Application Containerization on Docker

The Team

Trapti Gupta

Ekansh Gupta

Harshit Vats

Hansika Sharma

Nikita Kadambala

Introducing the Project

Weather app with
dynamic background

Objective

The Weather App with Dynamic Background is designed to provide users with a user-friendly and visually engaging platform for accessing current weather information.

This application fetches data from the OpenWeatherMap API based on the user's input of a city name and displays it and also changes the background based on the temperature range.

Here are some goals...

01

Provide User-Friendly Weather Information:

The primary goal of this application is to offer users an easy and user-friendly way to access current and forecasted weather information for a location of their choice. By allowing users to input a city name and receive weather data, the app aims to make weather information readily available and comprehensible.

02

Personalize Weather Experience

The app aims to personalize the weather experience for users. By choosing background images that match the temperature range , it creates a more immersive and relatable weather experience, helping users connect with the weather data on a more emotional level.

Methodology

01

Data Retrieval: The project fetches weather data from the OpenWeatherMap API using the city name provided by the user.

02

Temperature Range Mapping: The app categorizes the received temperature data into predefined ranges to determine the appropriate background image.

Methodology..

03

Background Image Selection: It selects and sets the background image based on the temperature range, enhancing user experience and providing visual context.

04

User Interface Update: The app dynamically updates the user interface to display weather information and the selected background image, ensuring a seamless and personalized weather experience.

OpenWeatherMap API Overview

Provider

OpenWeatherMap is a well-known and widely used weather data provider that offers free access to a comprehensive range of weather-related data.

Data Coverage

Access to weather data for locations worldwide, making it a valuable resource for applications with a global user base.

Data Types

Including current weather conditions, forecasts, historical weather data, and more

Authentication

Obtain an API key, which is used for authentication and authorization. The API key ensures secure access to the weather data.

HTTP Requests

The API is accessed through HTTP requests, mainly using GET requests to specific endpoints.

Response Format

The API returns data in JSON format, which is easy to parse and work with in web applications. JSON (JavaScript Object Notation) is a lightweight data interchange format

API Usage in the Weather App

01 Data Retrieval

The API is accessed by making HTTP requests, specifically a GET request, to its endpoints. The API key is included in the request to authenticate and authorize access.

02 City Input

Users provide the name of a city as input through the app's interface, which is then used as a query parameter in the API request to specify the location for which weather data is requested.

03 Data Response

The API responds with a JSON data object that contains a wide range of weather-related information, including current conditions, temperature, humidity, wind speed, and more.

04 Data Parsing

The app parses the JSON response to extract and display specific weather data elements, such as the current weather condition, temperature, and more.

Code snippets

HTML

The HTML code in the project forms the essential structure of the weather app's user interface. It includes input fields where users can enter city names, a "Submit" button to trigger data retrieval, and a dedicated area for presenting weather information. This code is crucial for enabling user interaction and rendering the app's visual elements.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Weather App</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <div class="container">
      <h1>Weather App</h1>
      <div class="glass">
      <input type="text" id="city" placeholder="Enter city name">
      <button id="submitBtn">Submit</button>
      <div id="icon"><img id="wicon" src="" alt="Weather icon"></div>
      <div id="weather-info"></div>

    </div>
    </div>
    <script src="app.js"></script>
  </body>
</html>

```

CSS

- CSS is used to style and format the elements in the HTML, making the app visually appealing.
- It defines the layout, colors, fonts, and styling of the user interface elements, enhancing the app's aesthetics.

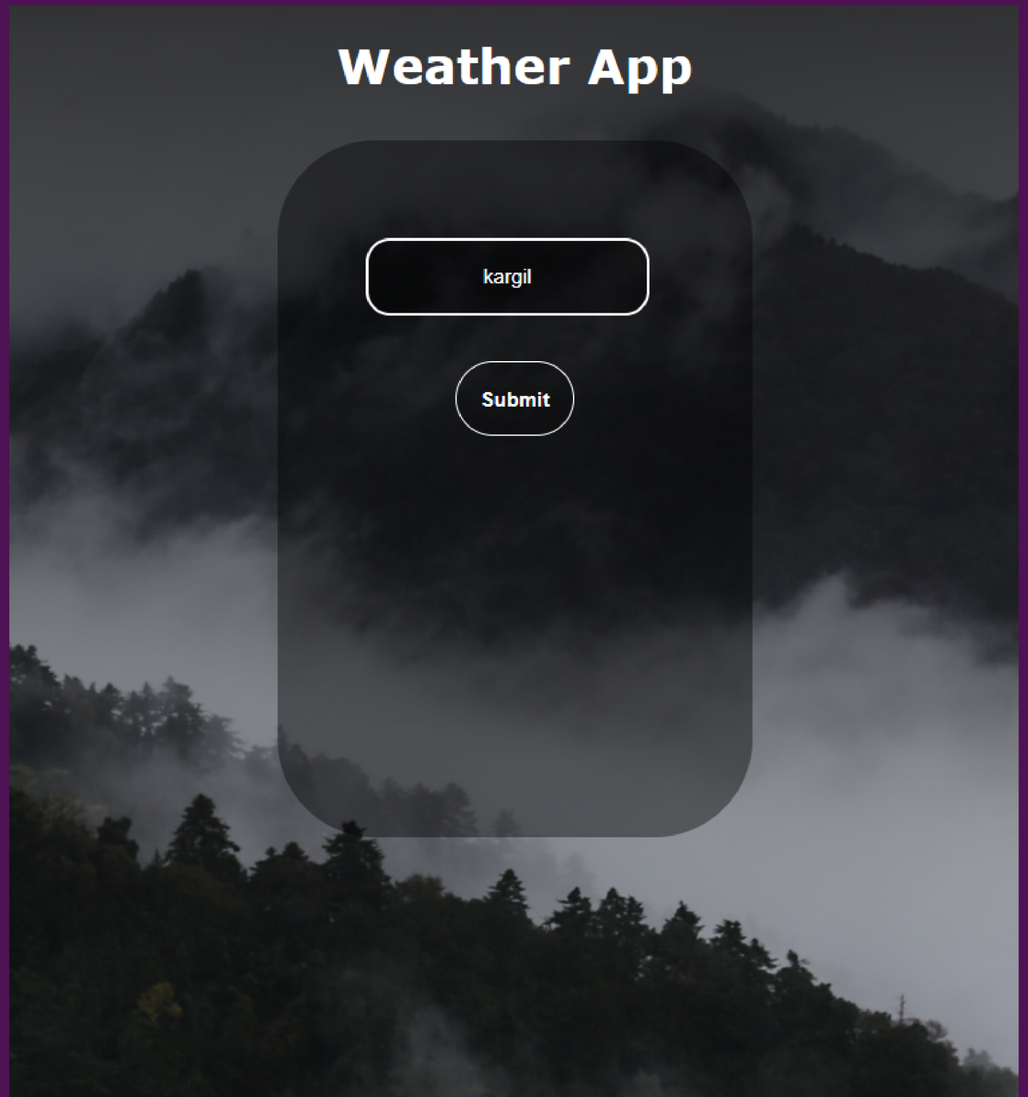
```css
*{
    margin: 0;
    padding:0;
}


