# Titanic Survival Prediction Model Deployment on Cloud

Name:  Titanic Survival Prediction Model Deployment on Cloud

Report date: 2025.03.03

Internship Batch: LISUM42

Version: 1.0

Data intake by: Abdukhakimov Asatilla

Data intake reviewer: Data Glacier Team

**Titanic Tabular data details:**

| | |
|---|---|
| **Total number of observations** | 887 |
| **Total number of files** | 1 |
| **Total number of features** | 8 |
| **Base format of the file** | .csv |
| **Size of the data** | 55.6KB |

**Dataset Selection**

The dataset chosen for this task is the Titanic dataset, which contains passenger details and survival status. The goal is to build a model that predicts whether a passenger survived or not based on their Pclass, Gender, Age, and Fare. The model is deployed on Railway Cloud.

## Step 1. Data Preparation

We have loaded the necessary packages and the Titanic dataset.

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pickle
```

[2]:
```
url = "https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/titanic.csv"
df = pd.read_csv(url)
df.head()
```

[2]:

| | Survived | Pclass | Name | Sex | Age | Siblings/Spouses Aboard | Parents/Children Aboard | Fare |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | Mr. Owen Harris Braund | male | 22.0 | 1 | 0 | 7.2500 |
| **1** | 1 | 1 | Mrs. John Bradley (Florence Briggs Thayer) Cum... | female | 38.0 | 1 | 0 | 71.2833 |
| **2** | 1 | 3 | Miss. Laina Heikkinen | female | 26.0 | 0 | 0 | 7.9250 |
| **3** | 1 | 1 | Mrs. Jacques Heath (Lily May Peel) Futrelle | female | 35.0 | 1 | 0 | 53.1000 |
| **4** | 0 | 3 | Mr. William Henry Allen | male | 35.0 | 0 | 0 | 8.0500 |

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 887 entries, 0 to 886
Data columns (total 8 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Survived                 887 non-null    int64
 1   Pclass                   887 non-null    int64
 2   Name                     887 non-null    object
 3   Sex                      887 non-null    int64
 4   Age                      887 non-null    float64
 5   Siblings/Spouses Aboard  887 non-null    int64
 6   Parents/Children Aboard  887 non-null    int64
 7   Fare                     887 non-null    float64
dtypes: float64(2), int64(5), object(1)
memory usage: 55.6+ KB
```

[3]:
```
df = df.dropna(subset=['Age', 'Fare', 'Sex'])
```

### Encode 'Sex' as 0 for male and 1 for female

[4]:
```
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
```

### Let's select features and target

[5]:
```
X = df[['Pclass', 'Age', 'Sex', 'Fare']]
y = df['Survived']
```

**Step 2. Model Training and Evaluation**

### Split data into training and test sets

[6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

### Train a Random Forest Classifier

[7]:
```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

[7]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

### Evaluate the model

[8]:
```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.8146067415730337
```

### Save the model

[9]:
```
# Save the trained model to a file
with open('titanic_model.pkl', 'wb') as f:
    pickle.dump(model, f)
```

Following training and evaluation, the model demonstrated an accuracy of 0.81, which is within acceptable limits.

**Step 3. Create the Flask App**

app.py

```python

from flask import Flask, request, jsonify, render_template
import pickle
import numpy as np
import os

# Load the trained model
with open('titanic_model.pkl', 'rb') as f:
    model = pickle.load(f)

app = Flask(__name__)

