

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.

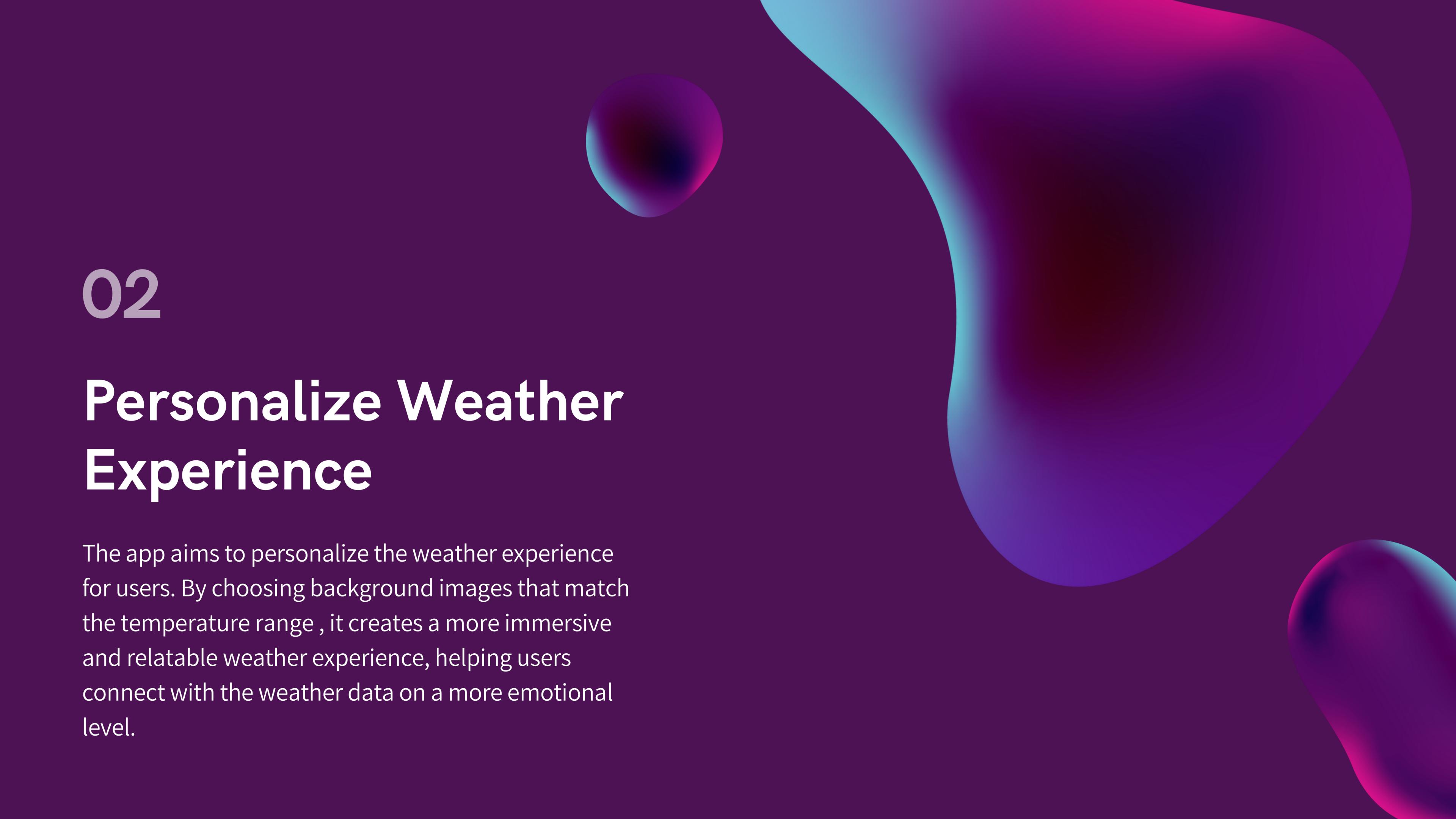
The background features a dark purple gradient with three large, semi-transparent overlapping circles. One circle is light blue at the top and transitions to pink at the bottom. Another is pink at the top and transitions to light blue at the bottom. A third, smaller circle is located in the bottom right corner, partially cut off by the frame.

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.

Authentication

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

Data Coverage

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

HTTP Requests

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

Data Types

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

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.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Weather App</title>
5          <link rel="stylesheet" href="styles.css">
6      </head>
7      <body>
8          <div class="container">
9              <h1>Weather App</h1>
10             <div class="glass">
11                 <input type="text" id="city" placeholder="Enter city name">
12                 <button id="submitBtn">Submit</button>
13             <div id="icon"><img id="wicon" src="" alt="Weather icon"></div>
14             <div id="weather-info"></div>
15
16         </div>
17     </div>
18     <script src="app.js"></script>
19
20 </body>
21 </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.