body {
    background-image: url('weather.jpg');
    /* overflow: hidden; */
    background-repeat: no-repeat;
    background-position: center center;
    background-attachment: fixed;
    background-size: cover;
    transition: all 1s ease-in-out;
  }
.container {
    display: flex;
    flex-direction: column;
    align-items: center;
    height: 100vh;
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    margin: auto;
    background: linear-gradient(to bottom, rgba(0,0,0,0.79),rgba(0,0,0,0));
}
h1{
    color:white;
    margin-top: 1.9rem;
}
.glass{
    display:flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    padding: 4rem;
    border-radius:4rem;
    background-color: rgba(0, 0, 0, 0.4);
    margin-bottom: 2rem;
    margin:1.9rem;

}

input {
padding: 10px;
border: none;
border-radius: 5px;
```

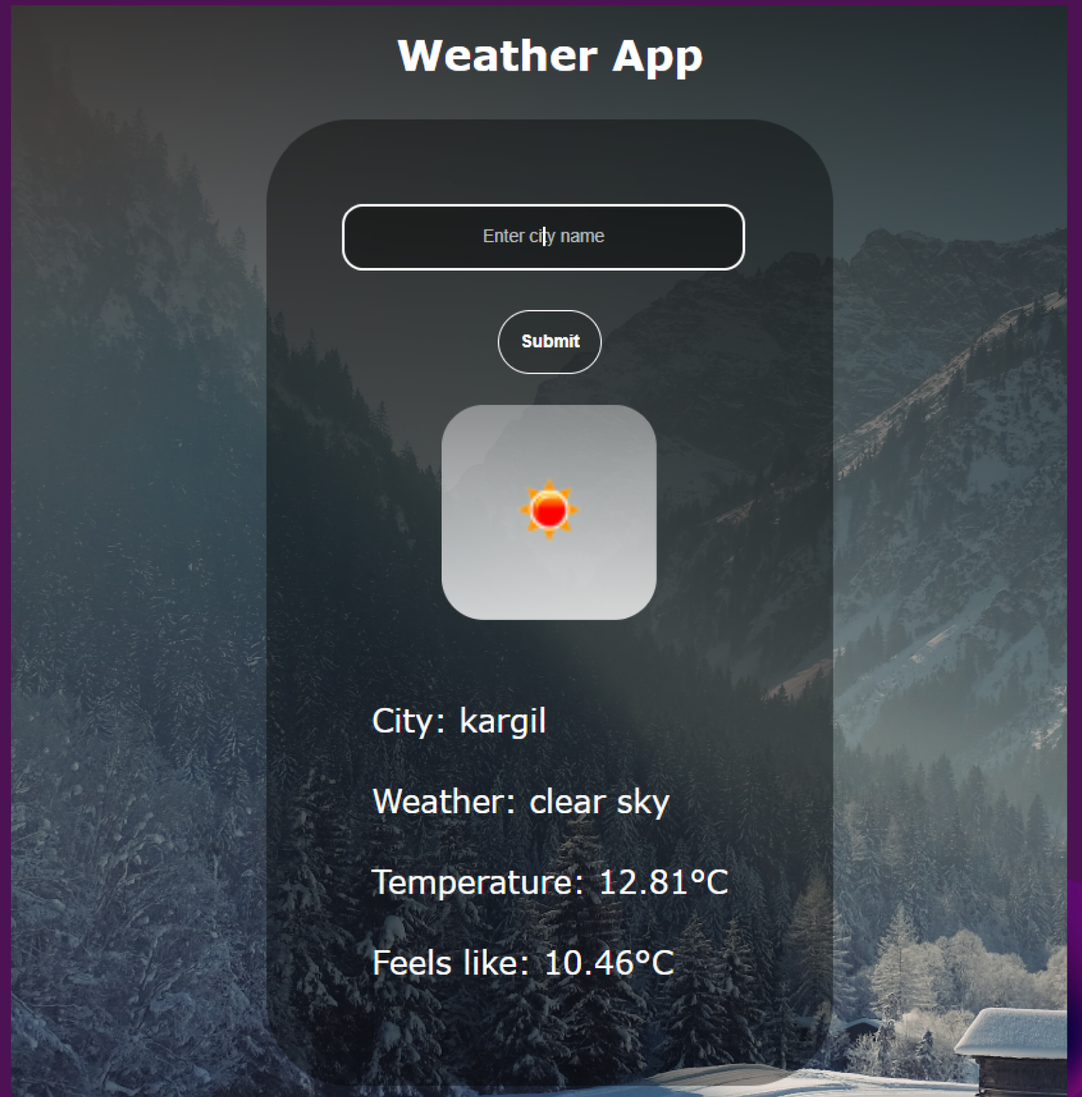
JS

- JavaScript is employed to add functionality and interactivity to the app.
- It handles API calls, data retrieval, parsing, and updating the UI in response to user input.
- Additionally, JavaScript dynamically changes the app's background image based on temperature, improving the user experience.

```js
const apiKey = "1cbc2832563c564f1af798d65f914305";
const submitBtn = document.getElementById("submitBtn");
const cityInput = document.getElementById("city");
const weatherInfoDiv = document.getElementById("weather-info");
const body = document.querySelector("body");

const temperatureRanges = [
  { minTemp: -20, maxTemp: 0, imgUrl: "cold.jpg" },
  { minTemp: 0, maxTemp: 15, imgUrl: "winter.jpg" },
  { minTemp: 15, maxTemp: 30, imgUrl: "summer.jpg" },
  { minTemp: 30, maxTemp: 40, imgUrl: "hot.jpg" },
];
submitBtn.addEventListener("click", () => {
  const city = cityInput.value;
  const temperatureRanges = [
    { minTemp: -20, maxTemp: 0, imgUrl: "cold.jpg" },
    { minTemp: 0, maxTemp: 15, imgUrl: "winter.jpg" },
    { minTemp: 15, maxTemp: 30, imgUrl: "summer.jpg" },
    { minTemp: 30, maxTemp: 40, imgUrl: "hot.jpg" },
  ];

