AISSMS
COLLEGE OF ENGINEERING
ज्ञानम् सकलजनहिताय

Approved by AICTE, New Delhi, Recognized by Government of Maharashtra
Affiliated to Savitribai Phule Pune University and recognized 2(f) and 12(B) by UGC
(Id.No. PU/PN/Engg./093 (1992)
Accredited by NAAC with "A+" Grade | NBA - 7 UG Programmes

Department of Computer Engineering

"CC Mini-project Stepwise explanation"

Submitted By

Name of the Student: Piyusha Rajendra Supe

Roll No: 23CO315

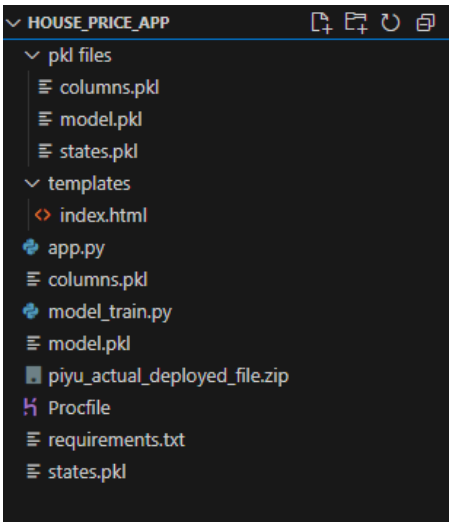
Under the Guidance of

Prof. V. M. Kanavde

Title: Deploying a machine learning application of House price prediction on Cloud using the AWS Elastic Beanstalk

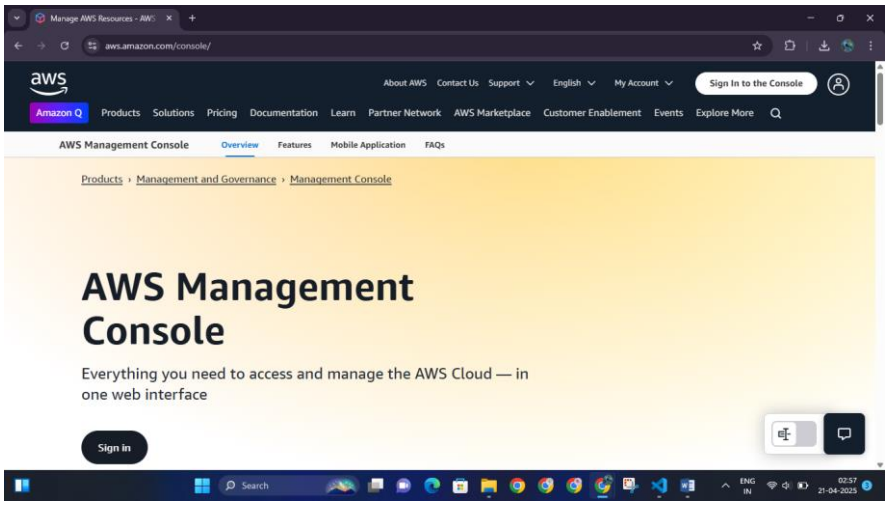
The steps for deploying the application are given as follows:

The directory structure is as follows:



Step 1: Sign Up and Log in to AWS

1. **Go to AWS**:
   Open your browser and go to AWS.
2. **Create an account**:
   - Click **Sign Up** and follow the prompts to create a new account. You'll need to provide billing information (AWS offers a **Free Tier** for new users).
   - **Sign in** once your account is set up.



Step 2: Prepare Your Flask App

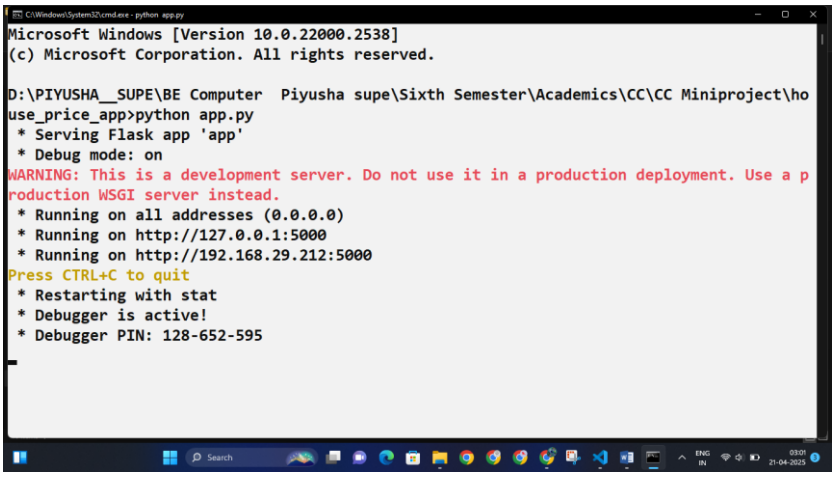
1. **Get your Flask app ready**: (**Testing the application locally**)
    - You should already have your Flask app ready (the one with the machine learning model). Make sure your app is working locally on your computer first.
    - You need to prepare two files for AWS:
        - requirements.txt: This file tells AWS what libraries your app needs. To create it, run the following in your project folder:

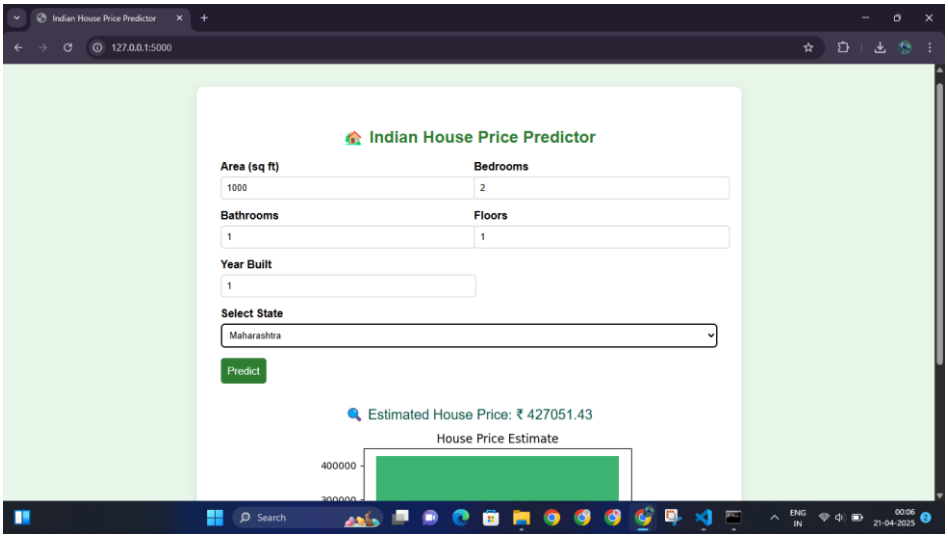
            pip freeze > requirements.txt

        - Procfile: This file tells Elastic Beanstalk how to run your app. Create a new file called Procfile (no extension) in your project folder, and add this line:

            web: python app.py

            Make sure your app.py file is the main script that runs your Flask app.



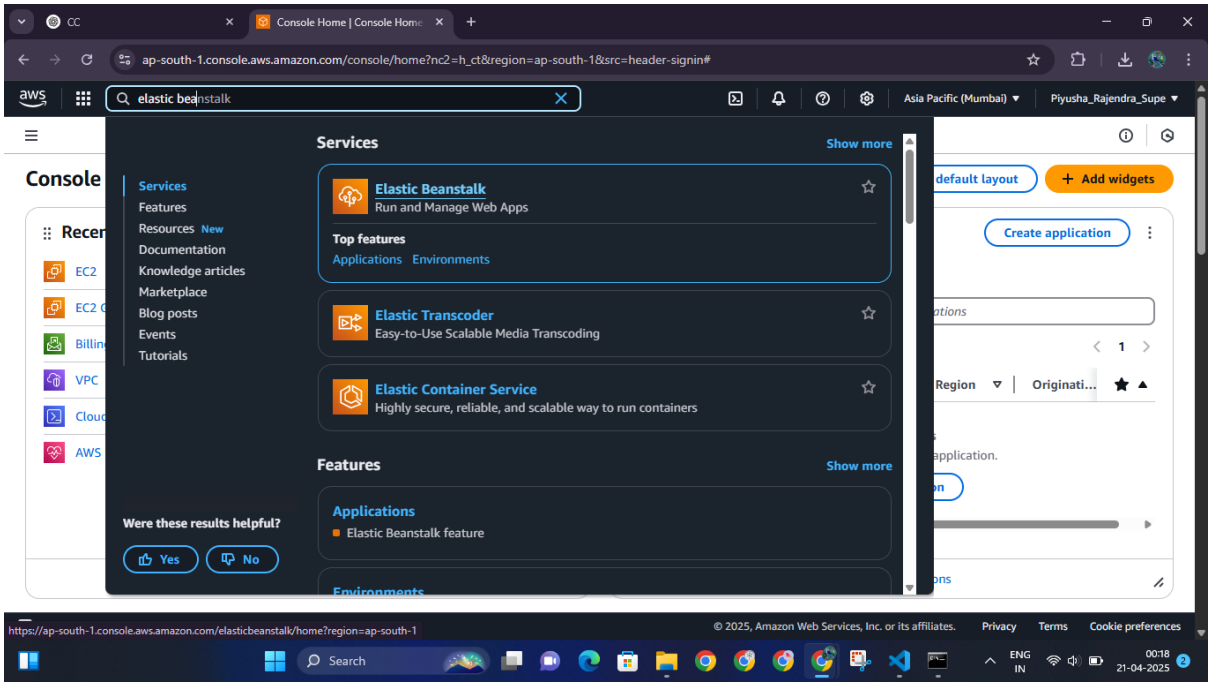


2. **Update app.py**:
   Make sure your Flask app listens on port 8080, which is the default for AWS:

```
if __name__ == "__main__":
        app.run(debug=True, host='0.0.0.0', port=8080)
```