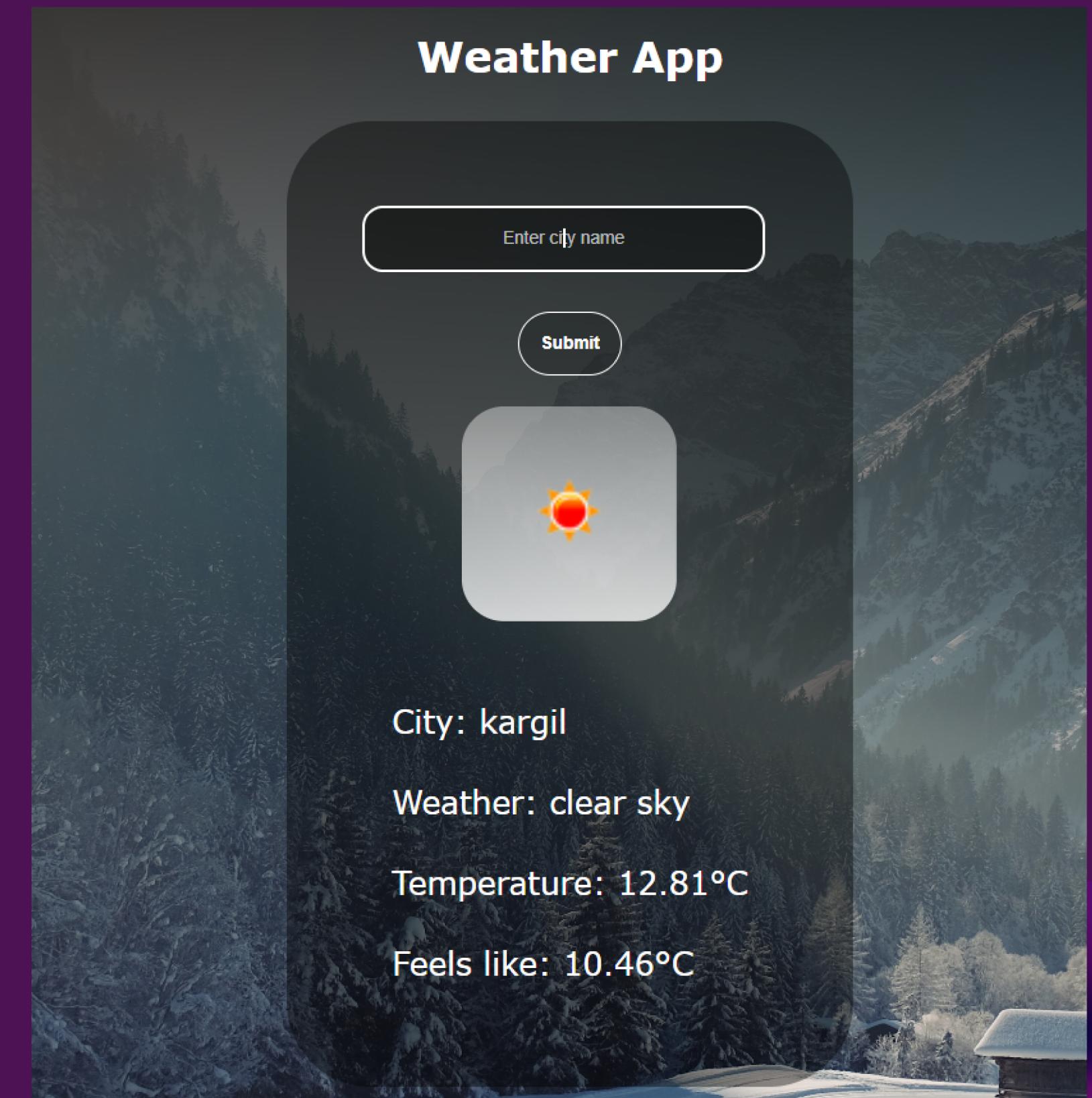
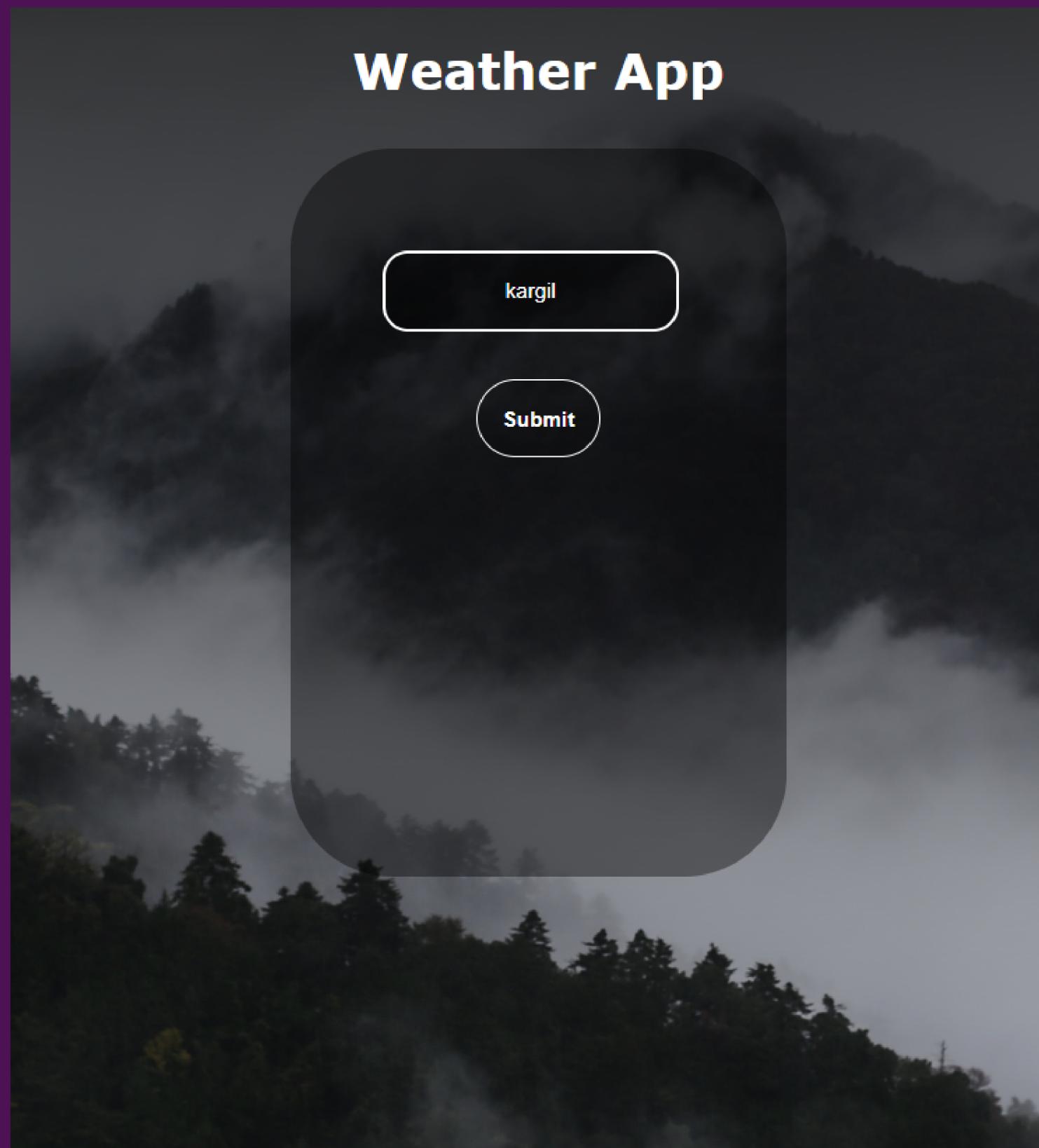
```
1  *{
2      margin: 0;
3      padding:0;
4  }
5
6
7  body {
8      background-image: url('weather.jpg');
9      /* overflow: hidden; */
10     background-repeat: no-repeat;
11     background-position: center center;
12     background-attachment: fixed;
13     background-size: cover;
14     transition: all 1s ease-in-out;
15  }
16 .container {
17     display: flex;
18     flex-direction: column;
19     align-items: center;
20     height: 100vh;
21     font-family: Verdana, Geneva, Tahoma, sans-serif;
22     margin: auto;
23     background: linear-gradient(to bottom, rgba(0,0,0,0.79),rgba(0,0,0,0));
24  }
25 h1{
26     color:white;
27     margin-top: 1.9rem;
28  }
29 .glass{
30     display:flex;
31     flex-direction: column;
32     justify-content: center;
33     align-items: center;
34     padding: 4rem;
35     border-radius:4rem;
36     background-color: rgba(0, 0, 0, 0.4);
37     margin-bottom: 2rem;
38     margin:1.9rem;
39  }
40  }
41
42  input {
43     padding: 10px;
44     border: none;
45     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.