# Home route - serves the index.html file
@app.route('/')
def home():
    return render_template('index.html')

```

```python

# Prediction API route
@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    features = np.array([[data['Pclass'], data['Age'], data['Sex'], data['Fare']]])
    prediction = model.predict(features)
    survival = "Survived" if prediction[0] == 1 else "Not Survived"
    return jsonify({"prediction": survival})


if __name__ == "__main__":
    port = int(os.environ.get("PORT", 5000))
    debug = os.environ.get("DEBUG", "False").lower() == "true"
    app.run(host='0.0.0.0', port=port, debug=debug)
```

app.py  index.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>Titanic Survival Prediction</title>
</head>
<body>
    <h1>Titanic Survival Prediction</h1>
    <p>Enter passenger details to predict survival:</p>
    <form id="predictionForm">
        <label for="Pclass">Pclass:</label><input type="number" id="Pclass" name="Pclass"><br><br>
        <label for="Age">Age:</label><input type="number" id="Age" name="Age"><br><br>
        <label for="Sex">Sex (0 for female, 1 for male):</label><input type="number" id="Sex" name="Sex"><br><br>
        <label for="Fare">Fare:</label><input type="number" id="Fare" name="Fare"><br><br>
        <button type="button" onclick="predictSurvival()">Predict</button>
    </form>

    <div id="predictionResult"></div>
```

```html
    <script>
        function predictSurvival() {
            const form = document.getElementById('predictionForm');
            const Pclass = form.Pclass.value;
            const Age = form.Age.value;
            const Sex = form.Sex.value;
            const Fare = form.Fare.value;

            fetch('/predict', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({
                    Pclass: parseInt(Pclass),
                    Age: parseFloat(Age),
                    Sex: parseInt(Sex),
                    Fare: parseFloat(Fare)
                })
            })
            .then(response => response.json())
            .then(data => {
                document.getElementById('predictionResult').innerText = 'Prediction: ' + data.prediction;
            });
        }
    </script>
</body>
</html>
```

**Step 4. Deploy the Model on Railway Cloud**

Before deploying to Railway Cloud, we need to prepare the model by creating a requirements.txt file, a Procfile, and a Dockerfile.

dockerfile

```
FROM python:3.9-slim-buster

WORKDIR /app

COPY requirements.txt .


RUN pip install --no-cache-dir -U Flask Werkzeug -r
requirements.txt


COPY . .

EXPOSE 5000

CMD ["flask", "run", "--host=0.0.0.0"]
```

Procfile

```
web: gunicorn app:app --bind :$PORT
```

requirements.txt

```
Flask==3.1.0
scikit-learn==1.6.1
pandas==2.2.3
Werkzeug==3.1.3
```

A Docker container ensures consistent application execution across environments. Following the Docker file update, we proceed to build and execute the Docker container.

A) Build the Docker image:

*docker build -t flask-titanic-app .*

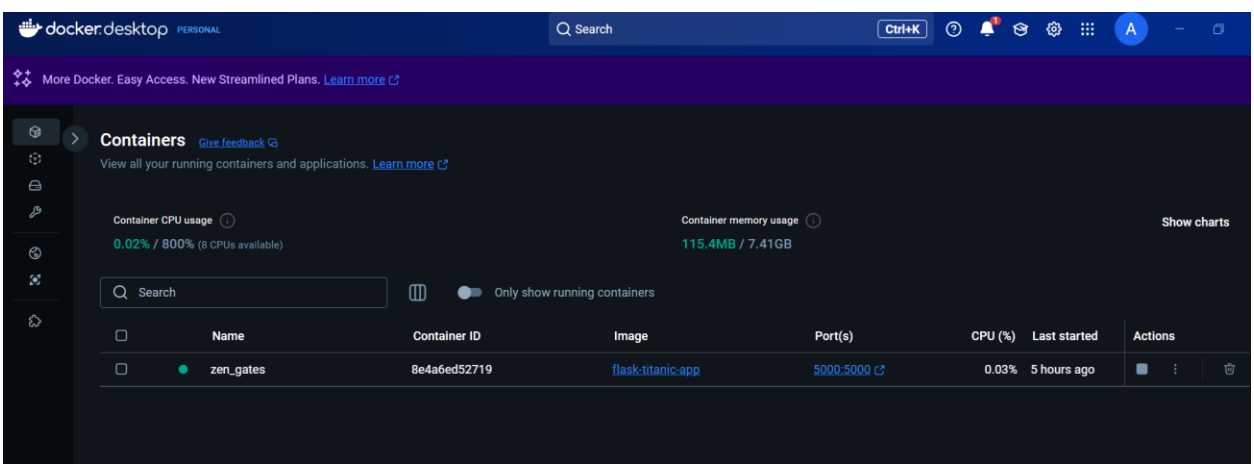
```
C:\Users\treme\Week 5 Added HTML>docker build -t flask-titanic-app .
[+] Building 37.8s (11/11) FINISHED                                                                    docker:desktop-linux
 => [internal] load build definition from dockerfile                                                                   0.0s
 => => transferring dockerfile: 255B                                                                                   0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim-buster                                             1.6s
 => [auth] library/python:pull token for registry-1.docker.io                                                          0.0s
 => [internal] load .dockerignore                                                                                      0.0s
 => => transferring context: 2B                                                                                        0.0s
 => [1/5] FROM docker.io/library/python:3.9-slim-buster@sha256:320a7a4250aba4249f458872adecf92eea88dc6abd2d76dc5c  0.0s
 => [internal] load build context                                                                                      0.0s
 => => transferring context: 747B                                                                                      0.0s
 => CACHED [2/5] WORKDIR /app                                                                                          0.0s
 => [3/5] COPY requirements.txt .                                                                                      0.1s
 => [4/5] RUN pip install --no-cache-dir -U Flask Werkzeug -r requirements.txt                                        33.1s
 => [5/5] COPY . .                                                                                                     0.1s
 => exporting to image                                                                                                 2.6s
 => => exporting layers                                                                                                2.6s
 => => writing image sha256:aac18a5631989acfd962dfb679623ccf87547196aea1c037b8e13867bb677667                           0.0s
 => => naming to docker.io/library/flask-titanic-app                                                                   0.0s
```