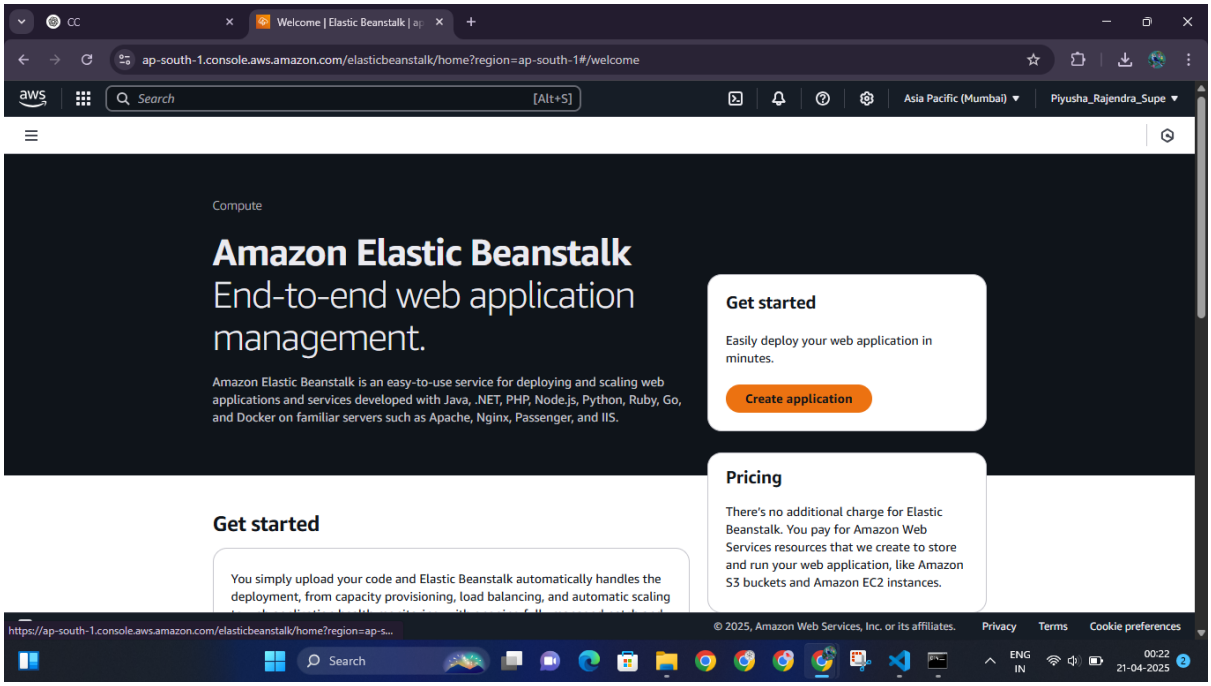
Step 3: Access AWS Console

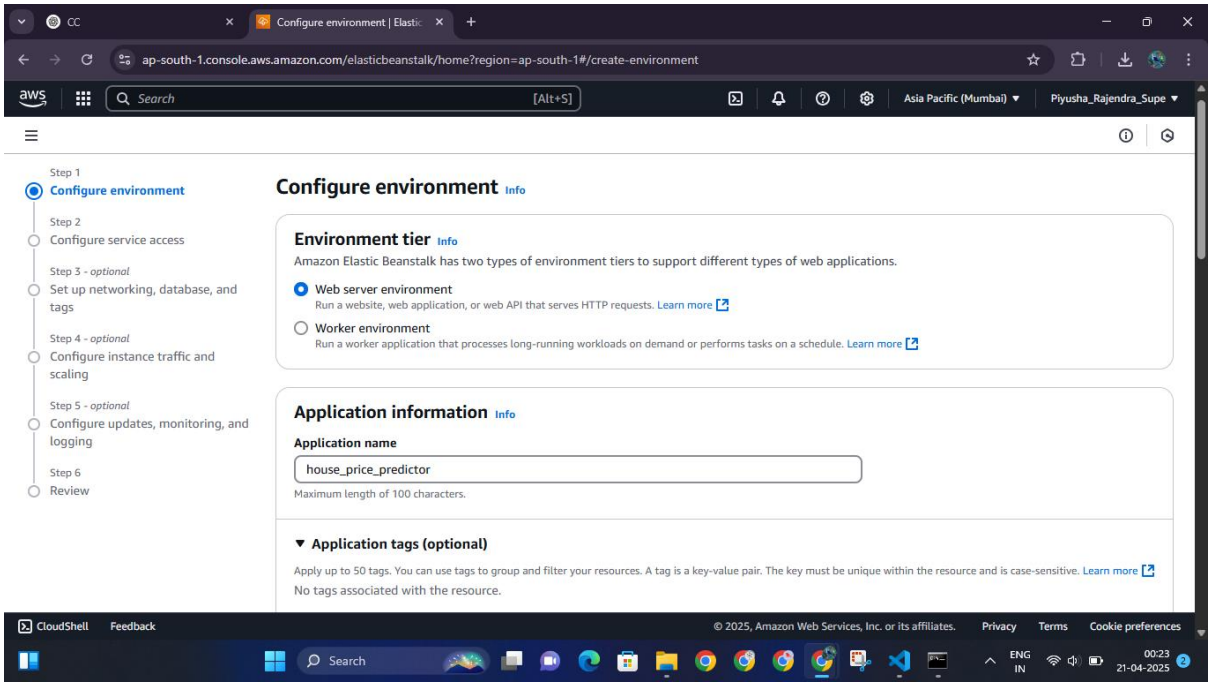
1. **Go to AWS Management Console**:
   - Open your browser and go to AWS Management Console.
   - **Sign in** with the AWS account you created.

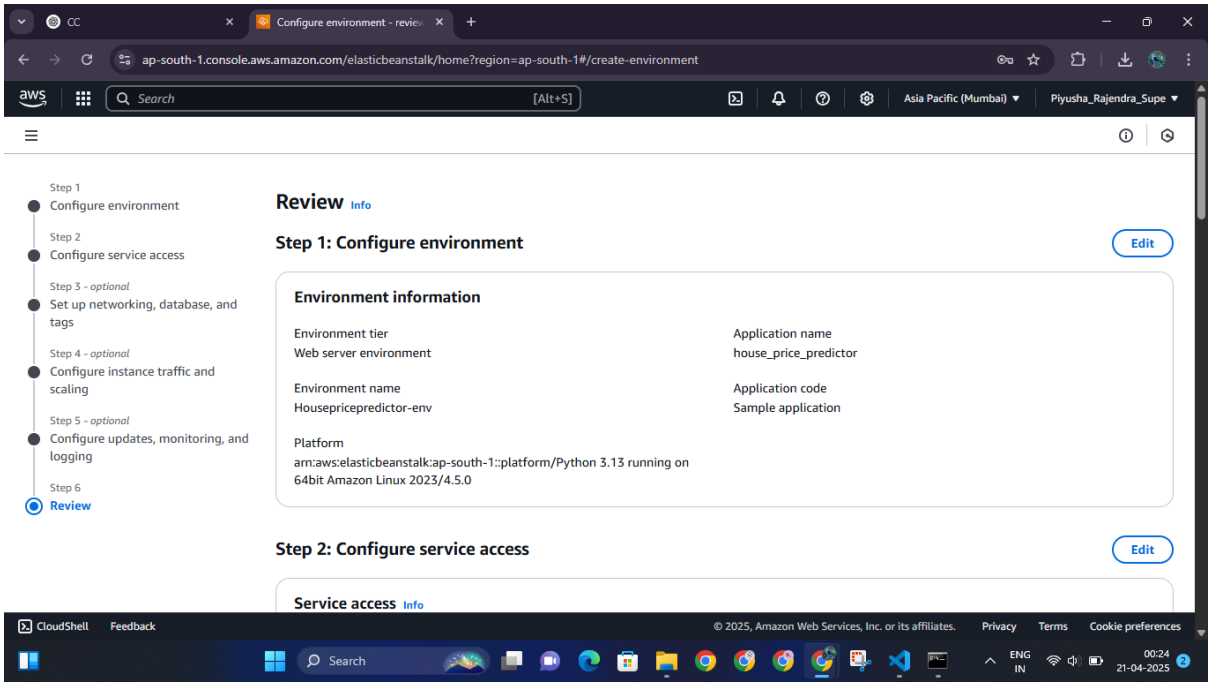


Step 4: Create an Elastic Beanstalk Application

1. **Search for Elastic Beanstalk**:
   - In the **Search bar** at the top, type **Elastic Beanstalk** and click on it.
2. **Create a new application**:
   - In the Elastic Beanstalk console, click **Create Application**.
   - **Give your app a name**: You can name it something like house-price-predictor.
   - **Platform**: Choose **Python**.
   - **Environment Tier**: Select **Web Server Environment**.
   - Click **Create Application**.

