktop\registration-form1>**kubectl get services**

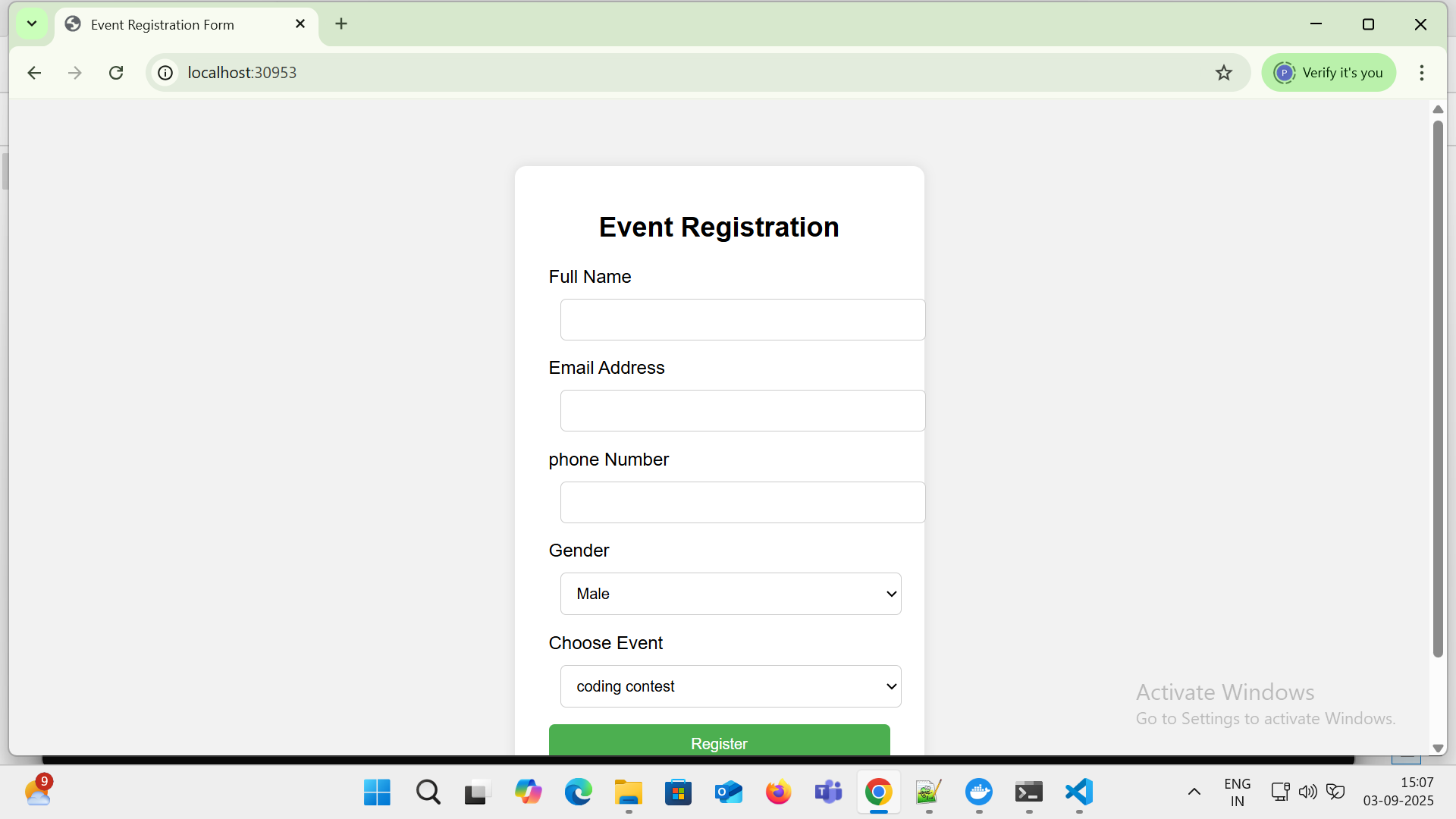
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 8h

registration-form-service NodePort 10.107.116.157 <none> 80:30080/TCP 41s

**Step 4: Access the App**

* Open browser and visit: [**http://localhost:30080**](http://localhost:30080)