B) Run the Docker container:

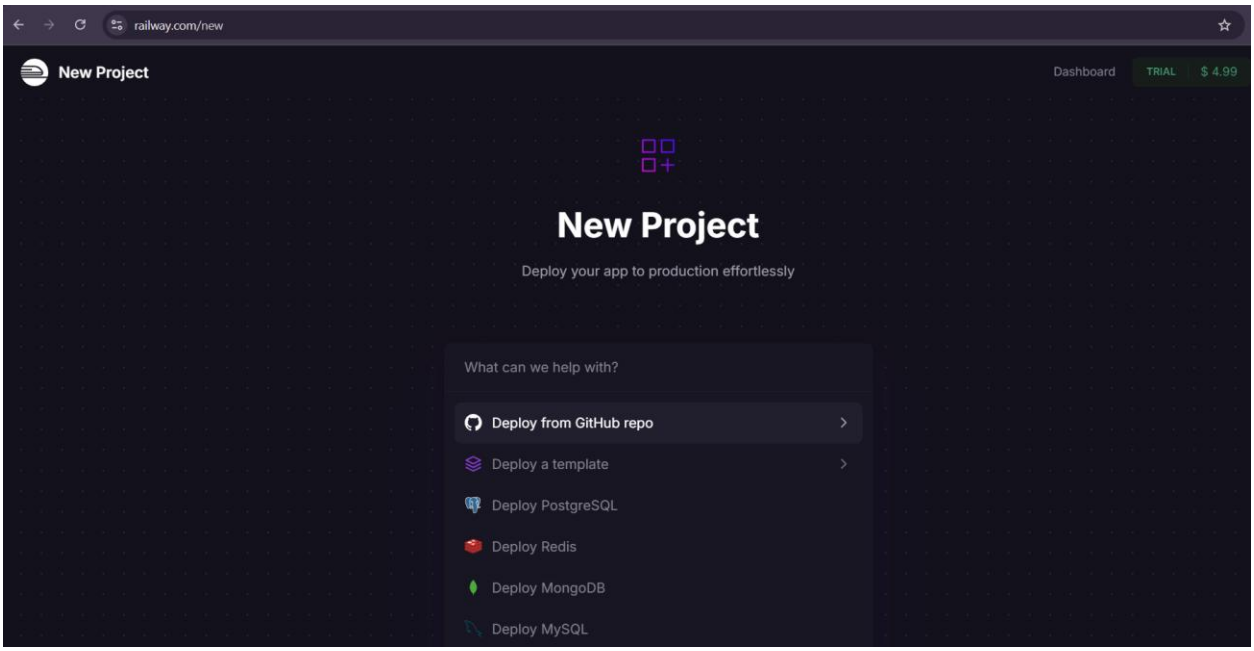
*docker run -p 5000:5000 flask-titanic-app*