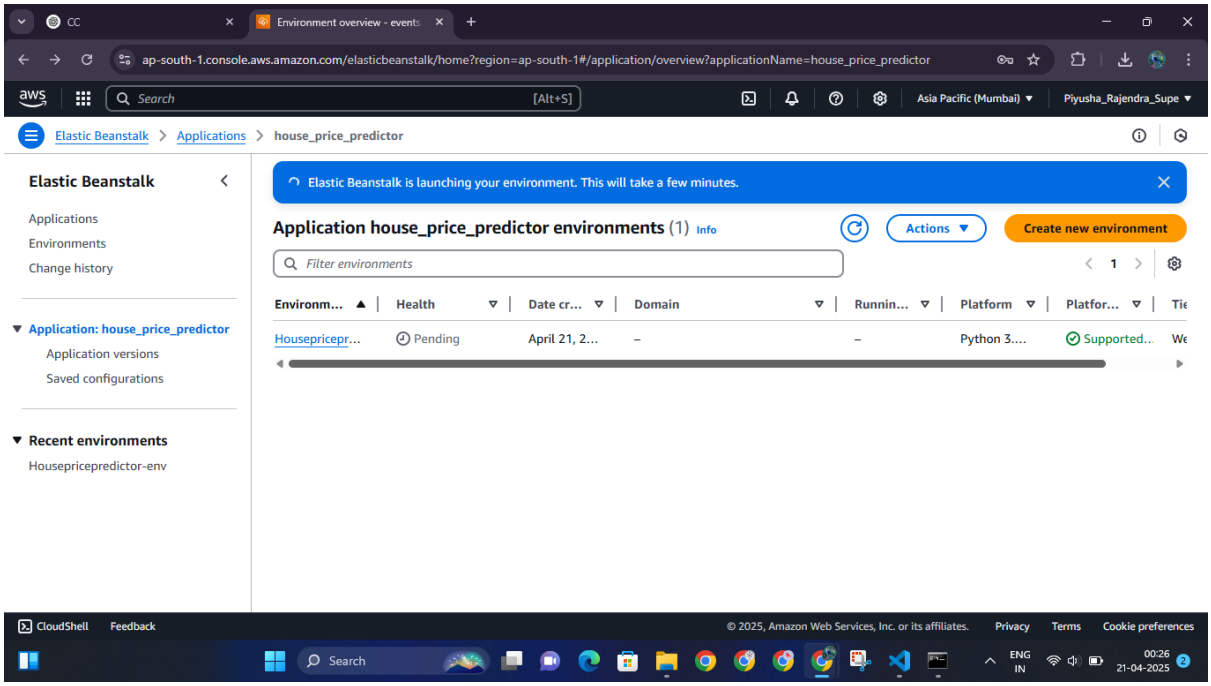

Welcome | Elastic Beanstalk
ap-south-1.console.aws.amazon.com/elasticbeanstalk/home?region=ap-south-1#/welcome
Search
[Alt+S]
Asia Pacific (Mumbai)
Piyusha_Rajendra_Supe
Compute
Amazon Elastic Beanstalk
End-to-end web application management.
Amazon Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS.
Get started
Easily deploy your web application in minutes.
Create application
Pricing
There's no additional charge for Elastic Beanstalk. You pay for Amazon Web Services resources that we create to store and run your web application, like Amazon S3 buckets and Amazon EC2 instances.
Get started
You simply upload your code and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, and automatic scaling
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
00:22
21-04-2025

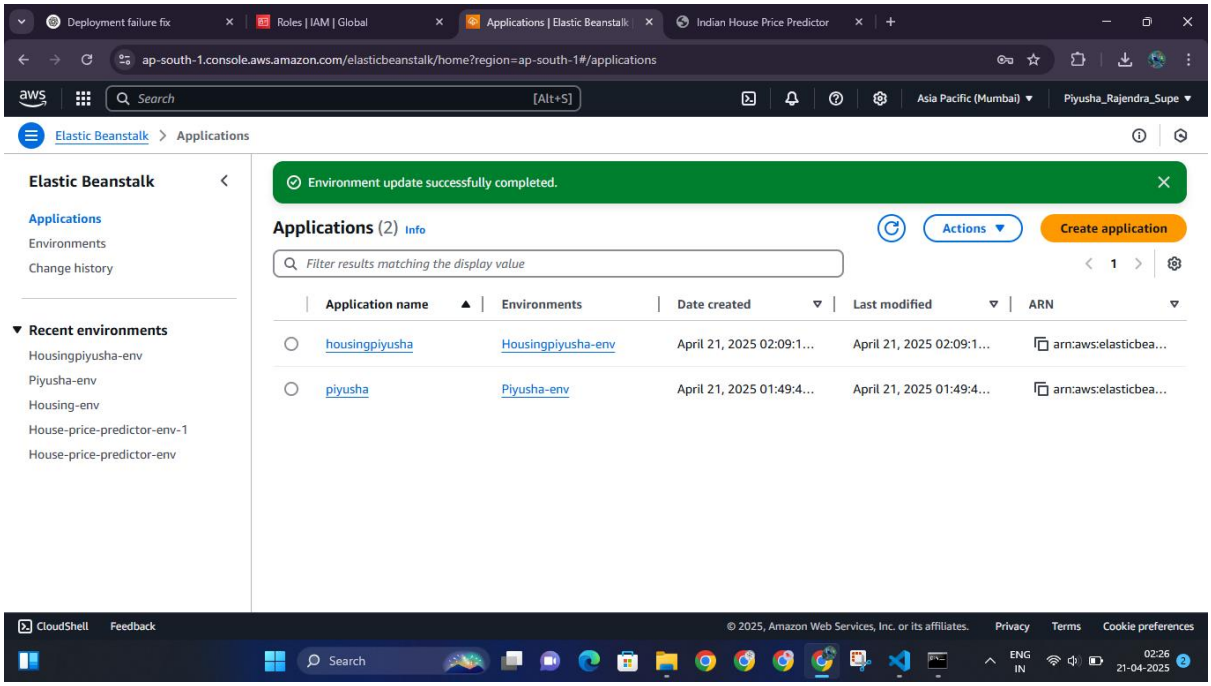

Configure environment | Elastic
ap-south-1.console.aws.amazon.com/elasticbeanstalk/home?region=ap-south-1#/create-environment
Search
[Alt+S]
Asia Pacific (Mumbai)
Piyusha_Rajendra_Supe
Step 1
Configure environment
Step 2
Configure service access
Step 3 - optional
Set up networking, database, and tags
Step 4 - optional
Configure instance traffic and scaling
Step 5 - optional
Configure updates, monitoring, and logging
Step 6
Review
Configure environment Info
Environment tier Info
Amazon Elastic Beanstalk has two types of environment tiers to support different types of web applications.
Web server environment
Run a website, web application, or web API that serves HTTP requests. Learn more
Worker environment
Run a worker application that processes long-running workloads on demand or performs tasks on a schedule. Learn more
Application information Info
Application name
house_price_predictor
Maximum length of 100 characters.
Application tags (optional)
Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key must be unique within the resource and is case-sensitive. Learn more
No tags associated with the resource.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
00:23
21-04-2025