## Step: 5 Automate with a Shell Script (Linux/Mac) or PowerShell (Windows)

Open Reg Folder from Desktop

Right Click in Reg Folder (Whitespace) -----> New---->New Text Document

Let’s create a file named **deploy.bat**

@echo off

echo Building Docker image...

docker build -t registration-form .

echo Deploying to Kubernetes...

kubectl apply -f registration-form-deployment.yaml

kubectl apply -f registration-form-service.yaml

echo Waiting for Pods to be ready...

kubectl get pods

echo Getting Services...

kubectl get services

How to save **deploy.bat**

File----->Save As ---->File Name: **deploy.bat**

Save As Type ----> All Files

Save this as deploy.bat in the same folder as your Dockerfile and YAML files.

## To Run Everything:

1. Ensure Kubernetes is enabled in Docker Desktop.
2. Put the following files in the same folder:
   * Dockerfile
   * index.html
   * registration-form-deployment.yaml
   * registration-form-service.yaml
   * deploy.bat
3. Open Command Prompt and run:

C:\Users\VAAGDEVI\Desktop\registration-form1>**deploy.bat**

**WEEK-10: Install and Explore Selenium for automated testing**

**AIM:** Install and Explore Selenium for automated testing

**DESCRIPTION:**

To install and explore Selenium for automated testing, you can follow these steps:

**Install Java Development Kit (JDK):**

• Selenium is written in Java, so you’ll need to install JDK in order to run it. You can download and install JDK from the official Oracle website.

• **Install the Selenium WebDriver:**

• You can download the latest version of the Selenium WebDriver from the Selenium website. You’ll also need to download the appropriate driver for your web browser of choice (e.g. Chrome Driver for Google Chrome).

[**https://www.selenium.dev/downloads/**](https://www.selenium.dev/downloads/)