```
1 const apiKey = "1cbc2832563c564f1af798d65f914305";
2 const submitBtn = document.getElementById("submitBtn");
3 const cityInput = document.getElementById("city");
4 const weatherInfoDiv = document.getElementById("weather-info");
5 const body = document.querySelector("body");
6
7 const temperatureRanges = [
8   { minTemp: -20, maxTemp: 0, imgUrl: "cold.jpg" },
9   { minTemp: 0, maxTemp: 15, imgUrl: "winter.jpg" },
10  { minTemp: 15, maxTemp: 30, imgUrl: "summer.jpg" },
11  { minTemp: 30, maxTemp: 40, imgUrl: "hot.jpg" },
12];
13 submitBtn.addEventListener("click", () => {
14   const city = cityInput.value;
15   const temperatureRanges = [
16     { minTemp: -20, maxTemp: 0, imgUrl: "cold.jpg" },
17     { minTemp: 0, maxTemp: 15, imgUrl: "winter.jpg" },
18     { minTemp: 15, maxTemp: 30, imgUrl: "summer.jpg" },
19     { minTemp: 30, maxTemp: 40, imgUrl: "hot.jpg" },
20   ];
21
22 // Make API call to OpenWeatherMap API
23 fetch(
24   `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric`
25 )
26 .then((response) => response.json())
27 .then((data) => {
28   const weather = data.weather[0].description;
29   const temp = data.main.temp;
30   const feelsLike = data.main.feels_like;
31
32   var iconcode = data.weather[0].icon;
33   var iconurl = "http://openweathermap.org/img/w/" + iconcode + ".png";
34   console.log(iconurl);
35   document.querySelector('#wicon').src=iconurl;
36   document.querySelector('#wicon').style.visibility="visible";
37   temperatureRanges.forEach((range) => {
38     if (temp >= range.minTemp && temp <= range.maxTemp) {
39       body.style.backgroundImage = `url('${range.imgUrl}')`;
40       console.log();
41     }
42   });
43 });
44
```

Output



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/>

The screenshot shows a web browser displaying the Docker Desktop for Windows installation guide. The URL in the address bar is <https://docs.docker.com/desktop/install/windows-install/>. The page has a blue header with navigation links: 'Search the docs' (which is highlighted), 'Home', 'Guides', 'Manuals', 'Reference', 'Samples', and 'Contribute'. Below the header, there's a breadcrumb trail: 'Install Docker Desktop / Install on Windows'. The main title is 'Install Docker Desktop on Windows'. A sub-section title 'Welcome to Docker Desktop for Windows' follows, along with a note about the download URL, instructions to install and update Docker Desktop for Windows. A prominent blue button labeled 'Docker Desktop for Windows' is shown with a cursor hovering over it. Below the button, a link 'For checksums, see [Release notes](#)' is provided. A section titled 'Docker Desktop terms' includes a note that commercial use in larger enterprises requires a paid subscription. Another section titled 'System requirements' lists the necessary Windows machine requirements to successfully install Docker Desktop. At the bottom, there's a note about cookie storage and a 'Cookie settings' button.

Install Docker Desktop on Windows

Welcome to Docker Desktop for Windows. This page contains information about Docker Desktop for Windows, download URL, instructions to install and update Docker Desktop for Windows.

Docker Desktop for Windows

For checksums, see [Release notes](#)

Docker Desktop terms

Commercial use of Docker Desktop in larger enterprises (more than 250 employees OR more than \$10M in annual revenue) requires a paid subscription.

System requirements

Your Windows machine must meet the following requirements to successfully install Docker Desktop.

WSL 2 backend Hyper-V backend and Windows containers

Cookie settings

Step 2:

Then, we have to Log into the docker

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

The screenshot shows a web browser window with the URL hub.docker.com/choose-plan?ref=signup. The page has a header "Hub" and a title "Choose a Plan". Below the title is a subtitle "Select a plan to get started with Docker". There are three main plan options:

- Personal** (\$0): Includes Docker essentials for individual developers, education, and small businesses. It includes Docker Desktop, unlimited public repositories, Docker Engine + Kubernetes, limited image pulls per day, and unlimited scoped tokens.
- Pro** (\$5 /month): Extends Docker capabilities for individual developers, including pro tools for accelerating productivity. It includes everything in Personal plus unlimited private repositories, 5,000 image pulls per day, 5 concurrent builds, 300 Hub vulnerability scans, and Docker Desktop.
- Team** (\$7 /month): Ideal for teams, including enhanced collaboration, productivity, and security. It includes everything in Pro plus unlimited teams, 15 concurrent builds, unlimited image scans, role-based access control, and audit logs.

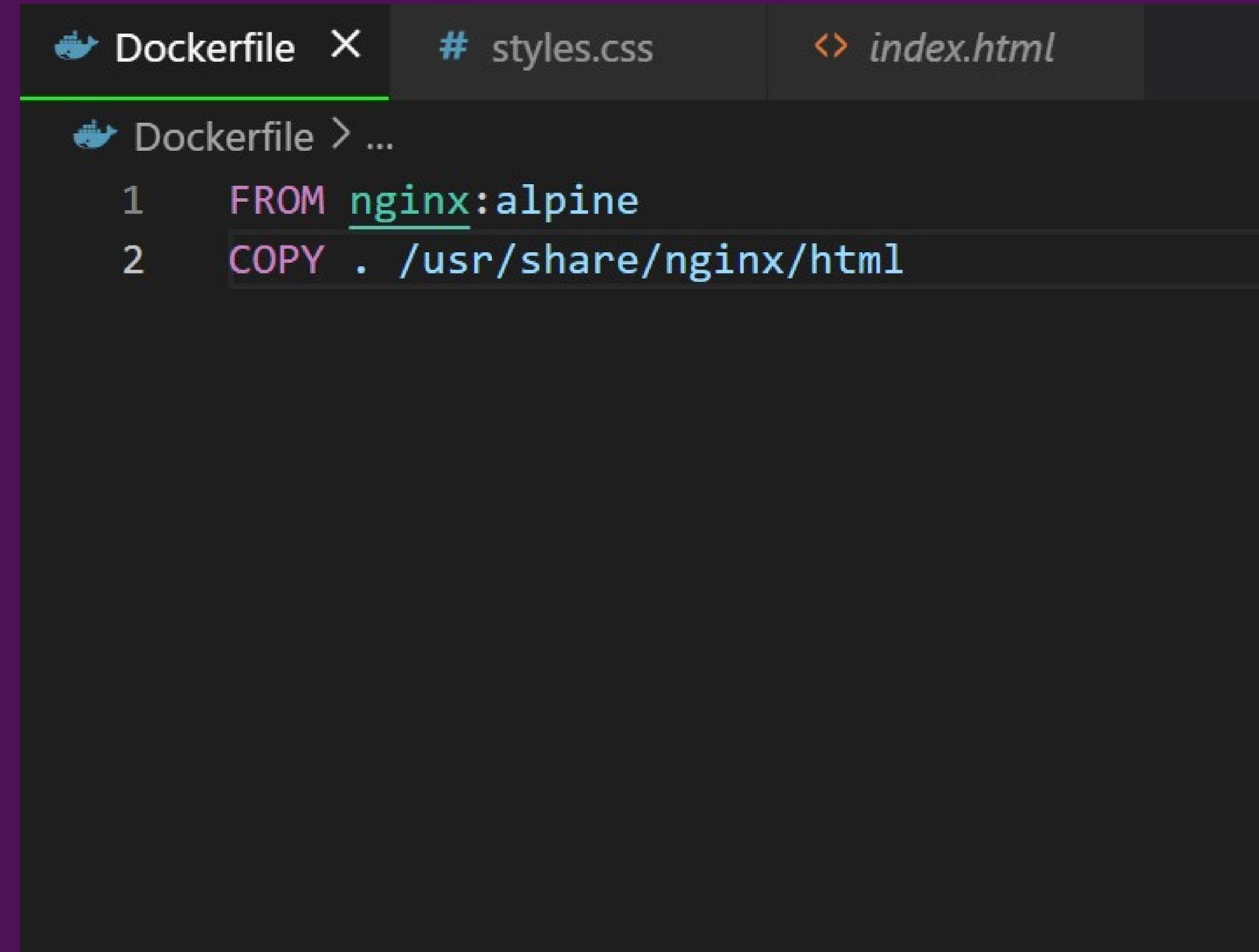
Each plan has a "Buy Now" button at the bottom.

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
```