  // Make API call to OpenWeatherMap API
  fetch(
    `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric`
  )
    .then((response) => response.json())
    .then((data) => {
      const weather = data.weather[0].description;
      const temp = data.main.temp;
      const feelsLike = data.main.feels_like;

      var iconcode = data.weather[0].icon;
      var iconurl = "http://openweathermap.org/img/w/" + iconcode + ".png";
      console.log(iconurl);
      document.querySelector('#wicon').src=iconurl;
      document.querySelector('#wicon').style.visibility="visible";
      temperatureRanges.forEach((range) => {
        if (temp >= range.minTemp && temp <= range.maxTemp) {
          body.style.backgroundImage = `url('${range.imgUrl}')`;
          console.log();
        }

      });

```

Output


Weather App
kargil
Submit


Weather App
Enter city name
Submit
City: kargil
Weather: clear sky
Temperature: 12.81°C
Feels like: 10.46°C

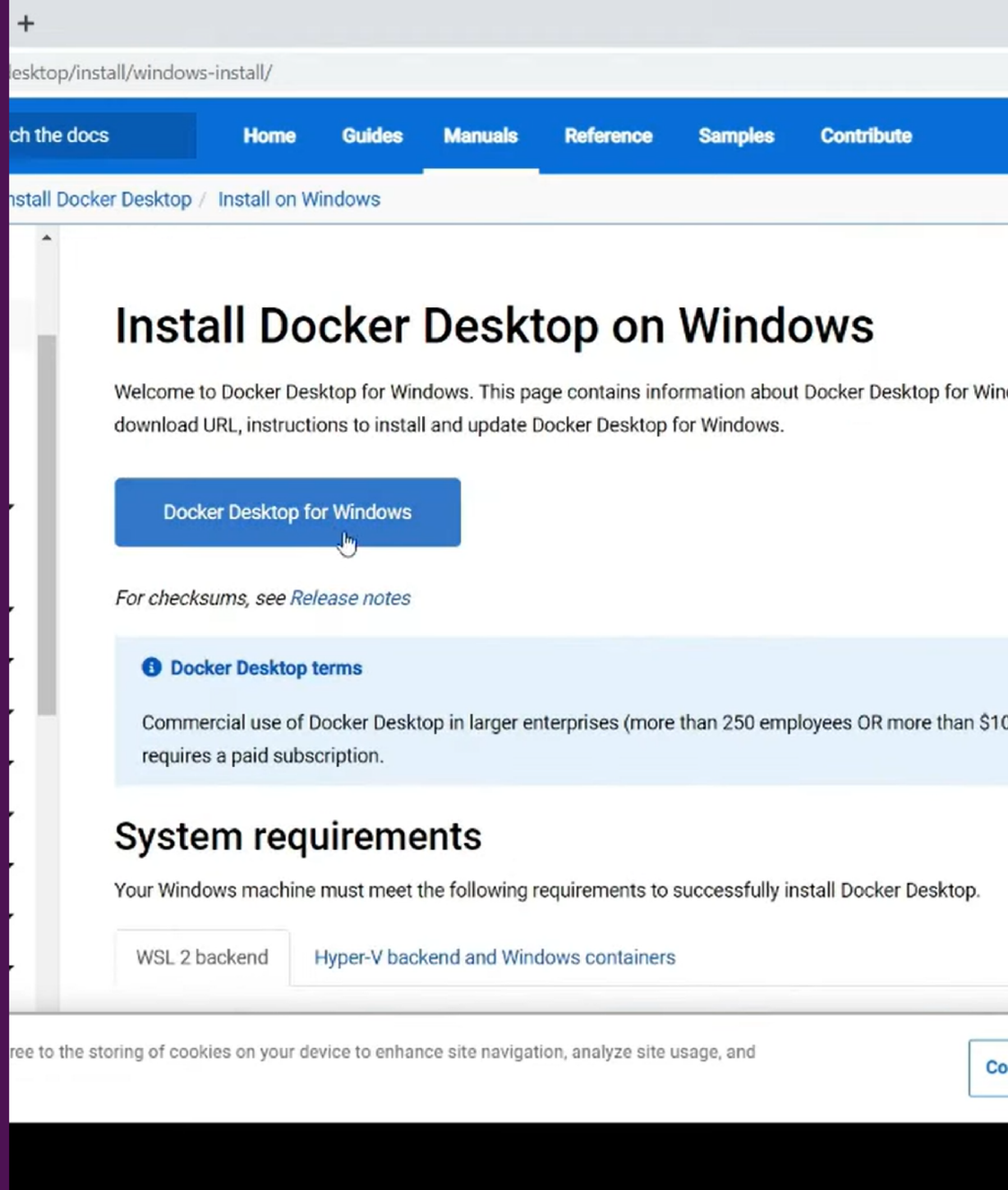
Steps to deploy project in docker

Step 1:

Firstly, we have to Install the docker desktop in our system.

To install the docker, we can directly download it from the given link,

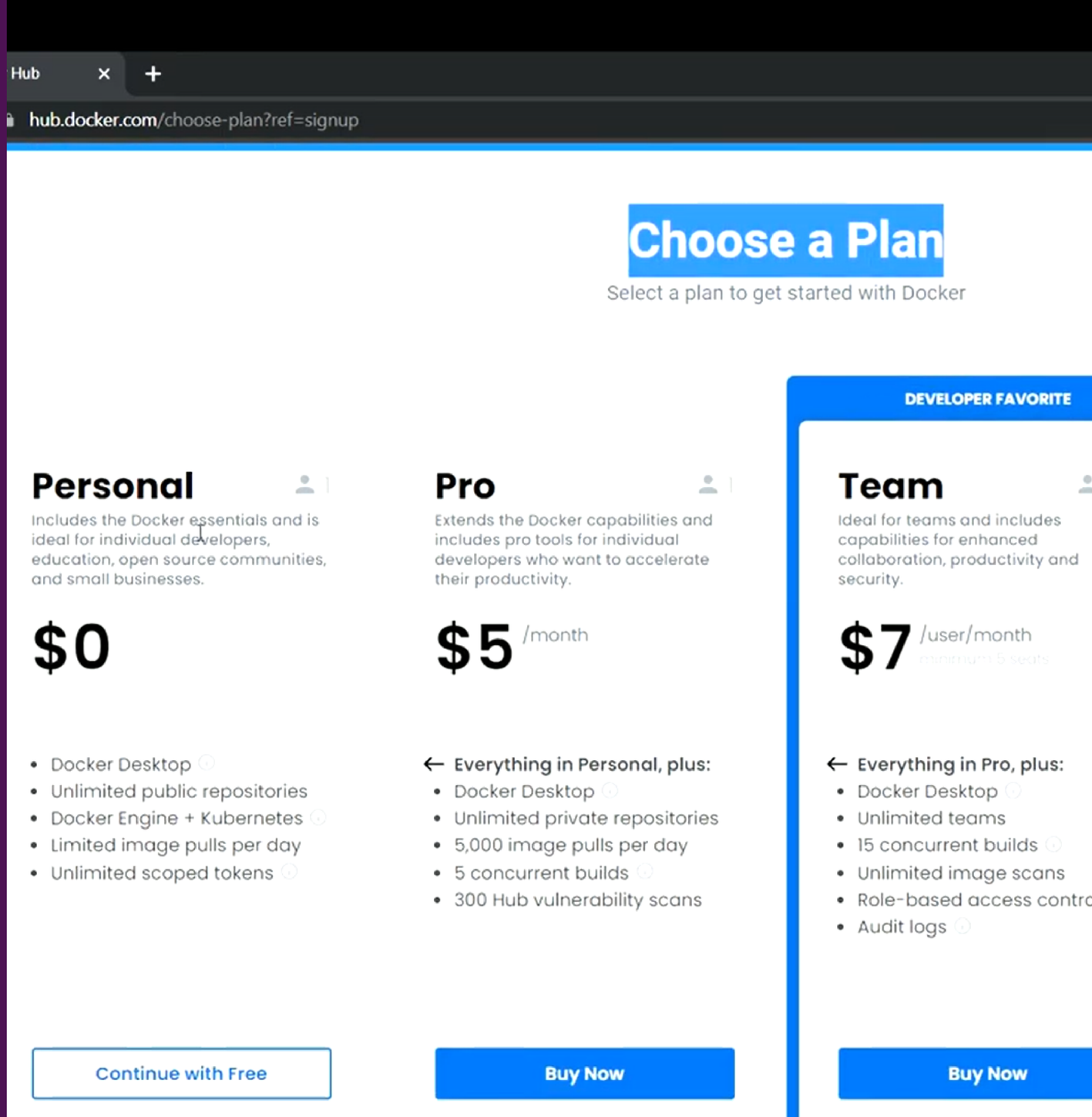
https://docs.docker.com/engine/install/



Step 2:

Then, we have to Log into the docker

After installing, we have to choose a plan to continue.

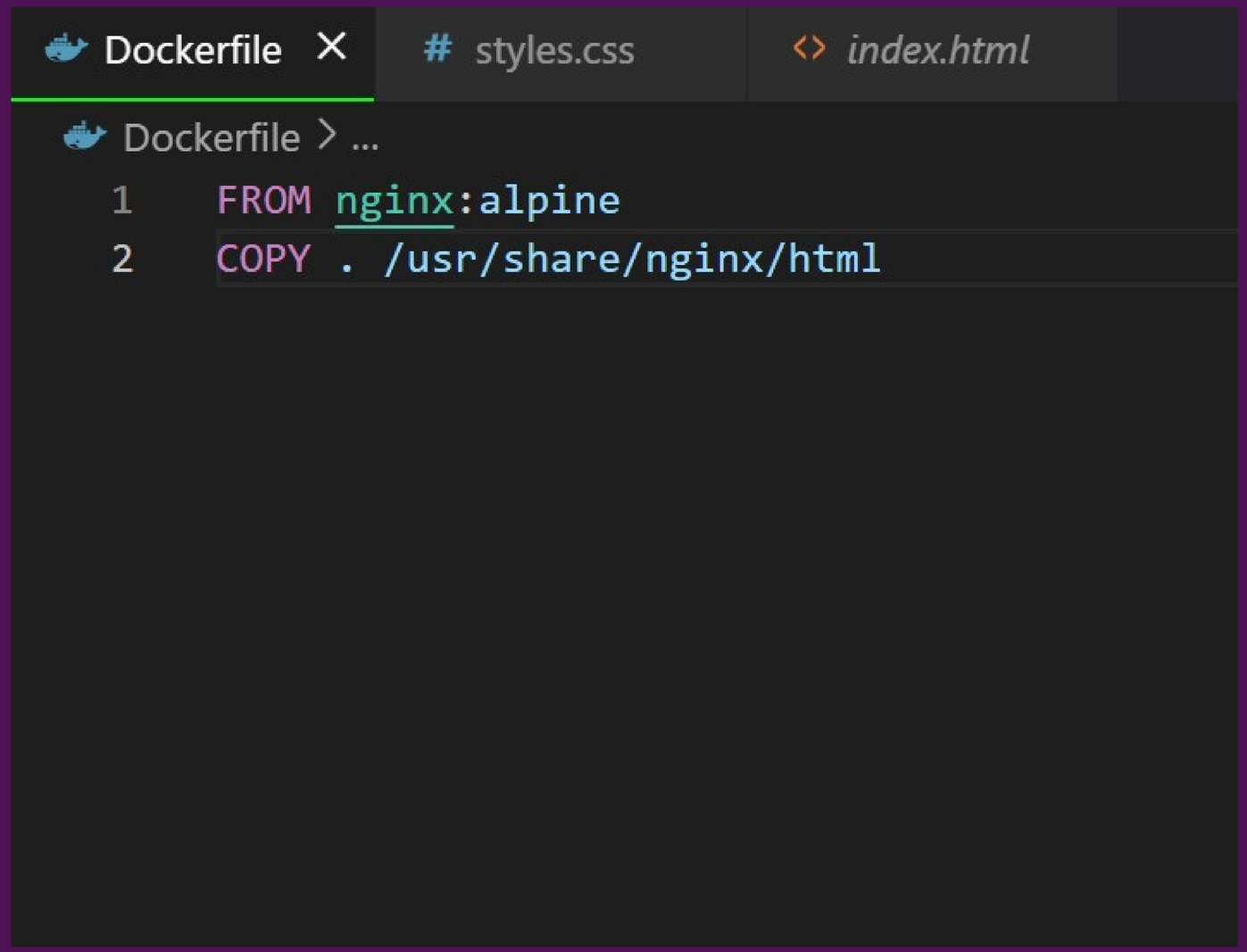


Step 3:

After that, we will create a Dockerfile inside our project folder.

After creating the file, add the following content to it,

FROM nginx:alpine