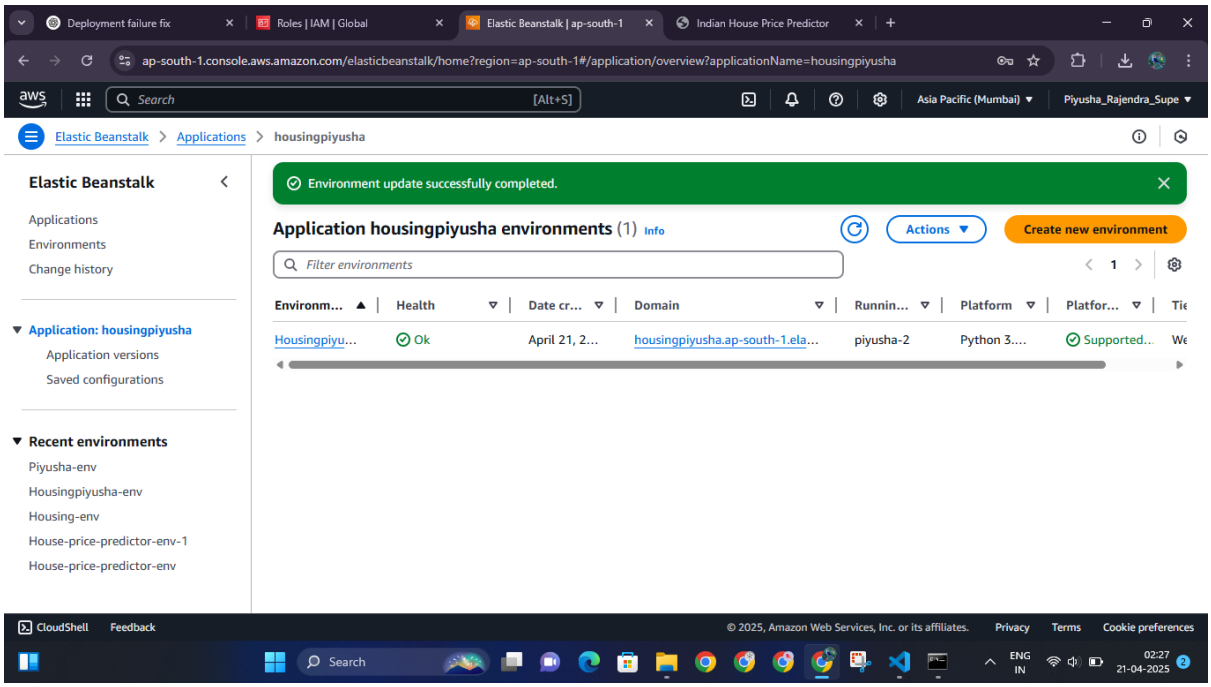


Step 5: Create an Elastic Beanstalk Environment

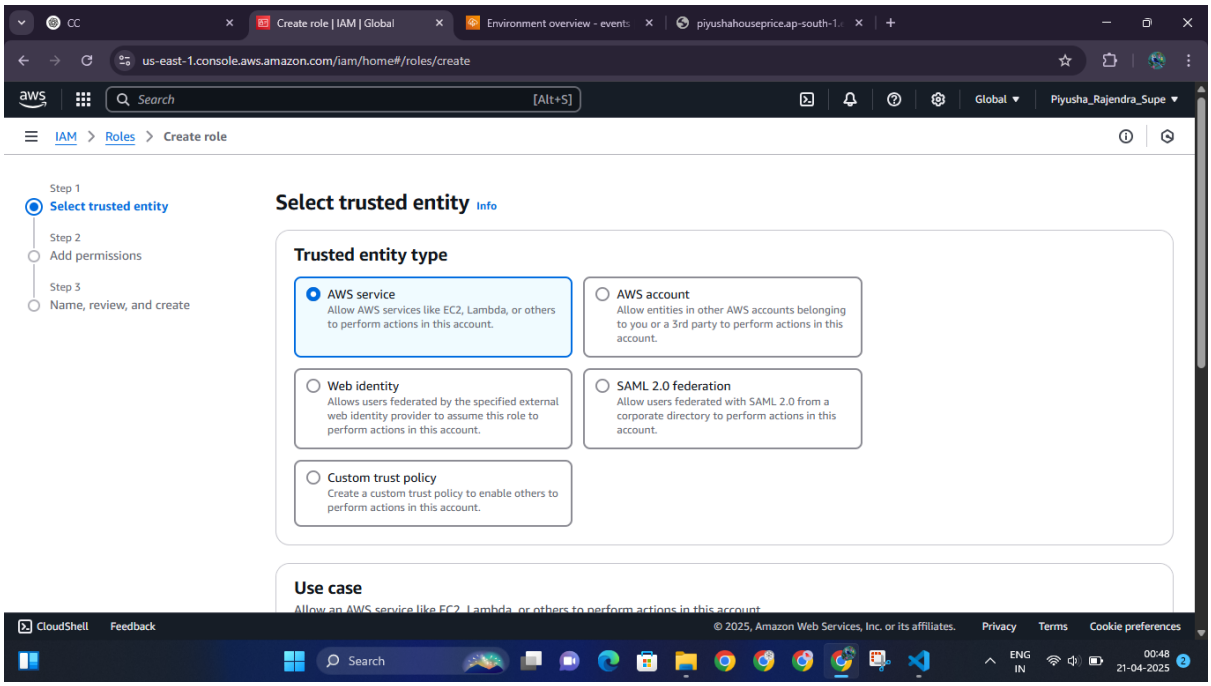
1. **Click on your newly created application**:
   - After the app is created, click on it from the list to go to the app dashboard.
2. **Create a new environment**:
   - Click on the **Create Environment** button.
   - **Choose environment type**: Select **Web Server Environment**.
   - **DNS name**: Enter a name for your app's URL (e.g., house-price-predictor).
   - **Platform**: Select **Python**.
   - Click **Next**.

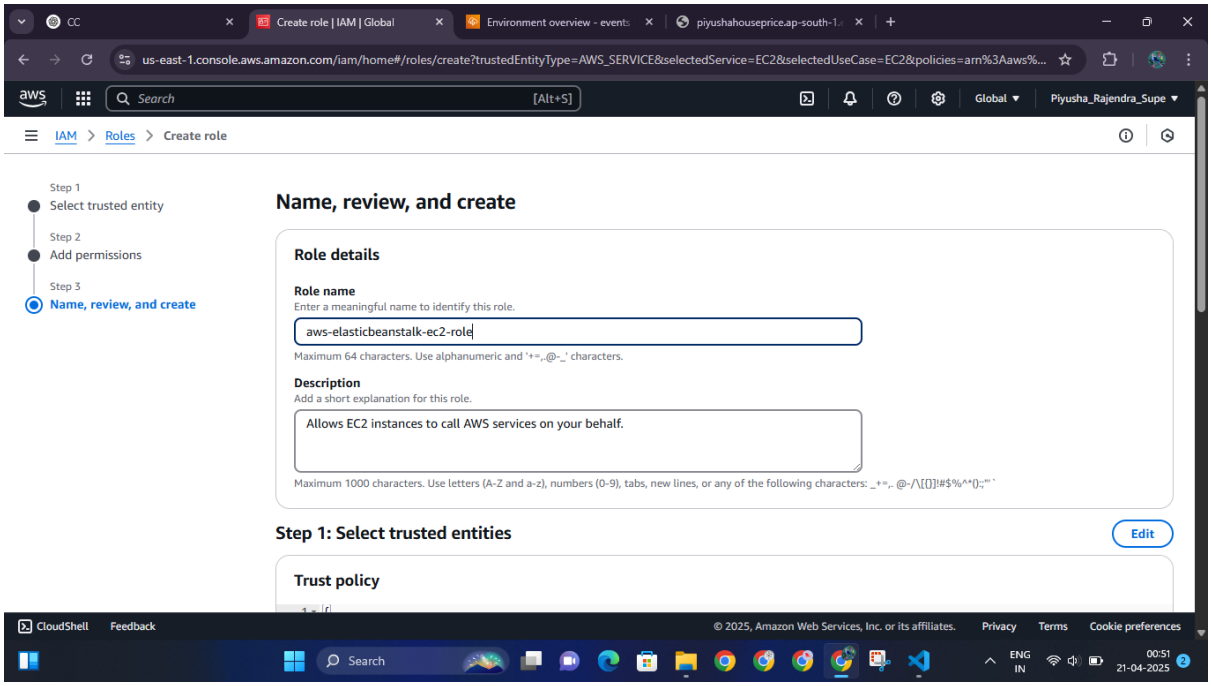


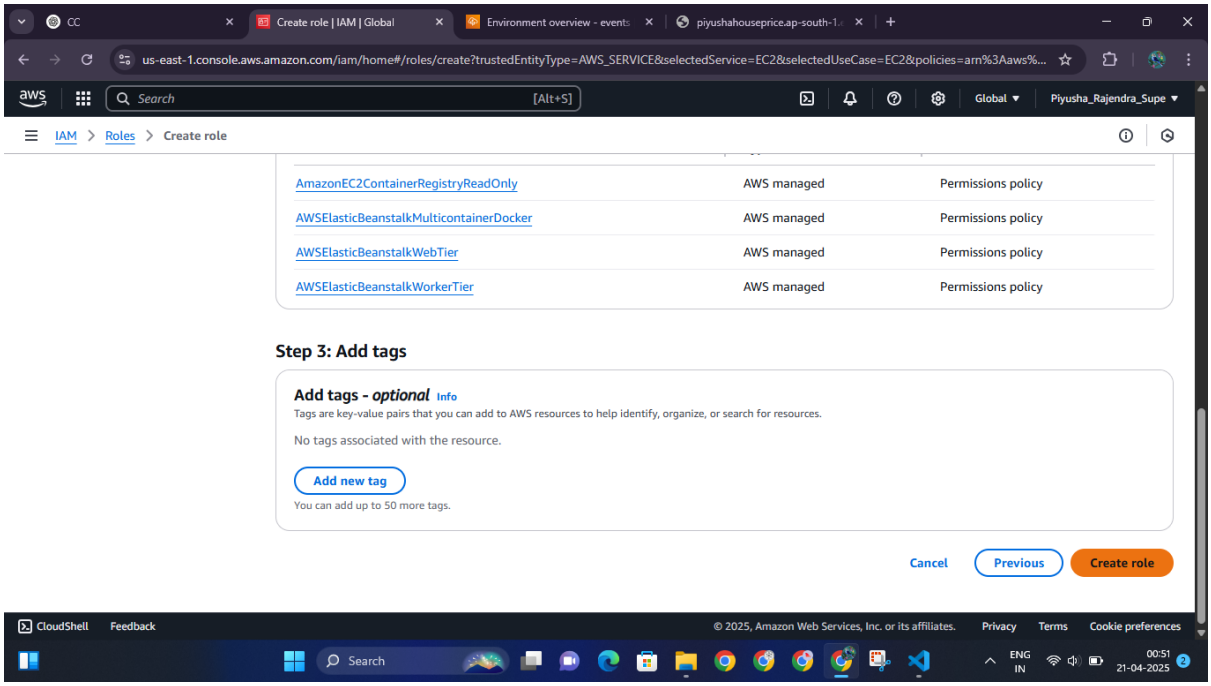

Deployment failure fix
Roles | IAM | Global
Applications | Elastic Beanstalk
Indian House Price Predictor
ap-south-1.console.aws.amazon.com/elasticbeanstalk/home?region=ap-south-1#/applications
Search
[Alt+S]
Asia Pacific (Mumbai)
Piyusha_Rajendra_Supe
Elastic Beanstalk > Applications
Elastic Beanstalk
Applications
Environments
Change history
Recent environments
Housingpiyusha-env
Piyusha-env
Housing-env
House-price-predictor-env-1
House-price-predictor-env
Environment update successfully completed.
Applications (2) Info
Actions
Create application
Filter results matching the display value
Application name
Environments
Date created
Last modified
ARN
housingpiyusha
Housingpiyusha-env
April 21, 2025 02:09:1...
April 21, 2025 02:09:1...
arn:aws:elasticbea...
piyusha
Piyusha-env
April 21, 2025 01:49:4...
April 21, 2025 01:49:4...
arn:aws:elasticbea...
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
ENG
IN
02:26
21-04-2025