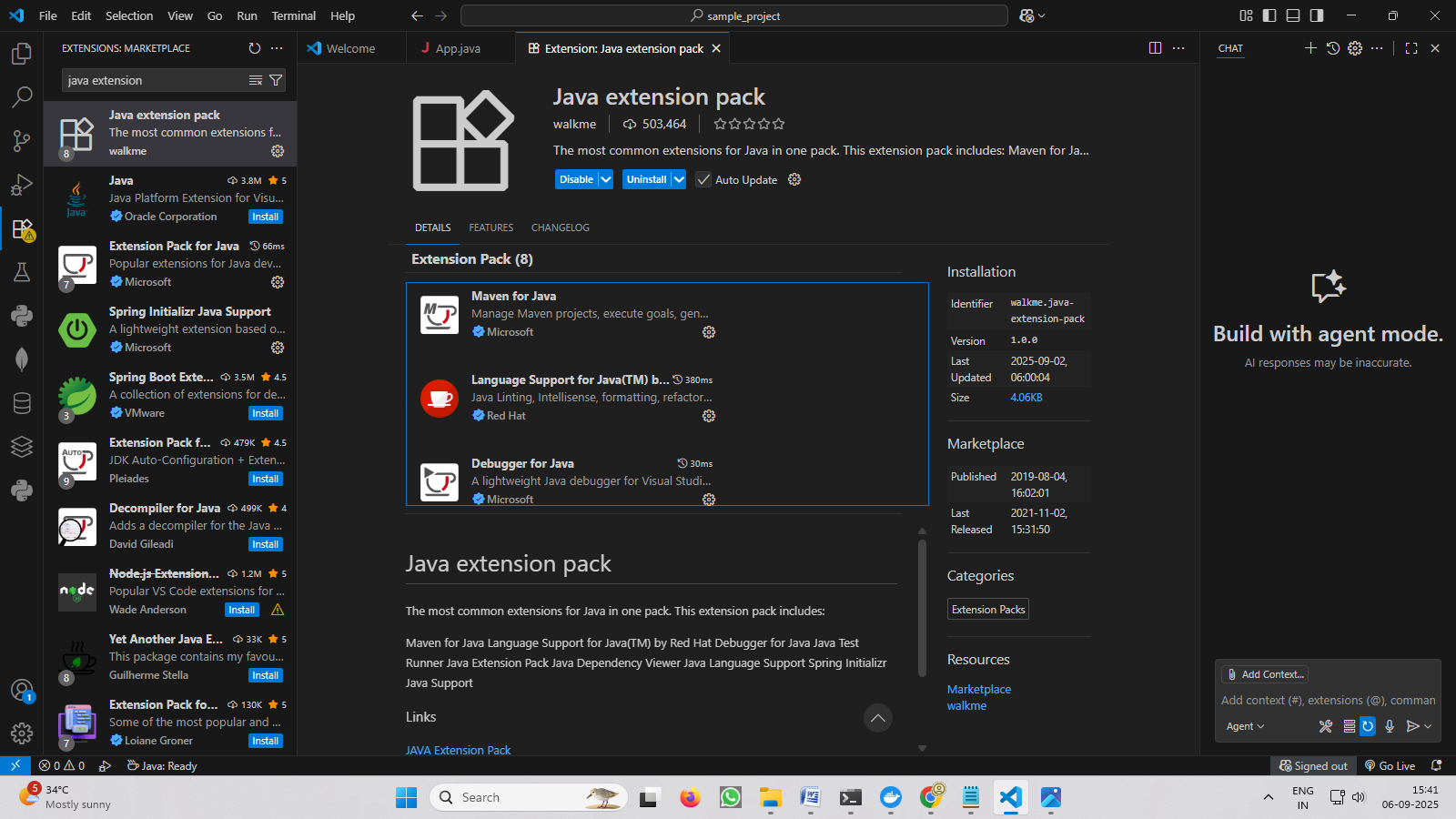
**.Download the chrome Driver:**

https://storage.googleapis.com/chrome-for-testing-public/139.0.7258.154/win64/chromedriver-win64.zip

**Install an Integrated Development Environment (IDE):**

• To write and run Selenium