# Streamlit App Deployment on AWS EC2 with Docker :

This document provides a step-by-step guide to deploying a Streamlit app on an AWS EC2 instance using Docker. By following these instructions, you will be able to host and access the app via a public IP address.

## Table of Contents

## 1. Introduction

This guide walks you through the process of deploying a Streamlit app on an AWS EC2 instance using Docker. The app is containerized for easy deployment and scalability. By the end of this guide, you will have a fully functional Streamlit app accessible via a public IP address.

## 2. Prerequisites

Before starting, ensure you have the following:

AWS Account: An active AWS account with access to EC2.
Docker Hub Account: A Docker Hub account to store the Docker image.
Local Machine Setup:
Docker installed and running.
SSH key pair (.pem file) for accessing the EC2 instance.

## 3. Project File Structure

The project has the following file structure:

```
Part2/
 ├── app.py
 ├── backend.py
 ├── requirements.txt
 ├── Dockerfile
```

### Key Files Explained

**app.py:**
This is the main Streamlit application file.
It defines the user interface (UI) and interacts with the backend to fetch or process data.

**backend.py:**
This file contains the backend logic for the app.
It processes data, performs calculations, or interacts with external APIs/databases.

**requirements.txt:**
This file lists all the Python dependencies required for the app.

**Dockerfile:**
This file defines the Docker image for the app.
It specifies the base image, installs dependencies, and runs the app.

## 4. How Docker Works in This Deployment

Docker is a containerization platform that packages an application and its dependencies into a lightweight, portable container. Here's how Docker is used in this deployment:

**1. Containerization:**
The Streamlit app and its dependencies are packaged into a Docker container.
This ensures that the app runs consistently across different environments (e.g., local machine, EC2 instance).

**2. Docker Image:**
A Docker image is created using the Dockerfile. This image contains the app code, dependencies, and runtime environment.
The image is stored in Docker Hub, a cloud-based repository for Docker images.

**3. Docker Container:**
A Docker container is a running instance of a Docker image.
On the EC2 instance, the Docker container runs the Streamlit app and makes it accessible via a specified port (e.g., port 8501).

**4. Portability:**
Docker allows the app to be easily moved between environments (e.g., from a local machine to an EC2 instance) without compatibility issues.

## 5. Step-by-Step Deployment

**Step 1: Set Up an EC2 Instance**

Log in to the AWS Management Console and navigate to the EC2 dashboard.
Launch a new EC2 instance using the Amazon Linux 2023 AMI.
Select an instance type (e.g., t2.micro).
Configure the instance details and add storage.

Configure the security group to allow inbound traffic on:
SSH (Port 22): For SSH access.
Custom TCP (Port 8501): For the Streamlit app.
Launch the instance and download the .pem key pair for SSH access.

**Step 2: SSH into the EC2 Instance**

Open a terminal on your local machine.
Use the .pem key pair to SSH into the EC2 instance:

```
ssh -i /path/to/key-only.pem ec2-user@<public-ip>
```

Note : Replace <public-ip> with your EC2 instance's public IP.

**Step 3: Install Docker on EC2**

Update the package index:

```
sudo yum update -y
```

Install Docker:

```
sudo yum install docker -y
```

Start and enable the Docker service:

```
sudo systemctl start docker
sudo systemctl enable docker
```

Verify Docker is installed:

```
docker --version
```

**Step 4: Build and Push Docker Image to Docker Hub**

On your local machine, navigate to the project directory containing the Dockerfile.
Build the Docker image:

```
docker build -t my-streamlit-app .
```

Tag the image with your Docker Hub username:

```
docker tag my-streamlit-app <your-dockerhub-username>/my-streamlit-app:latest
```

Log in to Docker Hub:

```
docker login
```

Push the image to Docker Hub:

```
docker push <your-dockerhub-username>/my-streamlit-app:latest
```

**Step 5: Run the Docker Container on EC2**

SSH into the EC2 instance (if not already logged in).
Pull the Docker image from Docker Hub:

```
docker pull <your-dockerhub-username>/my-streamlit-app:latest
```

Run the Docker container:

```
docker run -d -p 8501:8501 --name my-app <your-dockerhub-username>/my-streamlit-
app:latest
```

**Step 6: Access the Streamlit App**

Open a web browser and navigate to:

```
http://<public-ip>:8501
```

Replace <public-ip> with your EC2 instance's public IP.

The Streamlit app should now be accessible.

## 6. Usage Instructions

Accessing the App
The app is accessible at http://<public-ip>:8501, where <public-ip> is the public IP address of the EC2 instance.
Stopping and Restarting the Container

Stop the container:
```
docker stop my-app
```

Start the container:
```
docker start my-app
```

Restart the container:
```
docker restart my-app
```

## 7. Optional Enhancements

- Set Up a Custom Domain
- Register a domain using Route 53 or another registrar.
- Point the domain to the EC2 instance’s public IP address.
- Enable HTTPS
- Install Certbot to obtain an SSL certificate.
- Configure the certificate to enable HTTPS for the app.
- Automate Deployment with CI/CD
- Use CI/CD tools like GitHub Actions or AWS CodePipeline to automate the build and deployment process.

## 8. Troubleshooting

Common Issues and Solutions

App Not Accessible:
Check the security group settings to ensure inbound traffic is allowed on port 8501.
Verify that the Docker container is running.
Check the container logs for errors.

Docker Permission Denied:

Add the ec2-user to the docker group to grant necessary permissions.
Log out and log back in to apply the changes.
Docker Hub Push/Pull Issues:
Ensure you are logged in to Docker Hub.
Verify the repository name and permissions.

## 9. Conclusion:

This guide provides a comprehensive walkthrough for deploying a Streamlit app on AWS EC2 using Docker. By following these steps, you can easily host and scale your app in the cloud. For further enhancements, consider setting up a custom domain, enabling HTTPS, or automating the deployment process.