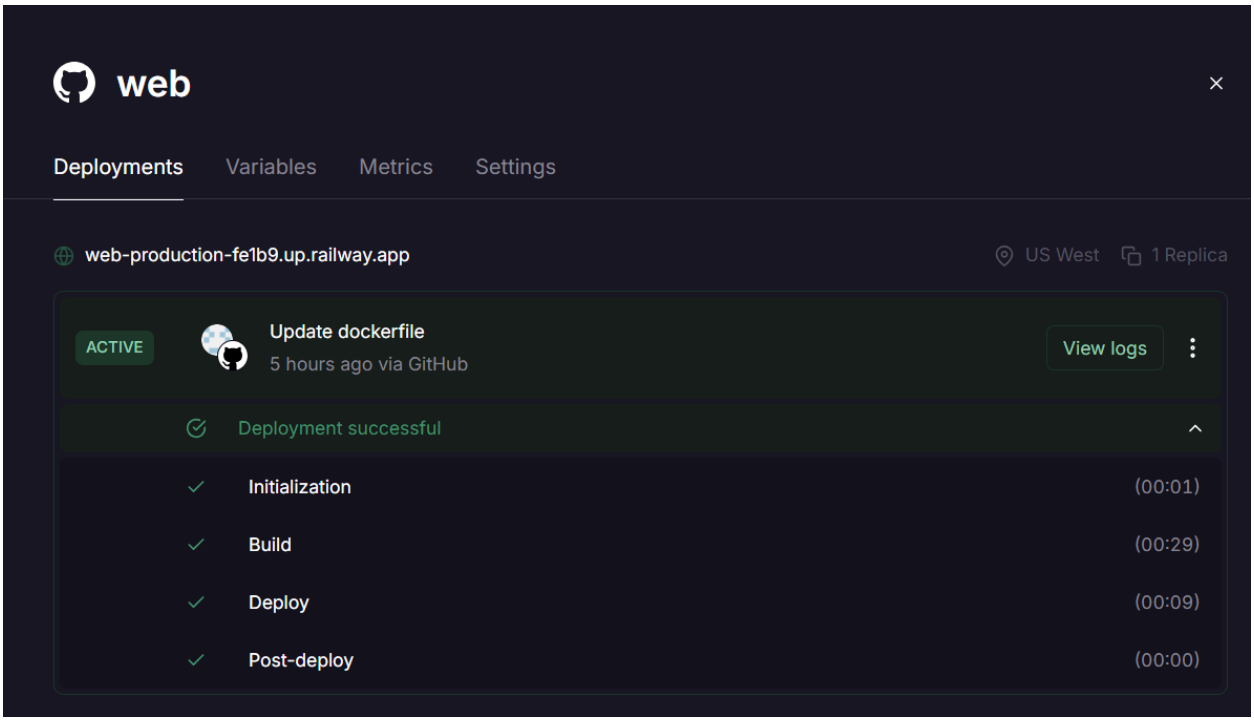
```
C:\Users\treme\Week 5 Added HTML>docker run -p 5000:5000 flask-titanic-app
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

We test the model locally with Docker Desktop.



All files have been uploaded to Github. **Railway Cloud** is used to deploy the trained model. Railway is a cloud platform that makes deploying web applications easy. It connects to GitHub repo, automatically builds an app, and provides a live URL.



web-production-fe1b9.up.railway.app

# Titanic Survival Prediction

Enter passenger details to predict survival:

Pclass: 3

Age: 22

Sex (0 for female, 1 for male): 1

Fare: 8

Predict

Prediction: Not Survived