tests, you’ll need an IDE. Some popular choices include Eclipse, IntelliJ IDEA, and Visual Studio Code.

Open Visual Studio Code and Go to Extensions search **Java extension pack** and install it.



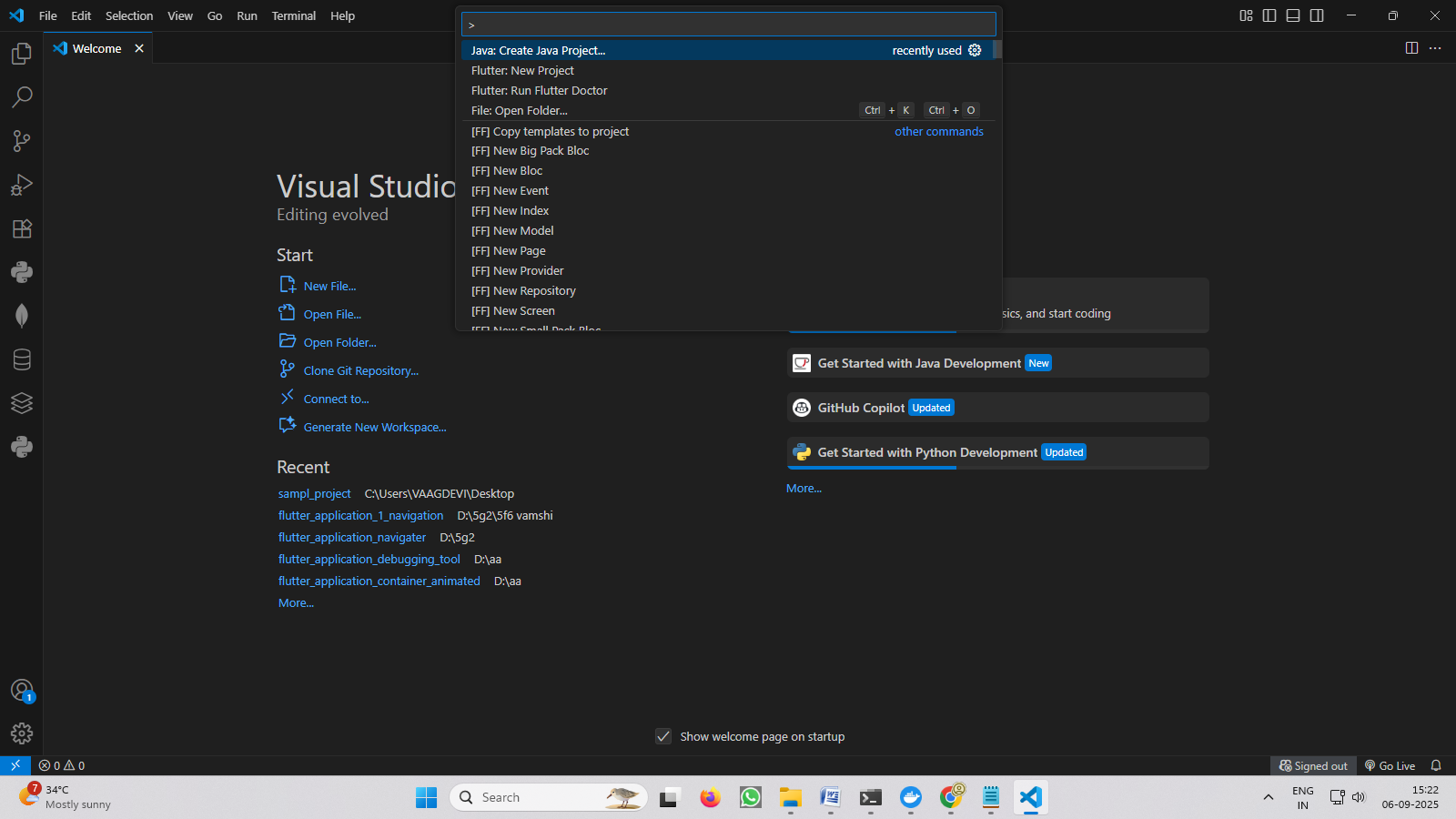
• Write a simple test:

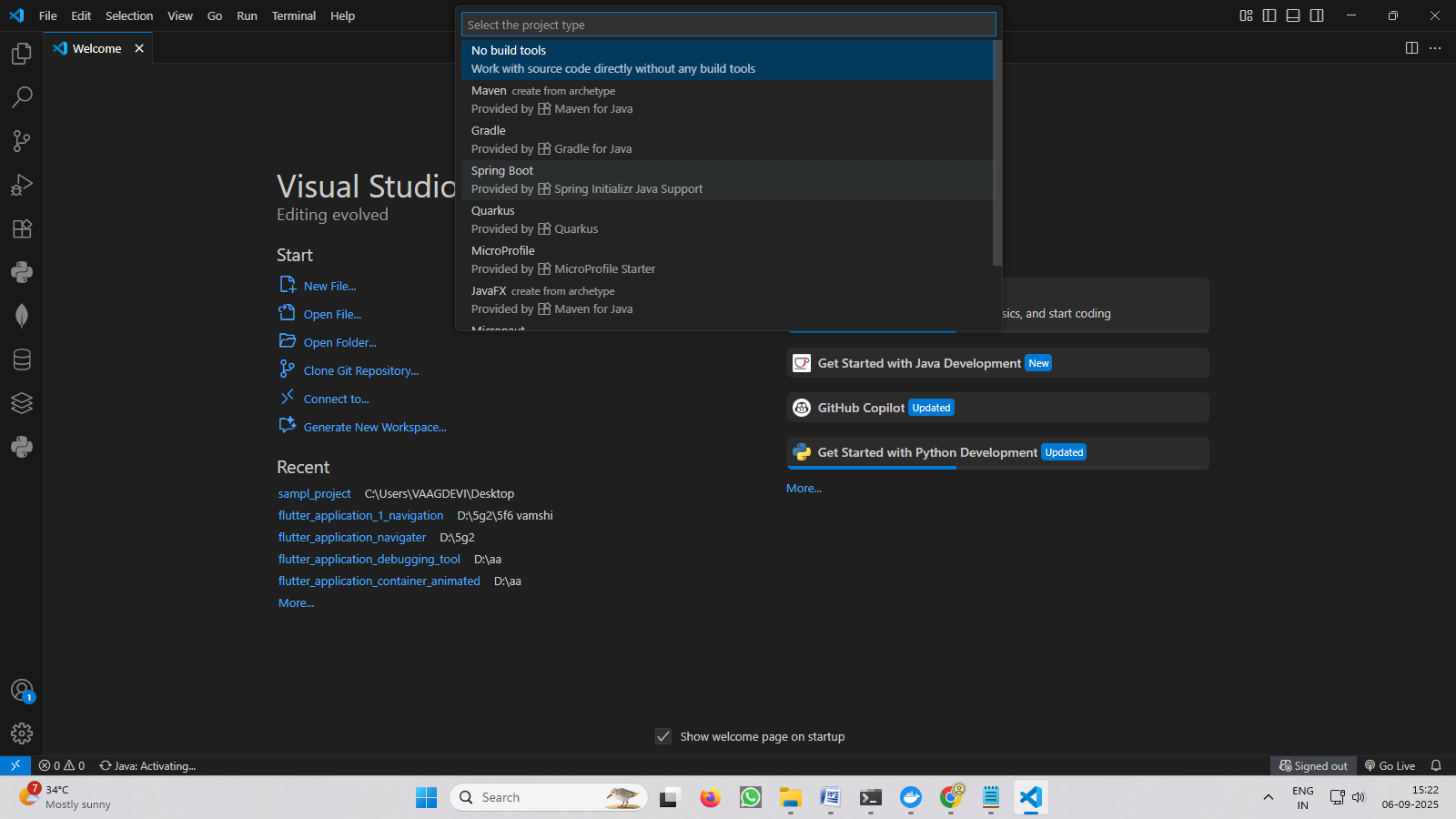
• Once you have your IDE set up, you can write a simple test using the Selenium