The screenshot shows a code editor interface with three tabs at the top: 'Dockerfile' (selected), '# styles.css', and '<> index.html'. The 'Dockerfile' tab contains the following code:

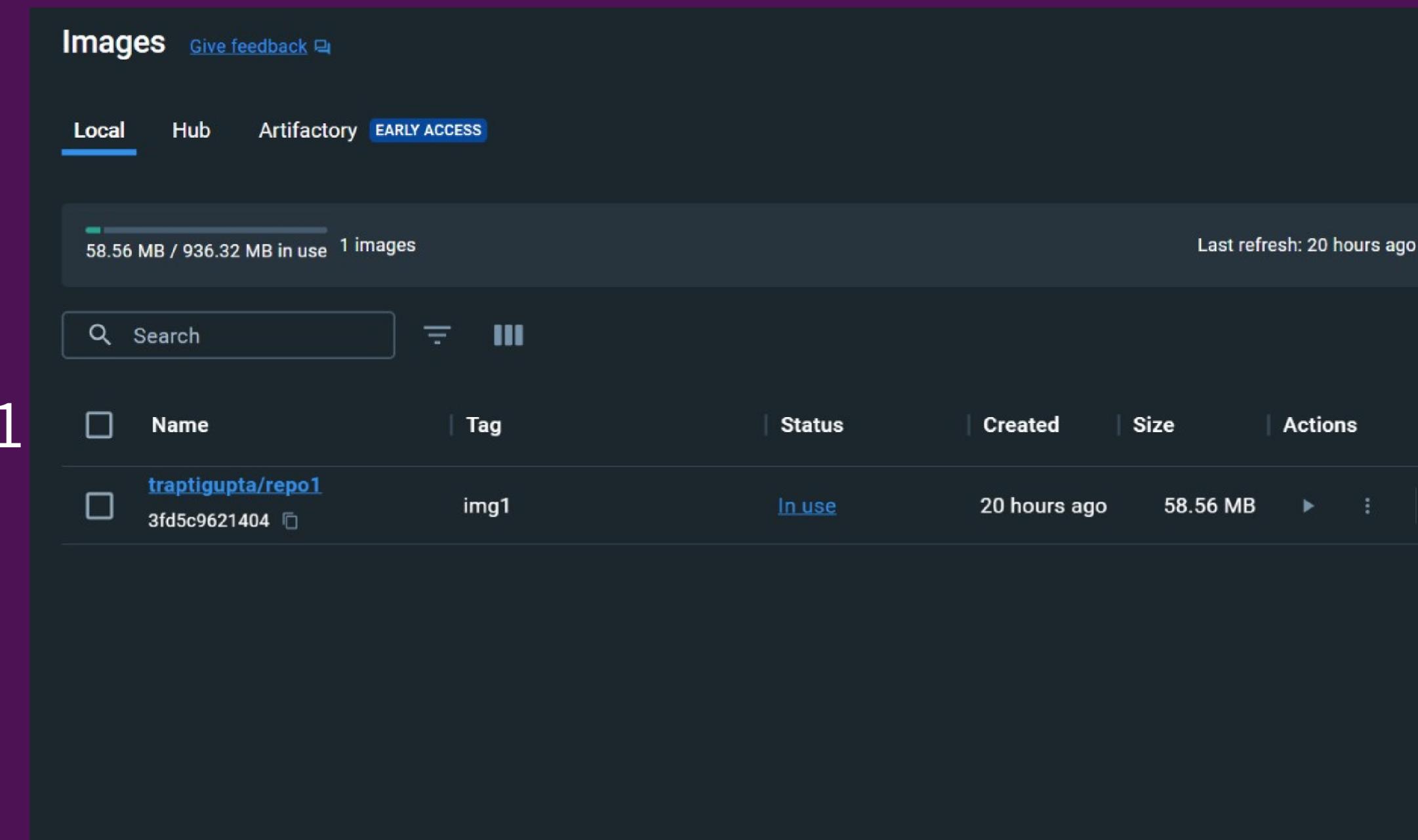
```
1 FROM nginx:alpine  
2 COPY . /usr/share/nginx/html
```