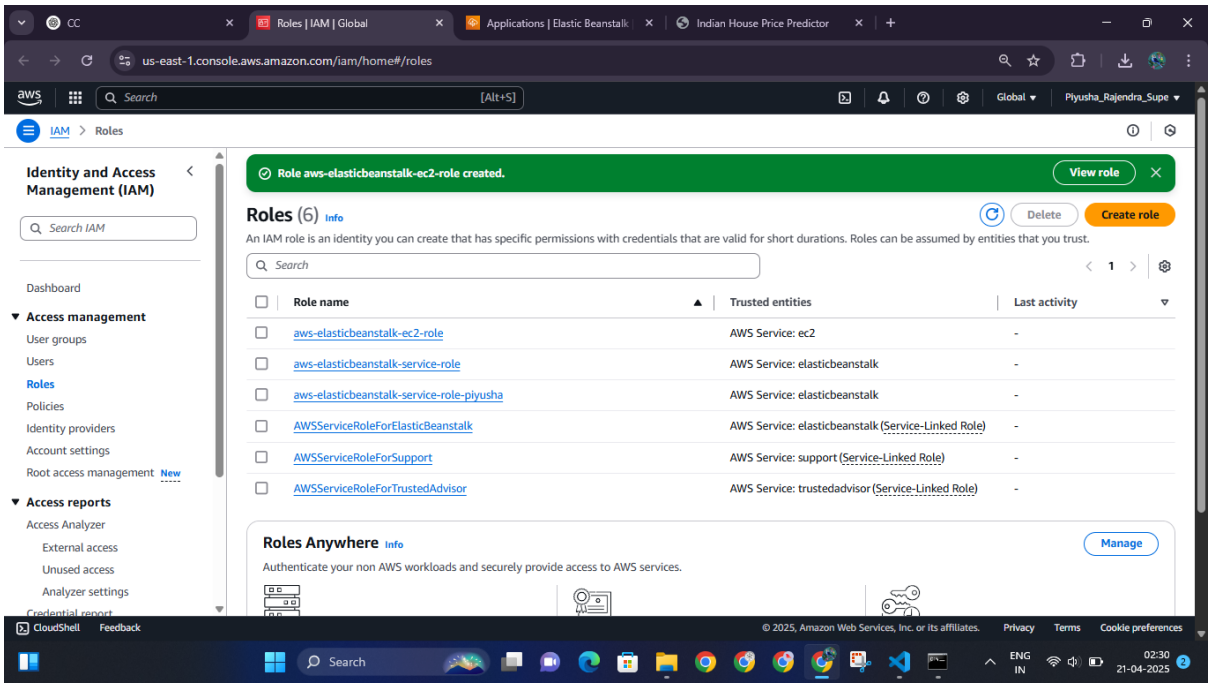

Deployment failure fix
Roles | IAM | Global
Elastic Beanstalk | ap-south-1
Indian House Price Predictor
ap-south-1.console.aws.amazon.com/elasticbeanstalk/home?region=ap-south-1#/application/overview?applicationName=housingpiyusha
Search
[Alt+S]
Asia Pacific (Mumbai)
Piyusha_Rajendra_Supe
Elastic Beanstalk > Applications > housingpiyusha
Elastic Beanstalk
Applications
Environments
Change history
Application: housingpiyusha
Application versions
Saved configurations
Recent environments
Piyusha-env
Housingpiyusha-env
Housing-env
House-price-predictor-env-1
House-price-predictor-env
Environment update successfully completed.
Application housingpiyusha environments (1) Info
Actions
Create new environment
Filter environments
Environm...
Health
Date cr...
Domain
Runnin...
Platform
Platfor...
Housingpiyu...
Ok
April 21, 2...
housingpiyusha.ap-south-1.ela...
piyusha-2
Python 3....
Supported...
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
ENG
IN
02:27
21-04-2025

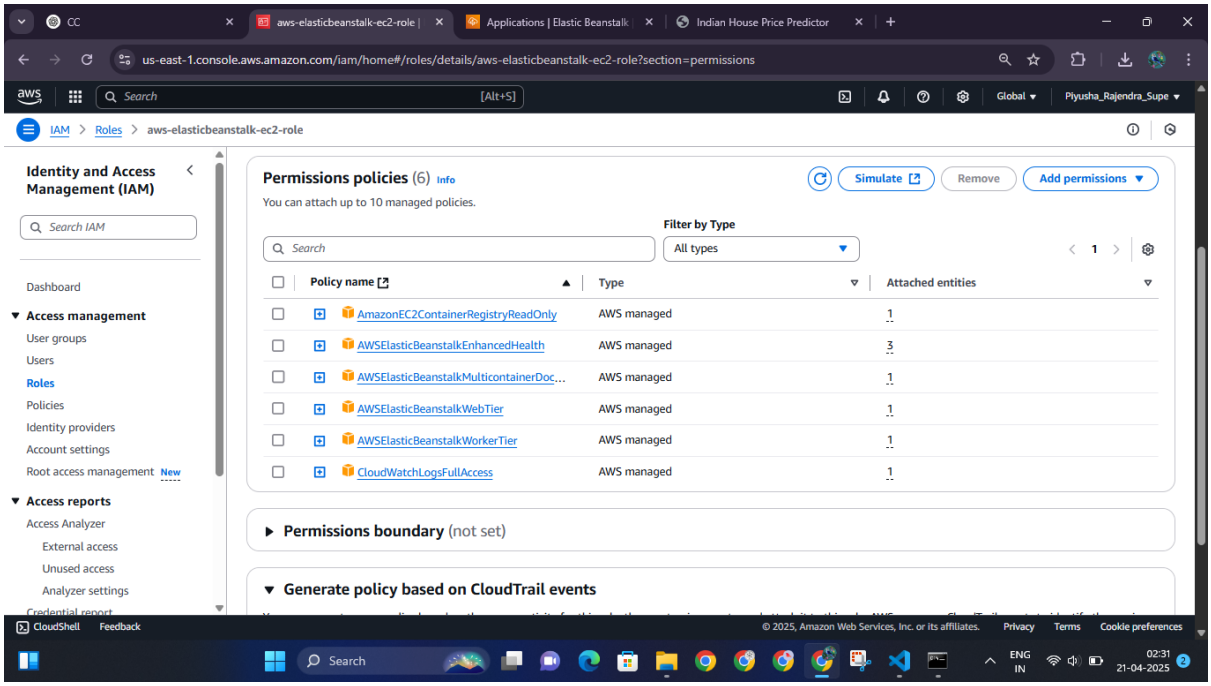
Make sure the following roles are created and the permissions are set to them:





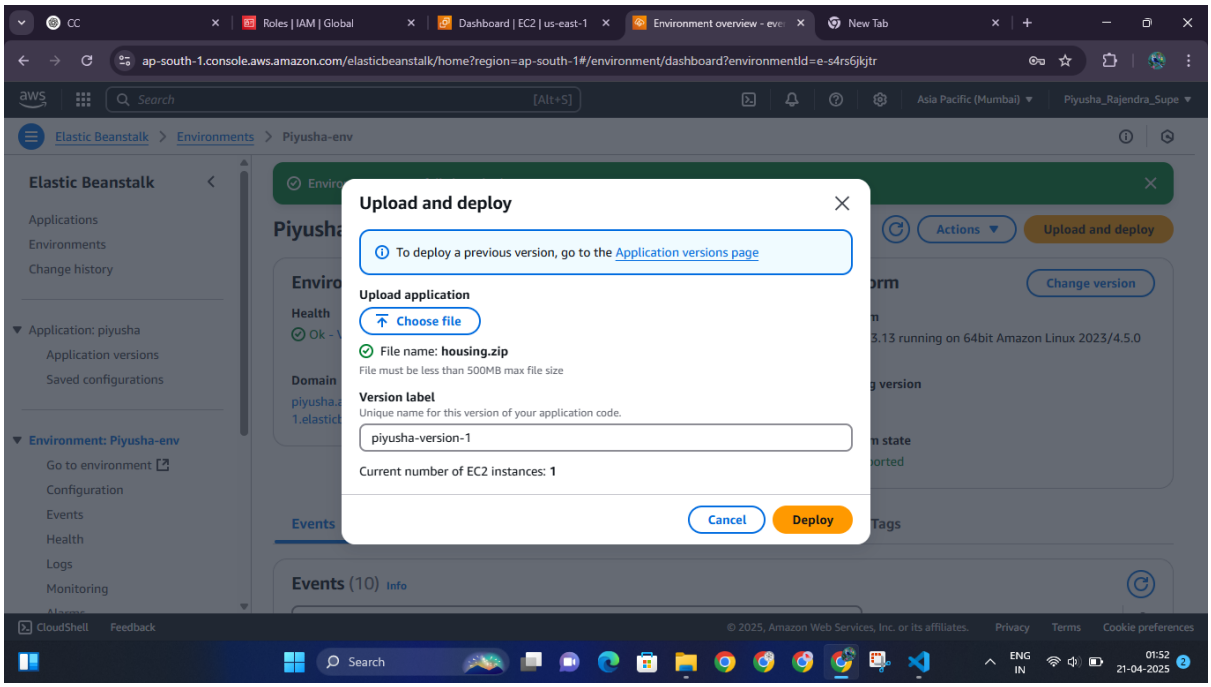

Create role | IAM | Global
us-east-1.console.aws.amazon.com/iam/home#/roles/create?trustedEntityType=AWS_SERVICE&selectedService=EC2&selectedUseCase=EC2&policies=arn%3Aaws%...
IAM > Roles > Create role
AmazonEC2ContainerRegistryReadOnly AWS managed Permissions policy
AWSElasticBeanstalkMulticontainerDocker AWS managed Permissions policy
AWSElasticBeanstalkWebTier AWS managed Permissions policy
AWSElasticBeanstalkWorkerTier AWS managed Permissions policy
Step 3: Add tags
Add tags - optional Info
Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.
No tags associated with the resource.
Add new tag
You can add up to 50 more tags.
Cancel
Previous
Create role
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Roles | IAM | Global
Applications | Elastic Beanstalk
Indian House Price Predictor
us-east-1.console.aws.amazon.com/iam/home#/roles
IAM > Roles
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Access Analyzer
External access
Unused access
Analyzer settings
Role aws-elasticbeanstalk-ec2-role created.
View role
Roles (6) Info
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.
Delete
Create role
Role name
Trusted entities
Last activity
aws-elasticbeanstalk-ec2-role AWS Service: ec2 -
aws-elasticbeanstalk-service-role AWS Service: elasticbeanstalk -
aws-elasticbeanstalk-service-role-piyusha AWS Service: elasticbeanstalk -
AWSServiceRoleForElasticBeanstalk AWS Service: elasticbeanstalk (Service-Linked Role) -
AWSServiceRoleForSupport AWS Service: support (Service-Linked Role) -
AWSServiceRoleForTrustedAdvisor AWS Service: trustedadvisor (Service-Linked Role) -
Roles Anywhere Info
Authenticate your non AWS workloads and securely provide access to AWS services.
Manage
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