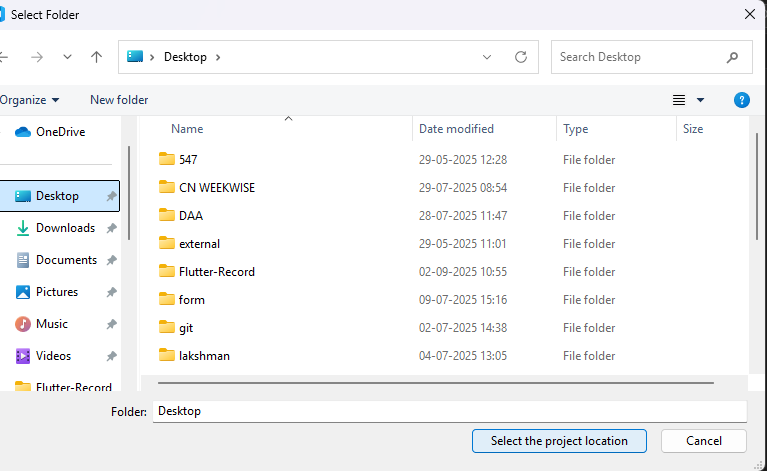
WebDriver. Here’s an example in Java:

How to Create Java Project:

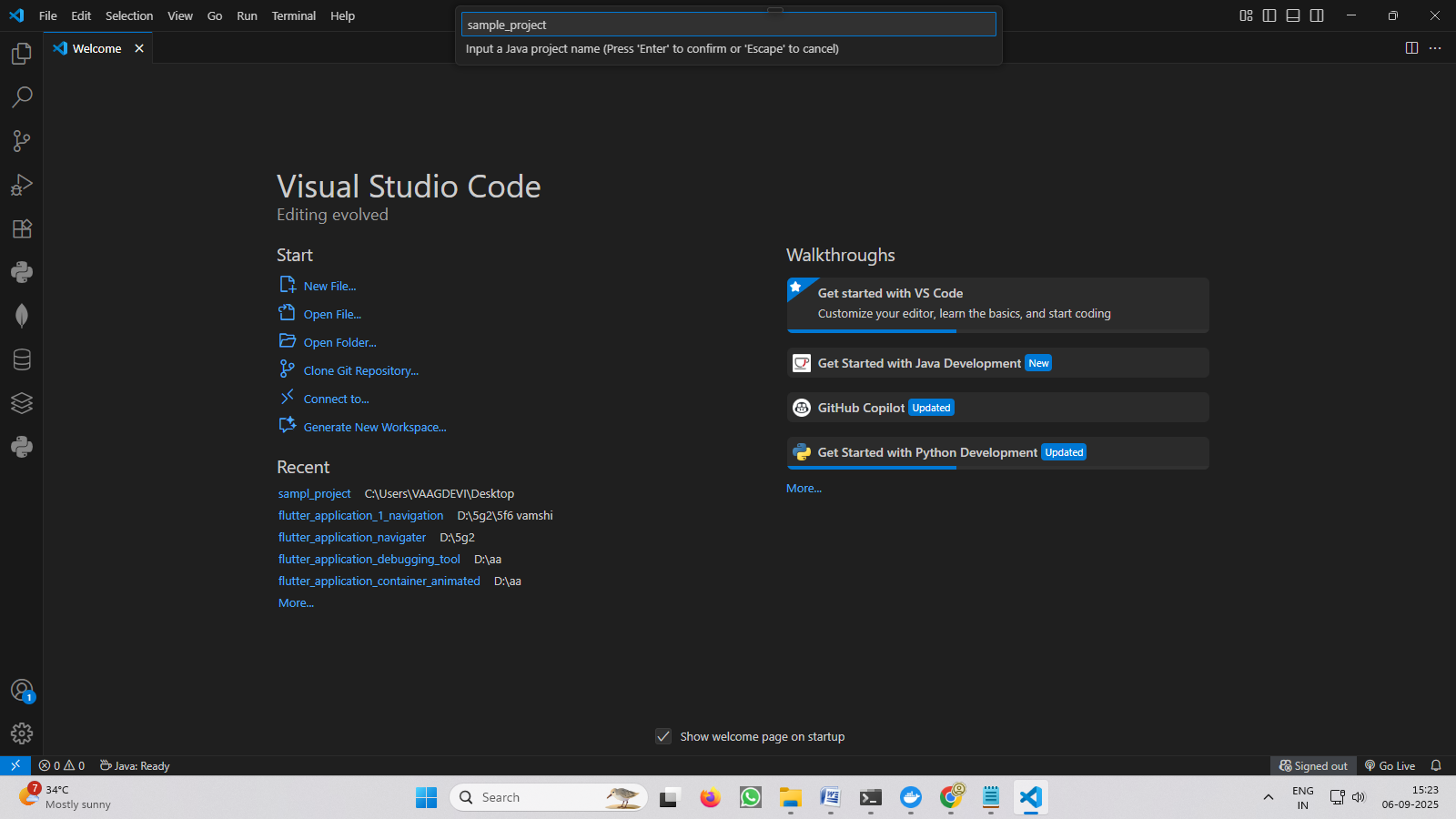
Open Visual Studio code----->Ctrl+Shift+P--- --.> Java:Create Java Project…