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 running a image, a container will be created for our image.

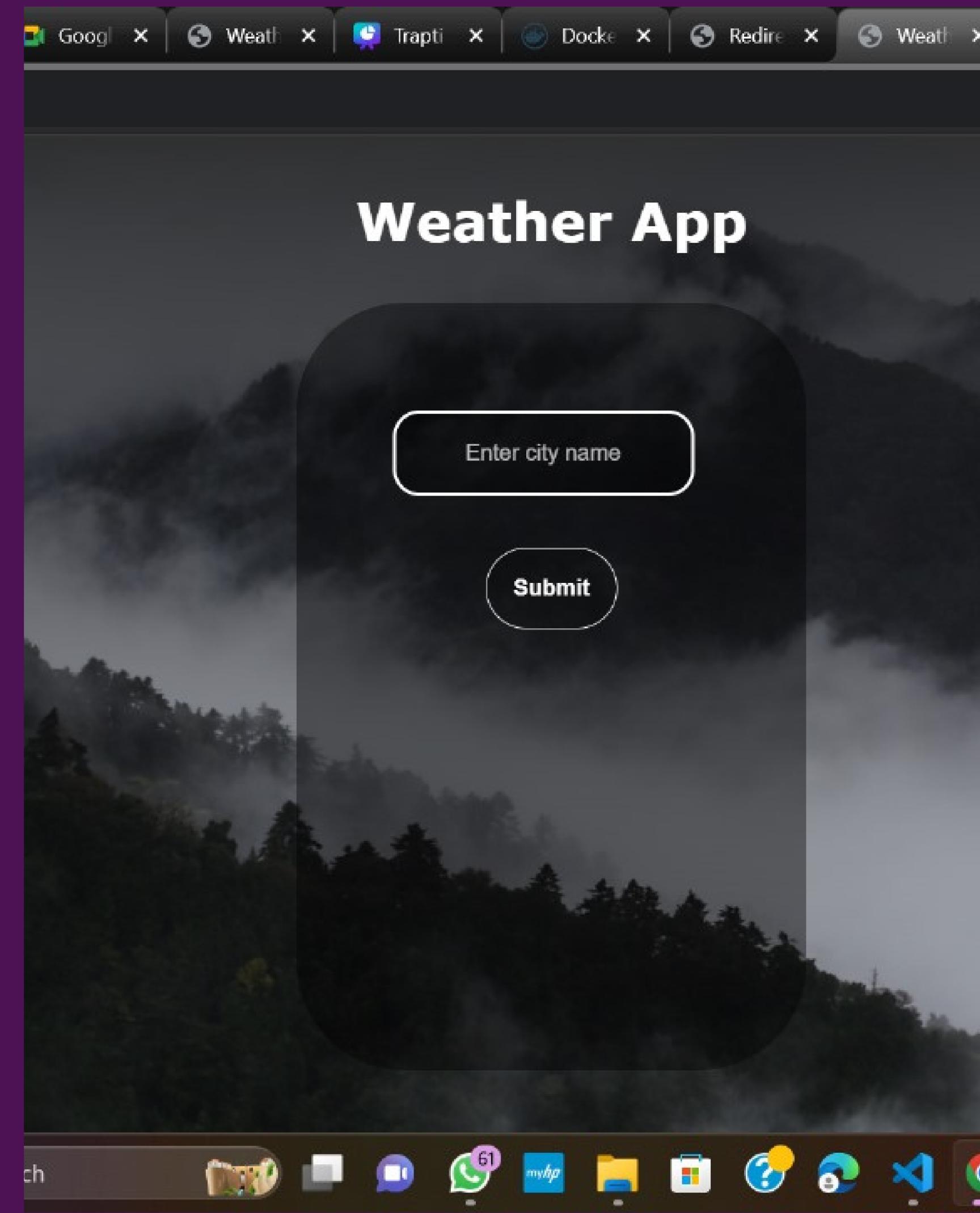
Containers					Give feedback
		Container CPU usage	Container memory		
		0.00% / 1000% (10 cores allocated)	0B / 7.43G		
		Search		Only show running containers	
		Name	Image	Status	CPU (%)
		magical_bar	traptigupta/repo1:i1	Running	0%
		4e0beff9aaae			

