Deploying a Flask Application with Docker on AWS EC2: Step-by-Step Guide

Introduction

This document serves as a comprehensive guide to containerizing a Python Flask application using Docker and deploying it on an AWS EC2 instance. It covers all steps including setting up the Dockerfile, resolving common errors, and configuring the AWS security group to expose the Flask application to the public IP.

Prerequisites

Before starting, ensure that you have the following installed on your machine:
1. Docker: Install Docker from https://docs.docker.com/get-docker/.
2. Python Application: A simple Python application (app.py and requirements.txt).
3. AWS EC2 instance with public IP access.
4. Security group configured to allow inbound traffic on port 5000.

Step 1: Creating the Python Application

1. Create a folder for the application:
  ```bash
  mkdir my-python-app
  cd my-python-app
  ```
2. Create the app.py file with the following content:
  ```python
  from flask import Flask
  app = Flask(__name__)

  @app.route('/')
  def hello():
      return 'Hello, World!'

  if __name__ == '__main__':
      app.run(host='0.0.0.0', port=5000)
  ```
3. Create the requirements.txt file to list the dependencies:
  ```text
  ```

```
Flask==2.0.1
```

Step 2: Writing the Dockerfile

Create a Dockerfile in the same directory as app.py and requirements.txt. Use the following content:

```dockerfile
# Step 1: Specify the base image
FROM python:3.9-slim

# Step 2: Set the working directory
WORKDIR /app

# Step 3: Copy the local files to the container
COPY . /app

# Step 4: Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Step 5: Expose the required port
EXPOSE 5000

# Step 6: Command to run the Flask app
CMD ['python', 'app.py']
```

Step 3: Building the Docker Image

To build the Docker image, run the following command in your terminal:

```bash
docker build -t python-app .
```

This will create a Docker image using the instructions specified in the Dockerfile.

Step 4: Running the Docker Container

Run the container using this command:

```bash
docker run -p 5000:5000 python-app
```

This will bind port 5000 of the container to port 5000 of the host machine, making the Flask app accessible.

Step 5: Configuring AWS EC2 Security Group

If you are using an AWS EC2 instance, make sure your security group is configured to allow inbound traffic on port 5000. Here's how you can do that:
1. Go to the EC2 Management Console.
2. Select your instance and navigate to the 'Security Groups' section.
3. Edit inbound rules to allow traffic on port 5000 from `0.0.0.0/0` (or restrict it to specific IPs for better security).

Step 6: Accessing the Flask App via Public IP

Once the container is running, the Flask app should be accessible via the public IP of your EC2 instance on port 5000. In a browser or via curl, you can access the app at:

```
http://<public-ip-of-ec2-instance>:5000
```

This will display the 'Hello, World!' message served by the Flask app.

Step 7: Common Errors and Fixes

1. **IndentationError: unexpected indent**
   Ensure that your `app.py` file has consistent indentation. Python relies on proper indentation to define blocks of code. Make sure you are using spaces consistently (4 spaces per indentation level) and avoid mixing tabs and spaces.

2. **ImportError: cannot import name 'url_quote' from 'werkzeug.urls'**
   This error arises due to an incompatibility between Flask and Werkzeug versions. To fix it, specify compatible versions in `requirements.txt`:

   ```text
   Flask==2.0.1
   Werkzeug==2.0.3
   ```

3. **OSError: [Errno 99] Cannot assign requested address**
   Ensure that you are binding to `0.0.0.0` in the `app.py` file, rather than a specific IP address. This allows Flask to listen on all available interfaces within the container. In `app.py`, the following line ensures that the app binds correctly:

```python
app.run(host='0.0.0.0', port=5000)
```

4. **Application running on internal container IP (172.x.x.x)**
   This is expected behavior as Docker assigns a private network to containers. However, by mapping the container's port to the host’s port using `-p 5000:5000`, the app can be accessed using the EC2 instance's public IP.

Conclusion

You have successfully containerized a Flask application and deployed it on AWS EC2 using Docker. By following the above steps, you can run the Flask app and resolve common issues related to Docker, Flask, and AWS networking.