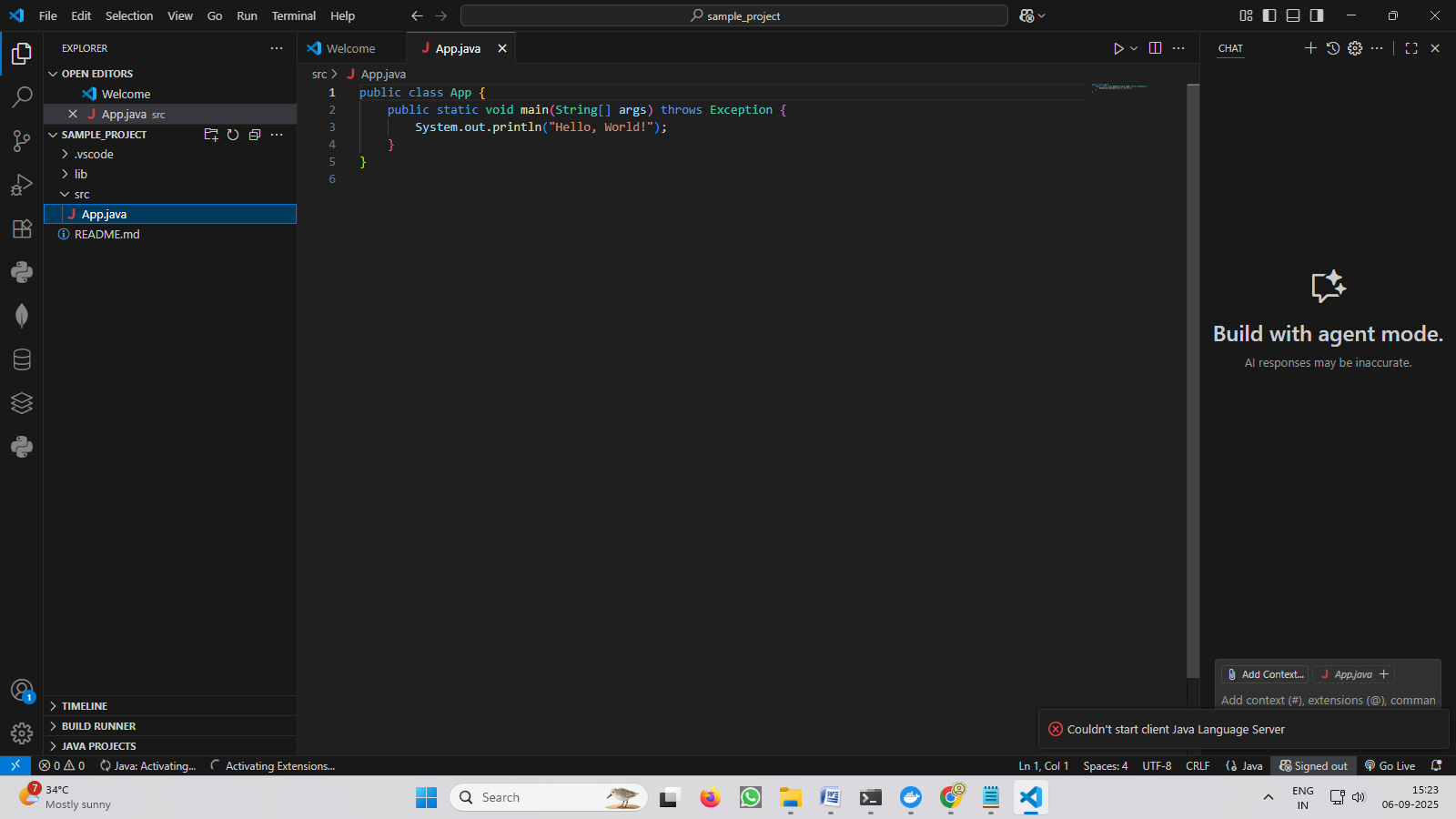




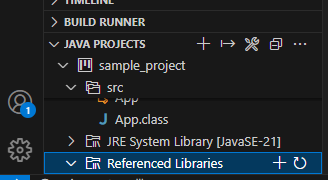


Give the Project Name: **sample\_project**

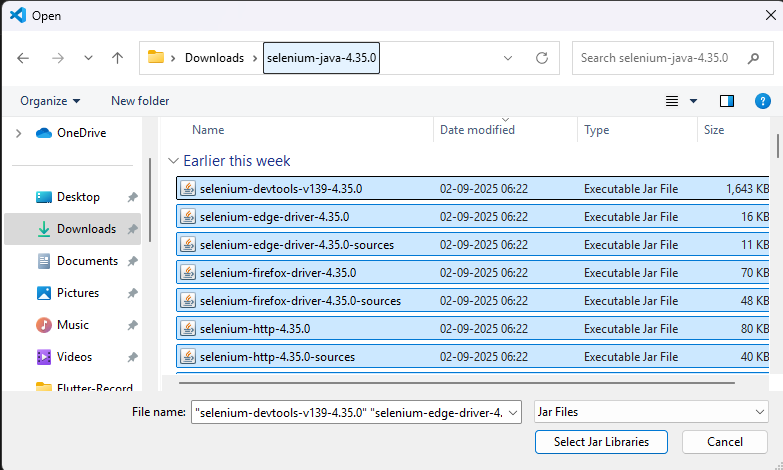




Click on **+** symbol in **Referenced Libraries**



Go to Selenium-java-4.35.0 folder and open the folder and add the all Selenium Jar Libraries /files.



Write the code in App.java

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

public class App {

    public static void main(String[] args) {

        System.setProperty("webdriver.chrome.driver",

                "C:\\Users\\VAAGDEVI\\Downloads\\chromedriver-win64\\chromedriver-win64\\chromedriver.exe");

        WebDriver driver = new ChromeDriver();

        driver.get("https://www.vaagdevi.edu.in/");

        System.out.println("Title: " + driver.getTitle());

        driver.quit();

    }

}

• Run the test:

Run the test by using Run -🡪Start Debugging (OR) F5

This is a basic example of how to get started with Selenium for automated testing. In a real-world scenario, you would likely write more complex tests and organize your code into test suites and test cases, but this example should give you a good starting point for exploring Selenium.