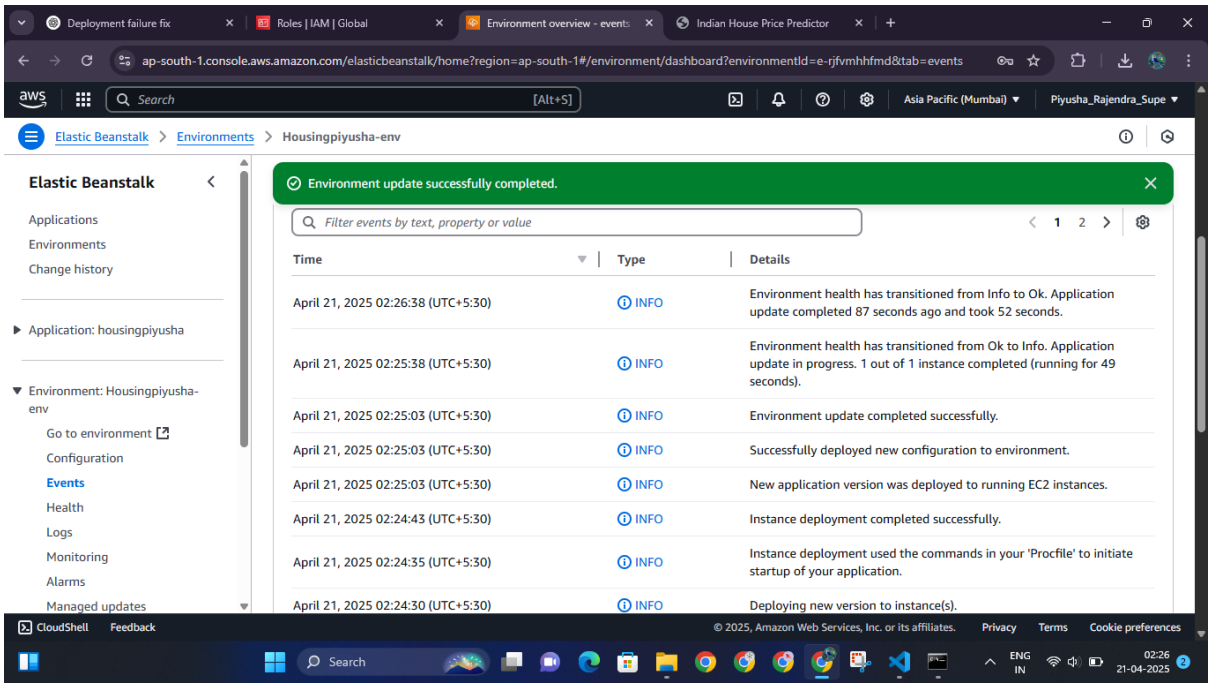
Step 6: Upload Your App

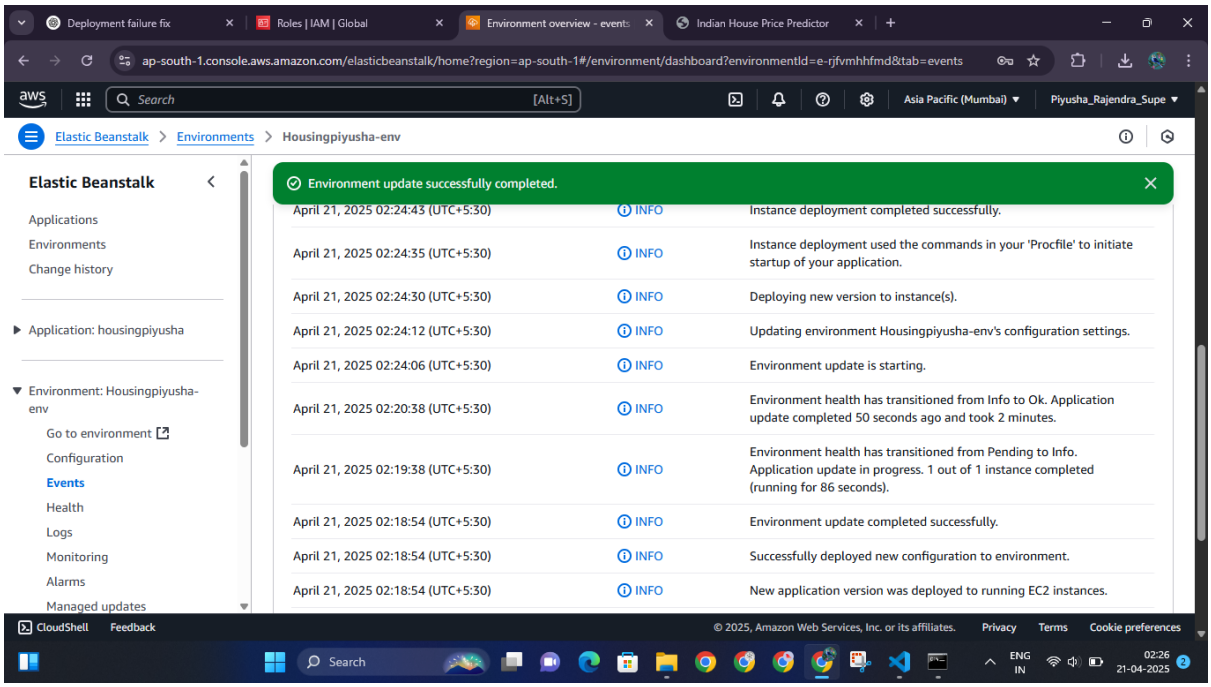
1. **Prepare your app ZIP file**:
   - Go to your project folder and **ZIP** all the files in it, including app.py, requirements.txt, and Procfile.
2. **Upload the ZIP file**:
   - Under **Application Version**, click **Upload your code**.
   - Click the **Choose file** button and select the ZIP file you just created.
   - After uploading, click **Next**.

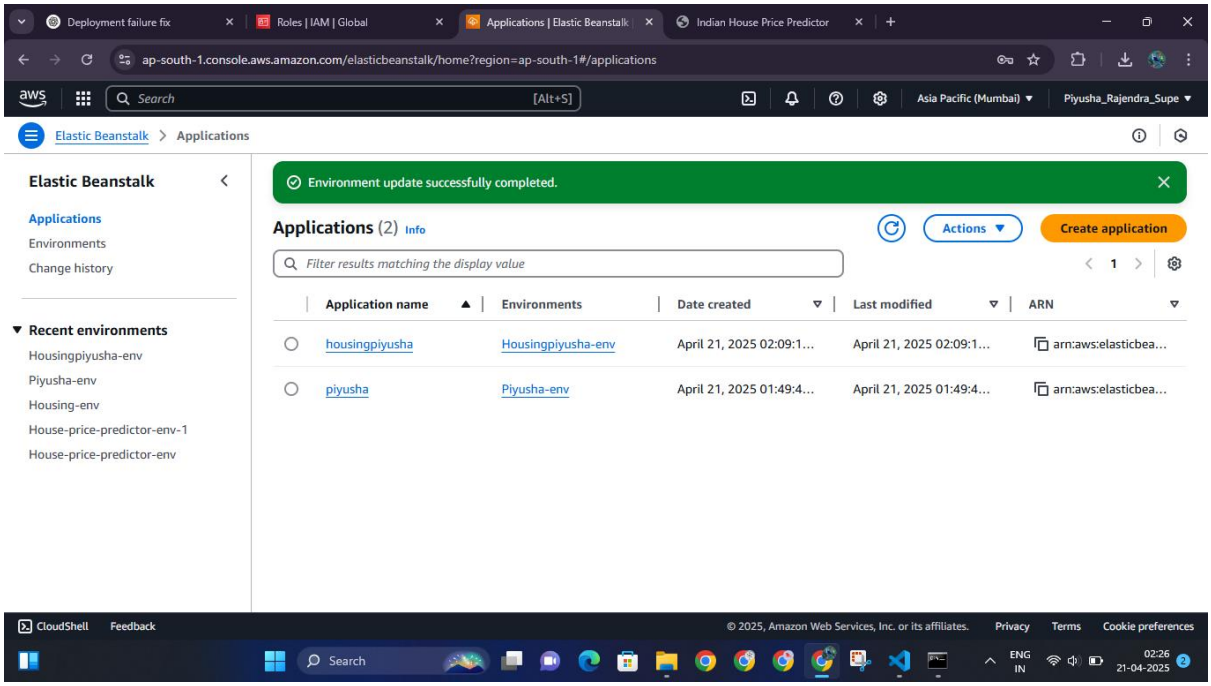


Step 7: Review and Launch the Application

1. **Review your environment**:
   - AWS will show a review of your environment settings (like DNS name, platform, etc.).
   - Check everything, then click **Launch**.
2. **Wait for Elastic Beanstalk to set up**:
   - AWS will now take a few minutes to provision your environment and deploy your app.
   - You will see a **status page** indicating when the environment is ready.