Step 6:

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

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

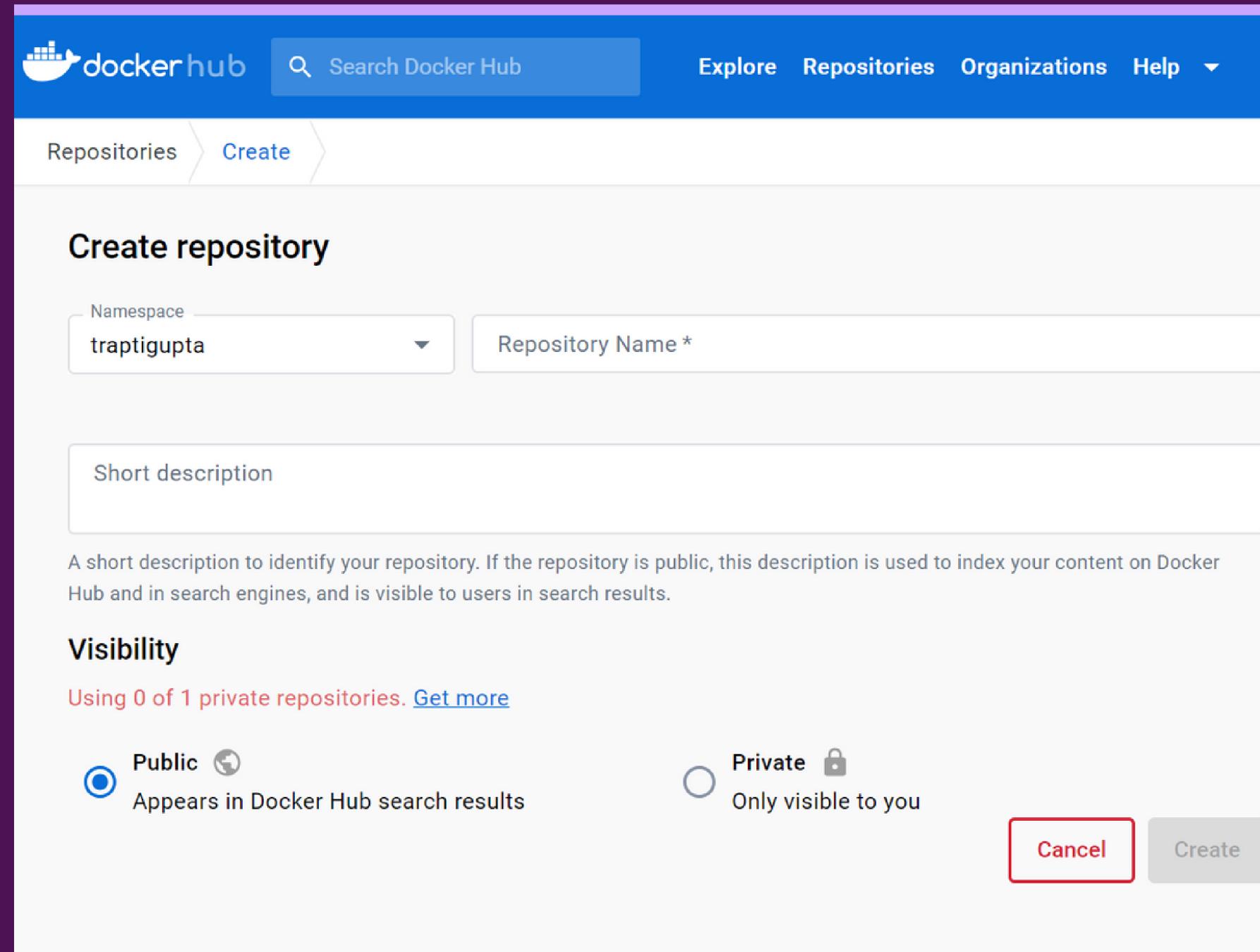
Then our project will open in our browser.



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
```

 **traptigupta / repo1**

Description

This repository does not have a description 

 Last pushed: 3 hours ago

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled
 img1		Image	7 hours ago

[See all](#) [Go to Advanced Images](#)

Repository overview 

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 instance by :

docker login -u traptigupta

aws Services Search

to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:
<https://hub.docker.com/>

For more examples and ideas, visit:
<https://docs.docker.com/get-started/>

```
root@ip-172-31-16-197:~# docker --version
Docker version 24.0.6, build ed223bc
root@ip-172-31-16-197:~# docker login -u traptigupta
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/con
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential
```

Login Succeeded
root@ip-172-31-16-197:~# docker pull traptigupta/repol:img1

i-047841bc154357b9f (dockerIns)

Public IPs: 54.91.89.166 Private IPs: 172.31.16.197

CloudShell Feedback ©

Type here to search

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