COPY . /usr/share/nginx.html

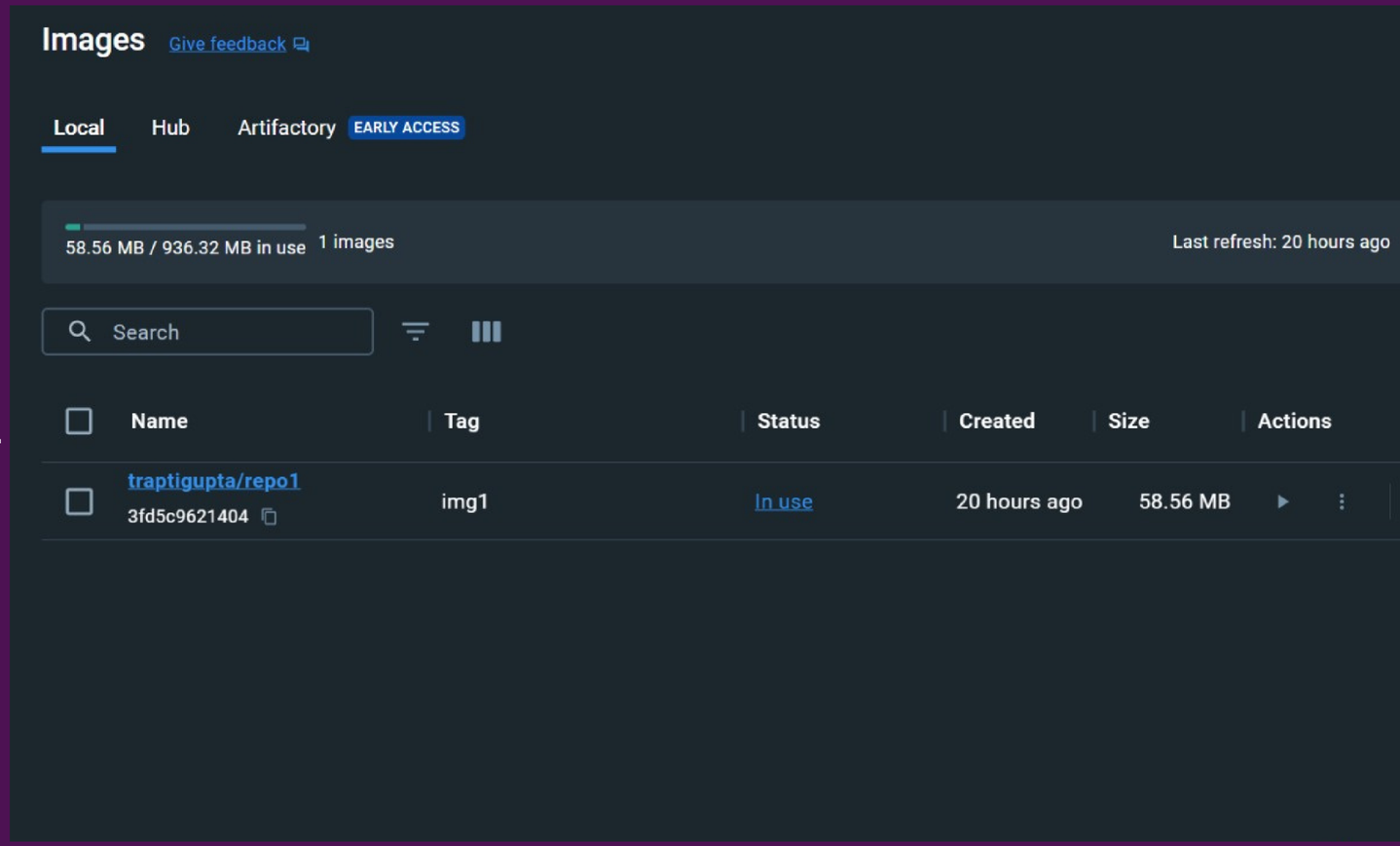


Step 4:

After saving the content, we have to run the given command,

docker build -t traptigupta/repo1:img1

After running the command, we can see this image in DockerDesktop.

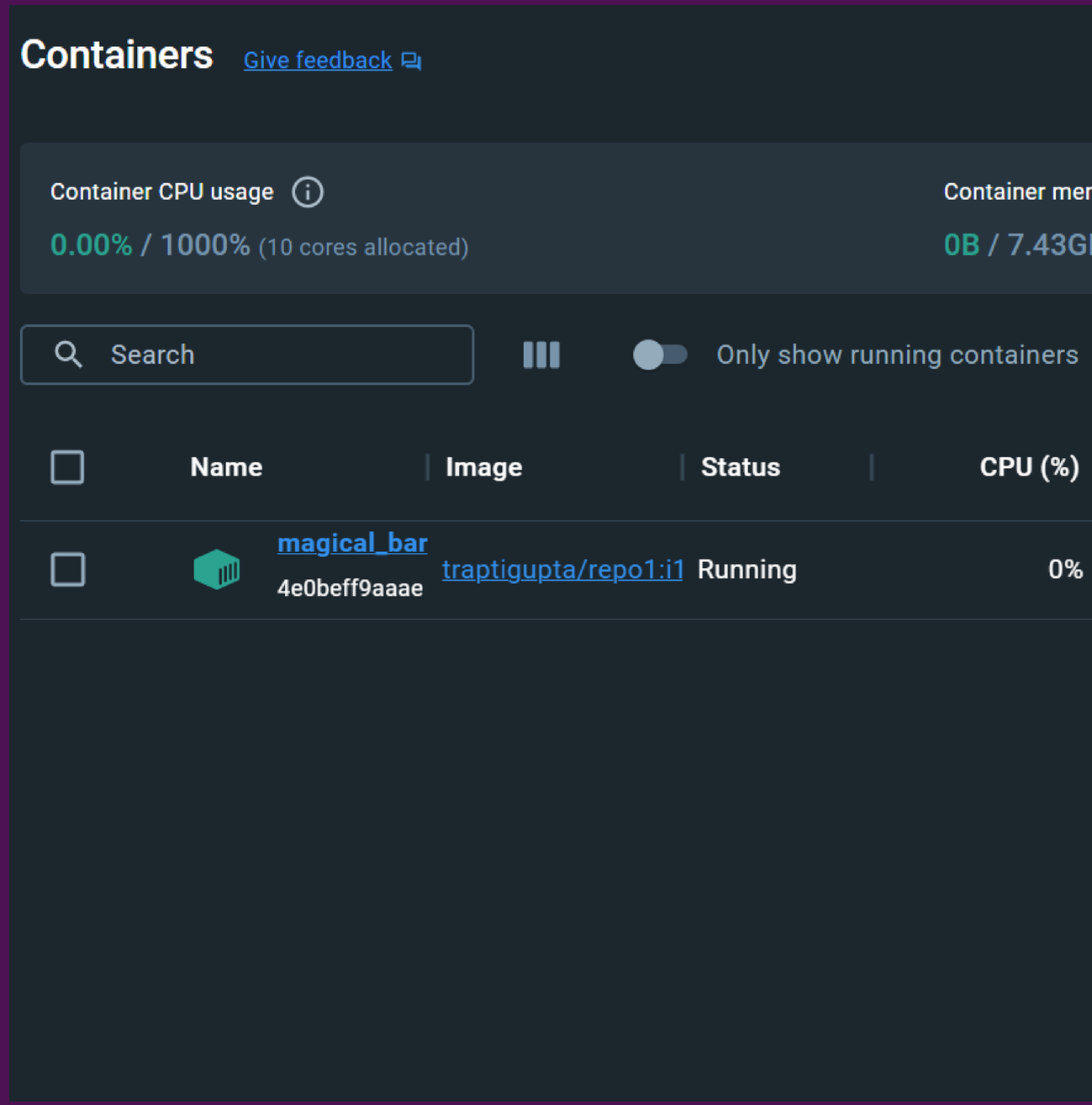


Step 5:

Now we have to run the created image by writing the command given below.

docker run -d -p 80:80 traptigupta/repo1:img1

After runing a image, a container will be created for our image.