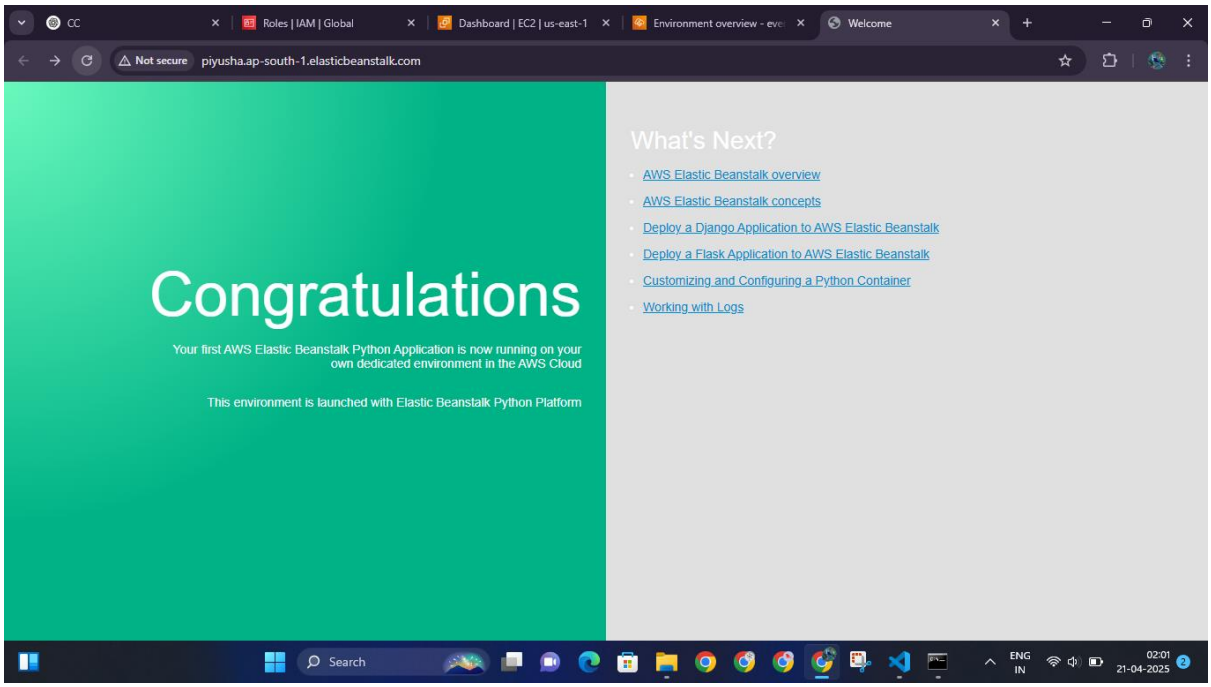


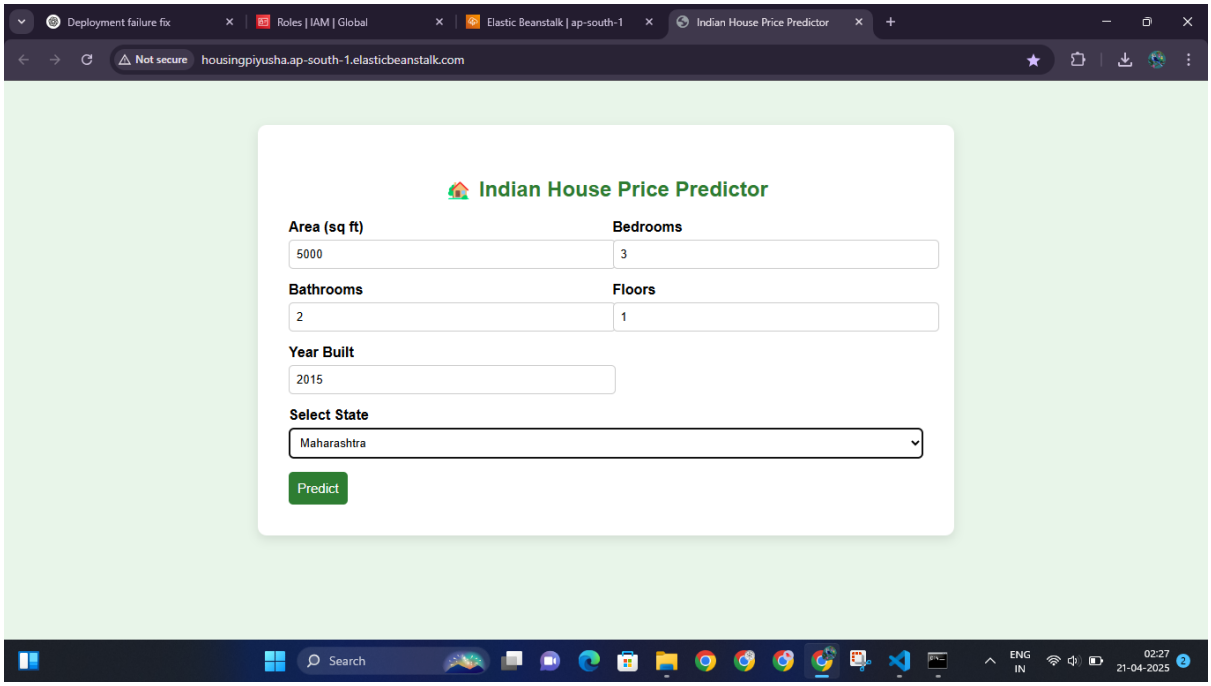


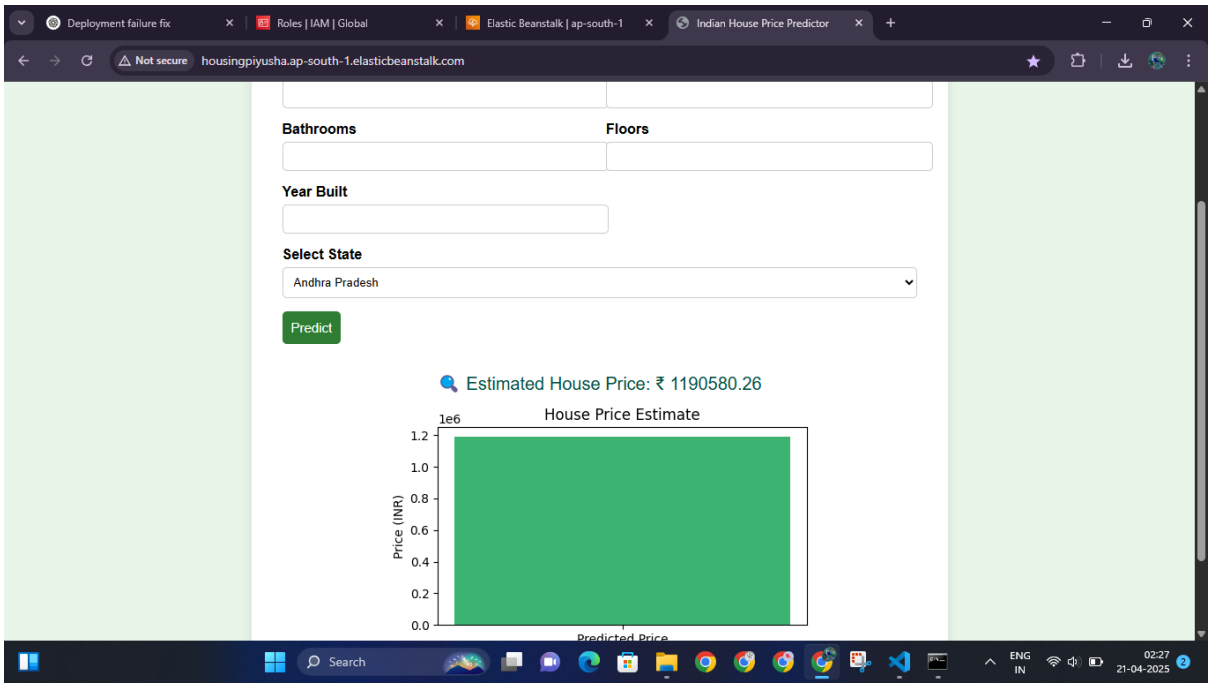


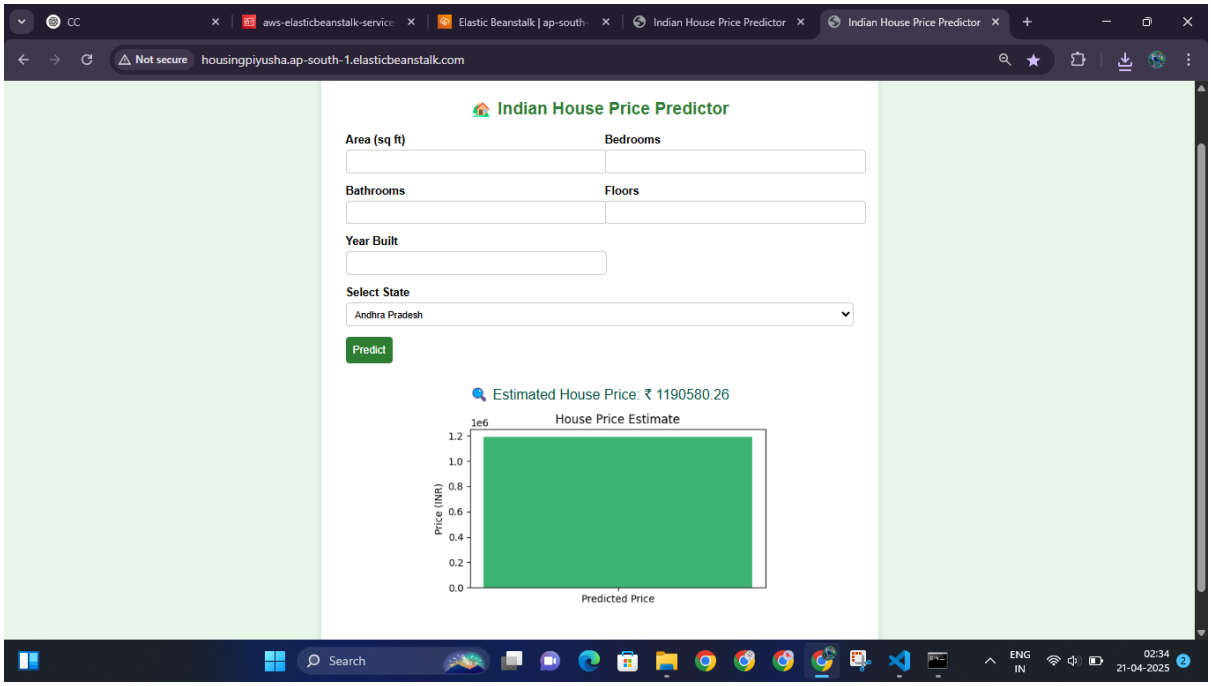
Step 8: Access Your Flask App

1. **Get your app URL**:
   - Once the environment is created, you'll see a **URL** for your app in the Elastic Beanstalk dashboard (e.g., http://house-price-predictor.elasticbeanstalk.com).
   - Click on this link, and your Flask app will open in a new tab!