```
root@ip-172-31-16-197:~# docker pull traptigupta/repo1:img1
img1: Pulling from traptigupta/repo1
96526aa774ef: Pull complete
2004135e416: Pull complete
bf1cf5026c4: Pull complete
8966af6931d: Pull complete
3ee70732c61: Pull complete
e2fd992447a: Pull complete
6cbc9ea6abf: Pull complete
7f8bcf34db7: Pull complete
8328ea38618: Pull complete
Digest: sha256:1b4a7b806d9e95805fd02b6e1591df7d4ff95e55058a256a9ae93ba7
Status: Downloaded newer image for traptigupta/repo1:img1
ocker.io/traptigupta/repo1:img1
root@ip-172-31-16-197:~# docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
traptigupta/repo1   img1    3fd5c9621404  13 hours ago  58.6MB
hello-world         latest   9c7a54a9a43c  5 months ago  13.3kB
root@ip-172-31-16-197:~# docker run -d -p 80:80 traptigupta/repo1:img1
ed438eebe82e52b5d764cb7feee0e3230f42a575b112e3d5cf8b6a733bd3406f
root@ip-172-31-16-197:~#
```

i-047841bc154357b9f (dockerIns)

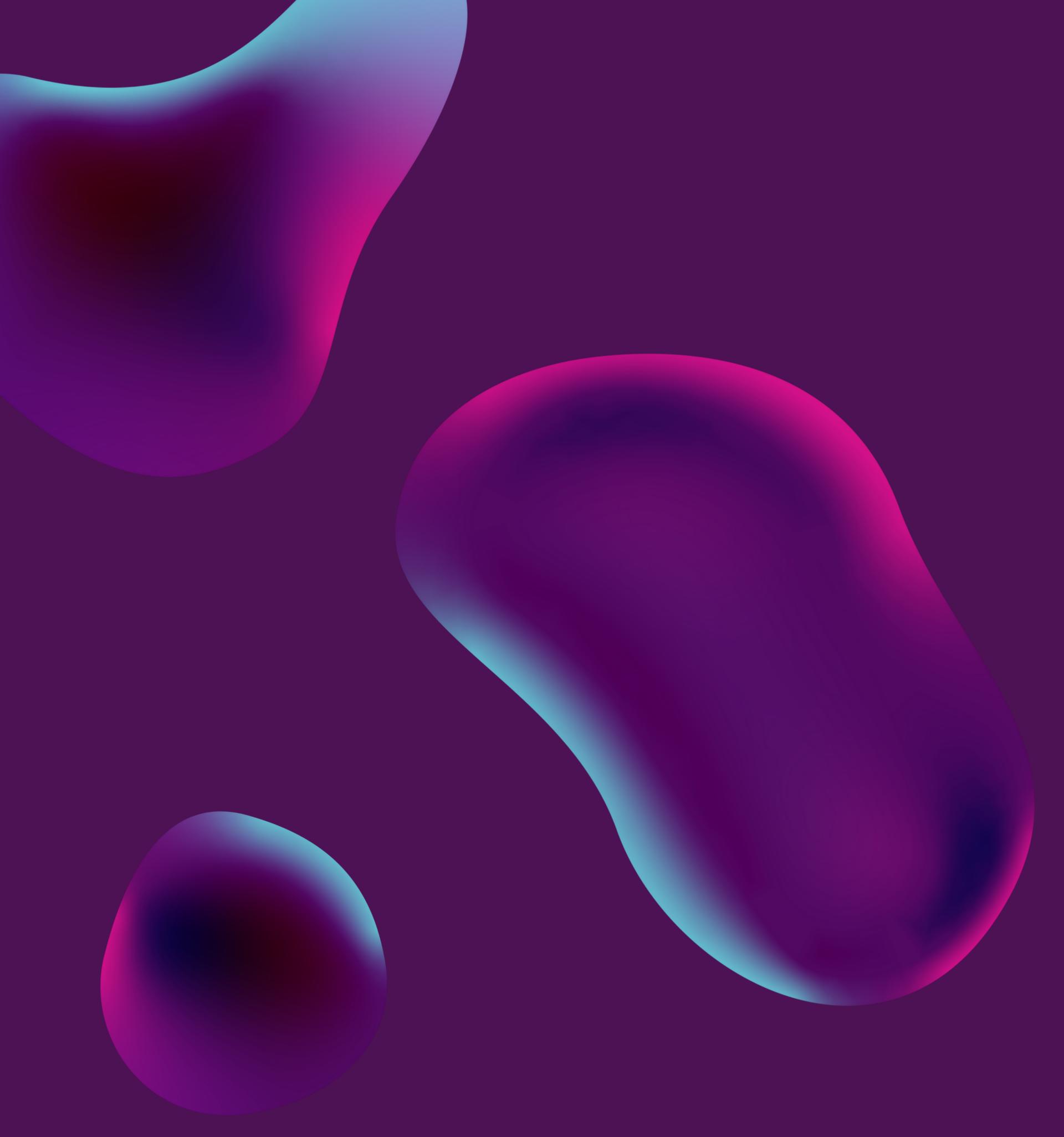
PublicIPs: 54.91.89.166 PrivateIPs: 172.31.16.197

[CloudShell](#) [Feedback](#)



Type here to search





Now project
will be
available on
public ip of our
instance

Public IP : 54.91.89.166

thank
you!