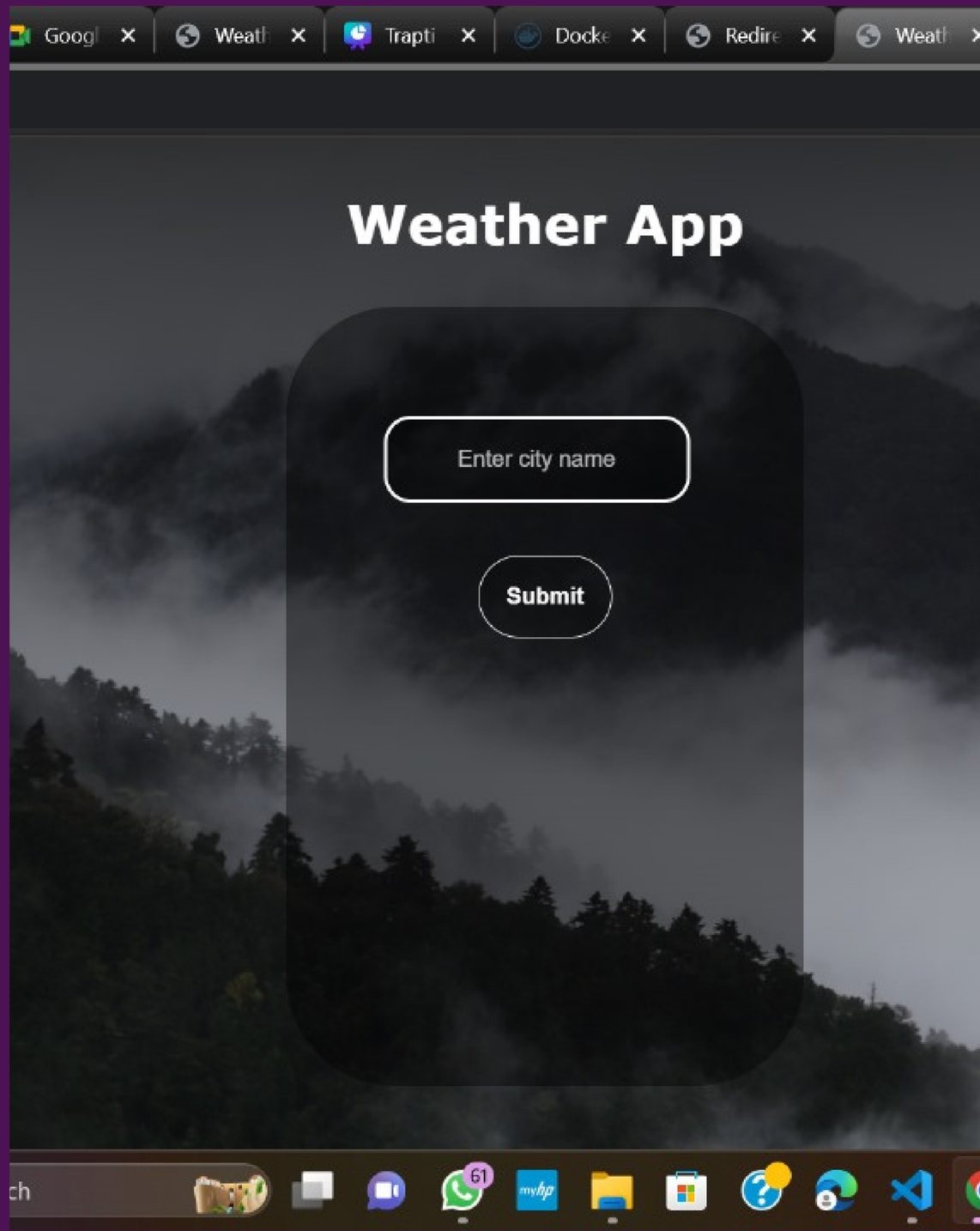


Step 6:

Now open chrome as we have provided port 80 while runing our image.

So type url **localhost:80** browser.

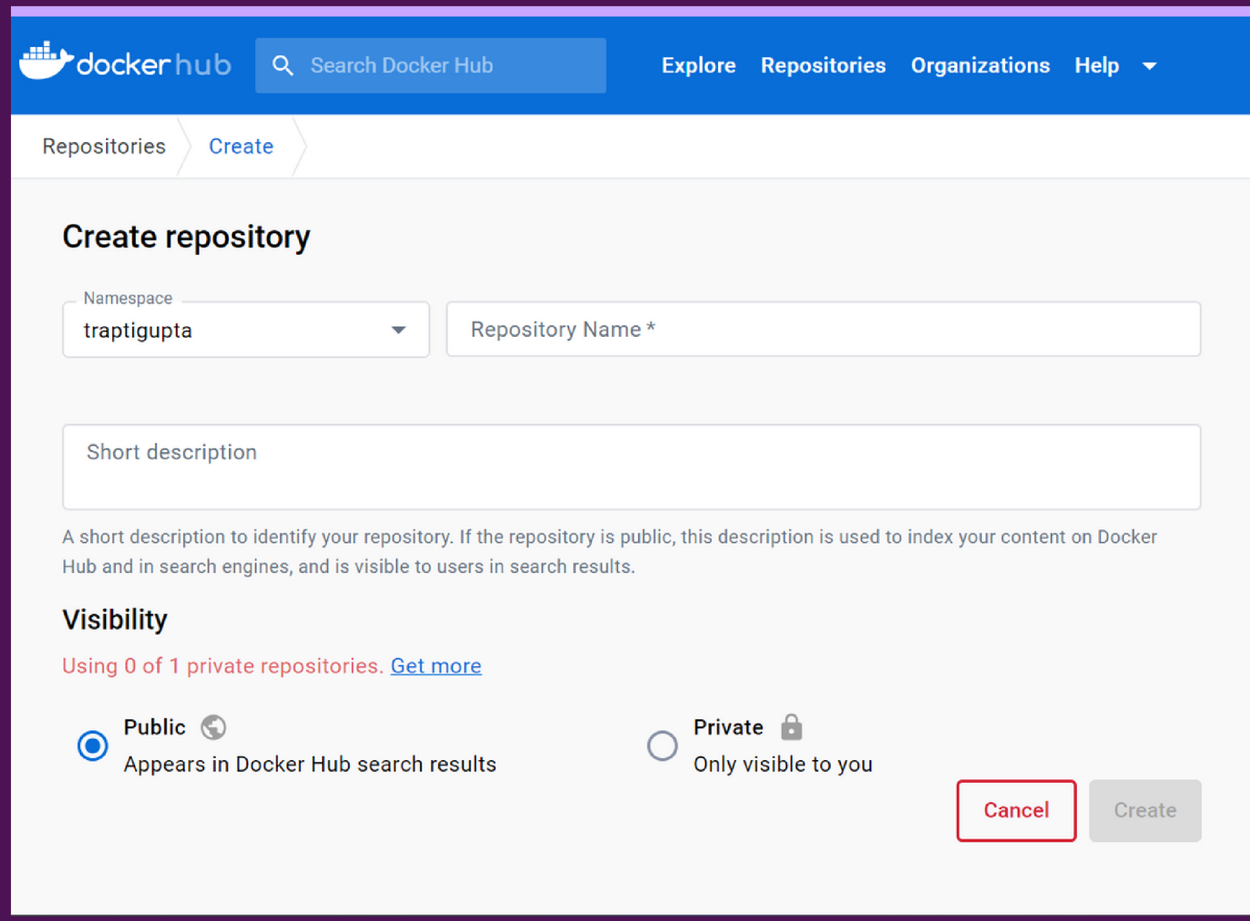
Then our project will open in our browser.



Step 7:

Login into the account

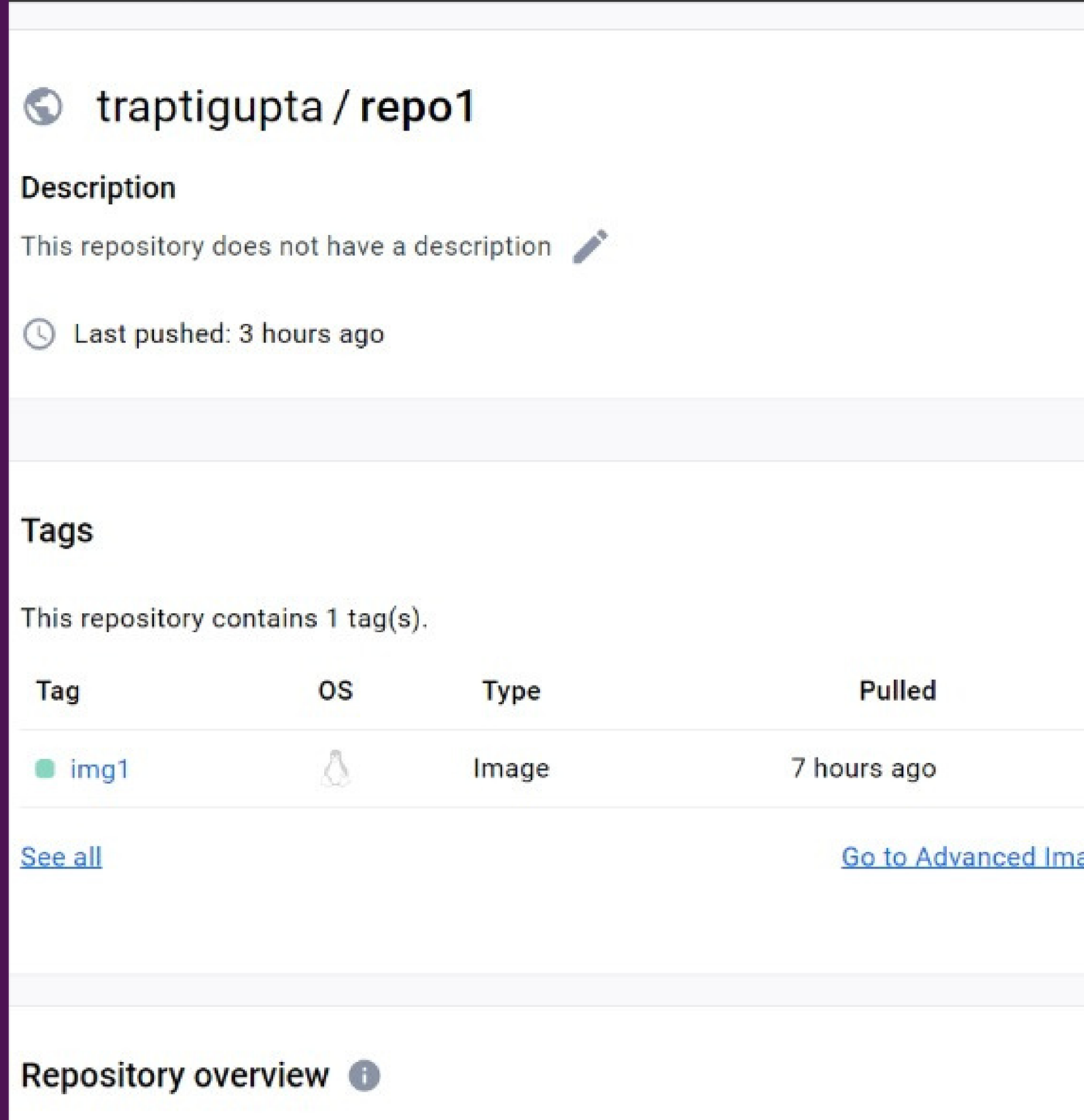
Create a repository with public accessibility by providing the repo name.



Step 8:

Push your project image into the repository you created using command:

docker push traptigupta/repo1:img1



Step 9:

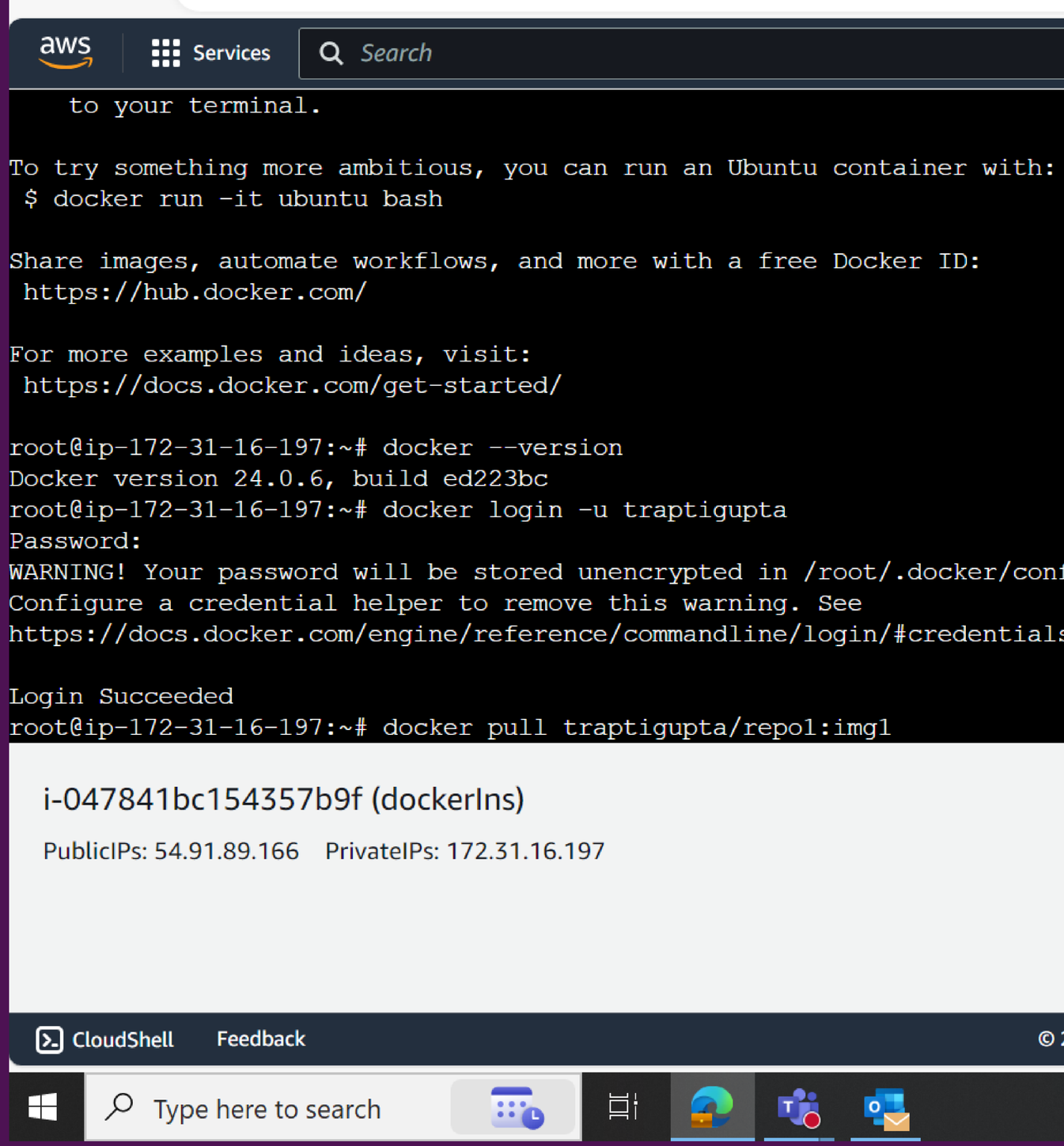
Create an EC2 instance using ubuntu AMI.

connect to the instance and install docker into it.

Step 10:

Login into your docker in ubuntu instancse by :

docker login -u traptigupta



Step 11:

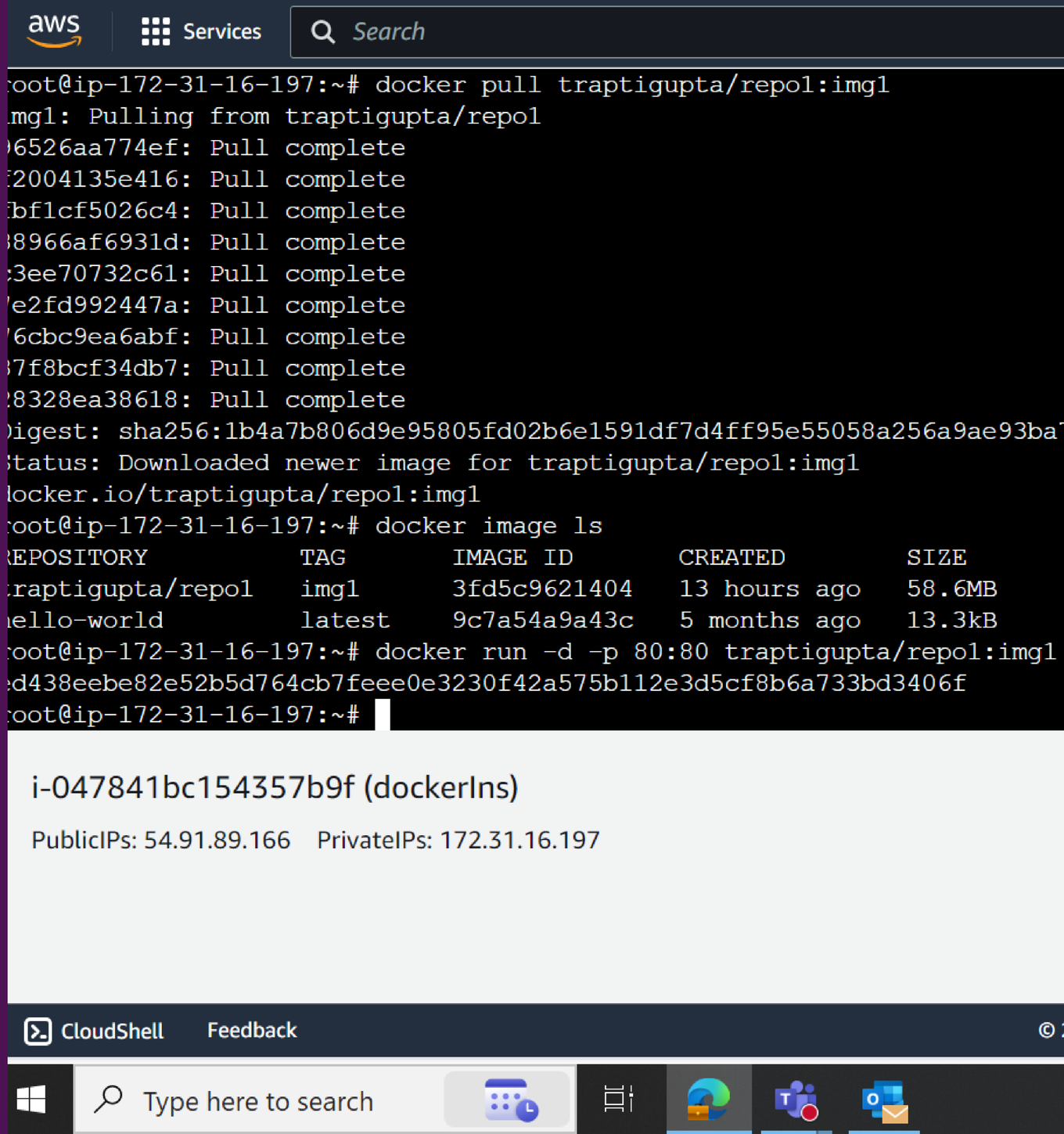
Pull your image to the instance

docker pull traptigupta/repo1:img1

Step 12:

run image on instance

docker run -d -p 80:80 traptigupta/repo1:img1



Now project will be available on public ip of our instance

Public IP : 54.91.89.166

thank you!