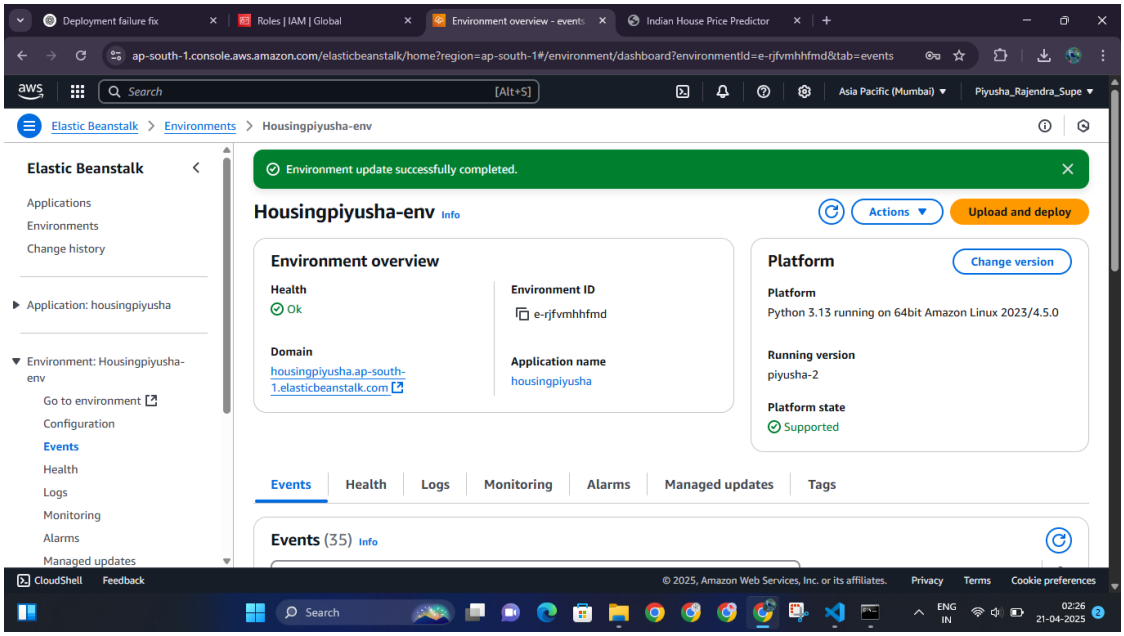

Deployment failure fix
Roles | IAM | Global
Elastic Beanstalk | ap-south-1
Indian House Price Predictor
Not secure
housingpiyusha.ap-south-1.elasticbeanstalk.com
Indian House Price Predictor
Area (sq ft)
5000
Bedrooms
3
Bathrooms
2
Floors
1
Year Built
2015
Select State
Maharashtra
Predict
Search
ENG
IN
02:27
21-04-2025


Deployment failure fix
Roles | IAM | Global
Elastic Beanstalk | ap-south-1
Indian House Price Predictor
Not secure
housingpiyusha.ap-south-1.elasticbeanstalk.com
Bathrooms
Floors
Year Built
Select State
Andhra Pradesh
Predict
Estimated House Price: ₹ 1190580.26
House Price Estimate
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Price (INR)
Predicted Price
Search
ENG
IN
02:27
21-04-2025



Step 9: Monitor Your App

1. **Check the health of your app**:
   - Go back to the **Elastic Beanstalk** dashboard.
   - You can see whether your app is healthy (Green), or if there are any issues (Red).
2. **View logs**:
   - If your app is not working, you can check the **logs** from the Elastic Beanstalk console.
   - Click **Logs** > **Request Logs** to get detailed logs that can help troubleshoot errors.



Step 10: Clean Up (Optional)

Once you're done testing or if you want to stop the app, you can **terminate the environment**:

1. **Terminate the environment**:
    - In the Elastic Beanstalk console, go to your environment.
    - Click **Actions** > **Terminate Environment**.
    - Confirm the termination to stop any resources and prevent charges.

Recap:

1. **Sign Up** for AWS and log in.
2. **Prepare your Flask app** by creating requirements.txt and Procfile.
3. **Go to AWS Console**, search for **Elastic Beanstalk**, and **create a new application**.
4. **Create a new environment** for your app.
5. **Upload your ZIP file** containing your Flask app.
6. **Launch the app** and **wait** for it to be ready.
7. **Get the URL** to access your live Flask app!
8. **Monitor your app** for health and logs.

The source code is as follows:

Model_train.py

```
import os

import pickle

import pandas as pd

import numpy as np

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split


# Simulated dataset with Indian states

states = [

    "Andhra Pradesh", "Arunachal Pradesh", "Assam", "Bihar", "Chhattisgarh",

    "Goa", "Gujarat", "Haryana", "Himachal Pradesh", "Jharkhand", "Karnataka",

    "Kerala", "Madhya Pradesh", "Maharashtra", "Manipur", "Meghalaya",

    "Mizoram", "Nagaland", "Odisha", "Punjab", "Rajasthan", "Sikkim",
```

```
    "Tamil Nadu", "Telangana", "Tripura", "Uttar Pradesh", "Uttarakhand",
    "West Bengal", "Delhi", "Jammu and Kashmir"
]

# Generate sample data
np.random.seed(42)
n_samples = 500
X = pd.DataFrame({
    "area_sqft": np.random.uniform(500, 4000, size=n_samples),
    "bedrooms": np.random.randint(1, 6, size=n_samples),
    "bathrooms": np.random.randint(1, 4, size=n_samples),
    "floors": np.random.randint(1, 4, size=n_samples),
    "year_built": np.random.randint(1980, 2022, size=n_samples),
    "state": np.random.choice(states, size=n_samples)
})

# Simulate price as a function of area, rooms, etc.
y = (
    X["area_sqft"] * 50 +
    X["bedrooms"] * 200000 +
    X["bathrooms"] * 150000 +
    X["floors"] * 50000 +
    (2022 - X["year_built"]) * -1000 +
    np.random.normal(0, 100000, size=n_samples)
)

# One-hot encode the 'state' column
X = pd.get_dummies(X, columns=["state"])
```

```
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Save model and column names
os.makedirs("indian_states_app", exist_ok=True)
pickle.dump(model, open("indian_states_app/model.pkl", "wb"))
pickle.dump(X.columns.tolist(), open("indian_states_app/columns.pkl", "wb"))
pickle.dump(states, open("indian_states_app/states.pkl", "wb"))
```

app.py

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "App is working!"

if __name__ == "__main__":
    import os
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)

from flask import Flask, render_template, request
```

```
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import base64
from io import BytesIO

app = Flask(__name__)
model = pickle.load(open("model.pkl", "rb"))
columns = pickle.load(open("columns.pkl", "rb"))
states = pickle.load(open("states.pkl", "rb"))

@app.route("/", methods=["GET", "POST"])
def home():
    prediction = None
    plot_url = None

    if request.method == "POST":
        input_data = []

        for col in columns:
            if col.startswith("state_"):
                input_data.append(1 if col == f"state_{request.form['state']}" else 0)
            else:
                val = float(request.form.get(col, 0))
                input_data.append(val)

        input_df = pd.DataFrame([input_data], columns=columns)
        prediction = round(model.predict(input_df)[0], 2)
```

```
        # Plot
        plt.figure(figsize=(5, 3))
        plt.bar(["Predicted Price"], [prediction], color='mediumseagreen')
        plt.ylabel("Price (INR)")
        plt.title("House Price Estimate")
        buf = BytesIO()
        plt.tight_layout()
        plt.savefig(buf, format="png")
        plt.close()
        plot_url = base64.b64encode(buf.getvalue()).decode("utf-8")

    return render_template("index.html", input_columns=columns, states=states,
prediction=prediction, plot_url=plot_url)

if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0', port=8080)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Indian House Price Predictor</title>
    <style>

body {
            font-family: Arial, sans-serif;
            background: #e8f5e9;
```

```
    margin: 0;
    padding: 0;
}
.container {
    max-width: 720px;
    margin: 50px auto;
    background: #ffffff;
    padding: 35px;
    border-radius: 10px;
    box-shadow: 0px 4px 12px rgba(0,0,0,0.1);
}
h2 {
    text-align: center;
    color: #2e7d32;
}
form {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 15px;
}
label {
    font-weight: bold;
    margin-bottom: 5px;
    display: block;
}
input, select {
    width: 100%;
    padding: 8px;
    border: 1px solid #ccc;
```

```
            border-radius: 5px;
        }
        .full-width {
            grid-column: span 2;
        }
        button {
            padding: 10px;
            background: #2e7d32;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            font-size: 15px;
        }
        button:hover {
            background: #1b5e20;
        }
        .result {
            margin-top: 30px;
            text-align: center;
            font-size: 20px;
            color: #004d40;
        }
    </style>
</head>
<body>
<div class="container">
    <h2>🏠 Indian House Price Predictor</h2>
    <form method="POST">
```

```
<div>
    <label>Area (sq ft)</label>
    <input type="number" step="any" name="area_sqft" required>
</div>
<div>
    <label>Bedrooms</label>
    <input type="number" name="bedrooms" required>
</div>
<div>
    <label>Bathrooms</label>
    <input type="number" name="bathrooms" required>
</div>
<div>
    <label>Floors</label>
    <input type="number" name="floors" required>
</div>
<div>
    <label>Year Built</label>
    <input type="number" name="year_built" required>
</div>
<div class="full-width">
    <label>Select State</label>
    <select name="state">
        {% for state in states %}
            <option value="{{ state }}">{{ state }}</option>
        {% endfor %}
    </select>
</div>
<div class="full-width">
```

```
            <button type="submit">Predict</button>
        </div>
    </form>
    {% if prediction %}
    <div class="result">
        🔍 Estimated House Price: ₹ {{ prediction }}
        <br>
        <img src="{{ plot_url }}" alt="Graph">
    </div>
    {% endif %}
</div>
</body>
</html>
```

Conclusion: Thus we have successfully configured our mini-project and this is how we implemented it step by step