

Project : Containerized Web App Deployment using Azure Container Apps

Project Steps

Step 1: Set Up Azure Environment

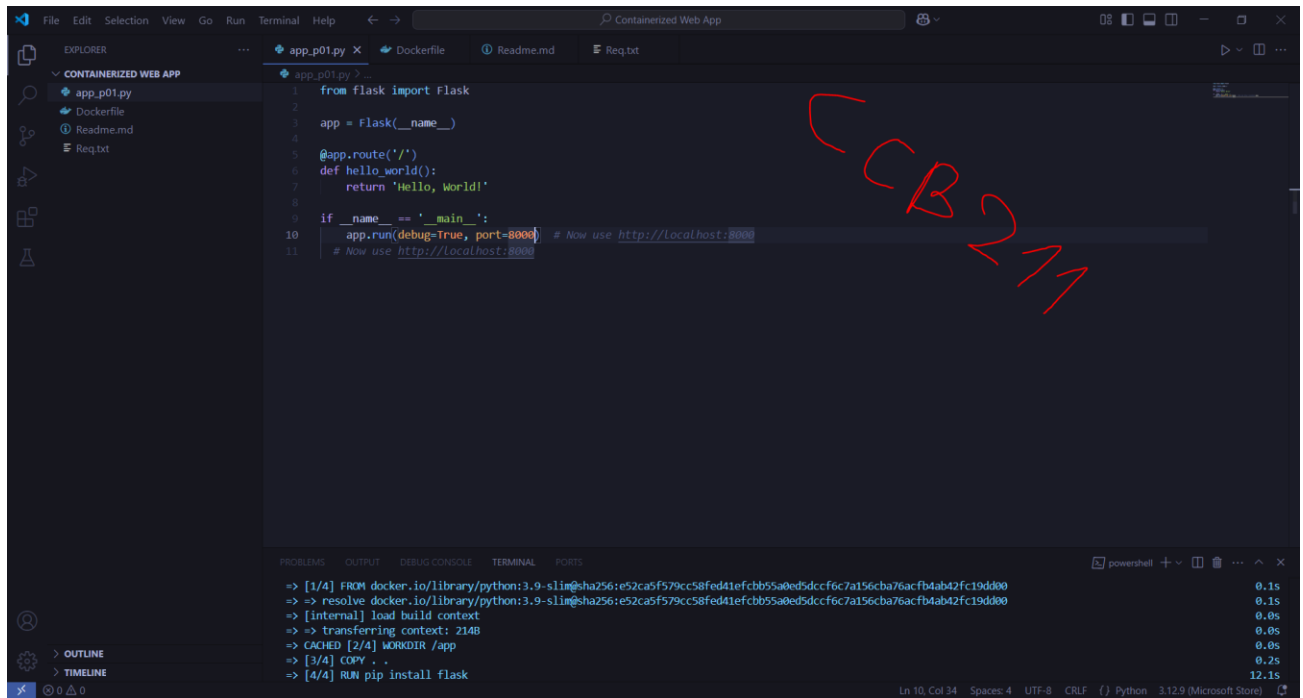
- **Description:** Created a resource group to manage all project resources.
- **Screenshot:**

The screenshot shows the Microsoft Azure portal interface. At the top, there's a blue header bar with the Microsoft Azure logo, a search bar, and user information. Below the header, the main content area displays the 'Containerizedwebapp' resource group. The left sidebar contains navigation links: Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings, Cost Management, Monitoring, Automation, and Help. The main content area shows a list of resources under the 'Resources' tab. The list includes five resources: 'contwebapp' (Container registry), 'managedEnvironment-Containerizedwe-98f4' (Container Apps Environment), 'project-contapps' (Container App), 'pythoncontwebapp' (Container instances), and 'workspacecontainerizedwebapp97ae' (Log Analytics workspace). All resources are located in 'UAE North'. The page also features a 'Filter for any field...' dropdown, a 'Type equals all' filter, and a 'Location equals all' filter. The bottom of the page shows a pagination bar with '< Previous', 'Page 1 of 1', and 'Next >' buttons. An 'Activate Windows' watermark is visible in the bottom right corner.

Name	Type	Location
contwebapp	Container registry	UAE North
managedEnvironment-Containerizedwe-98f4	Container Apps Environment	UAE North
project-contapps	Container App	UAE North
pythoncontwebapp	Container instances	UAE North
workspacecontainerizedwebapp97ae	Log Analytics workspace	UAE North

Step 2: Develop the Web Application

- **Description:** Built a web app using flask and tested it locally.
- **Screenshot:**



The screenshot shows a Visual Studio Code editor window titled "Containerized Web App". The Explorer sidebar on the left shows a project named "CONTAINERIZED WEB APP" with files: `app.p01.py`, `Dockerfile`, `Readme.md`, and `Req.txt`. The main editor displays the content of `app.p01.py`, which is a Flask application:

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def hello_world():
7     return 'Hello, World!'
8
9 if __name__ == '__main__':
10     app.run(debug=True, port=8000) # Now use http://localhost:8000
11 # Now use http://localhost:8000
```

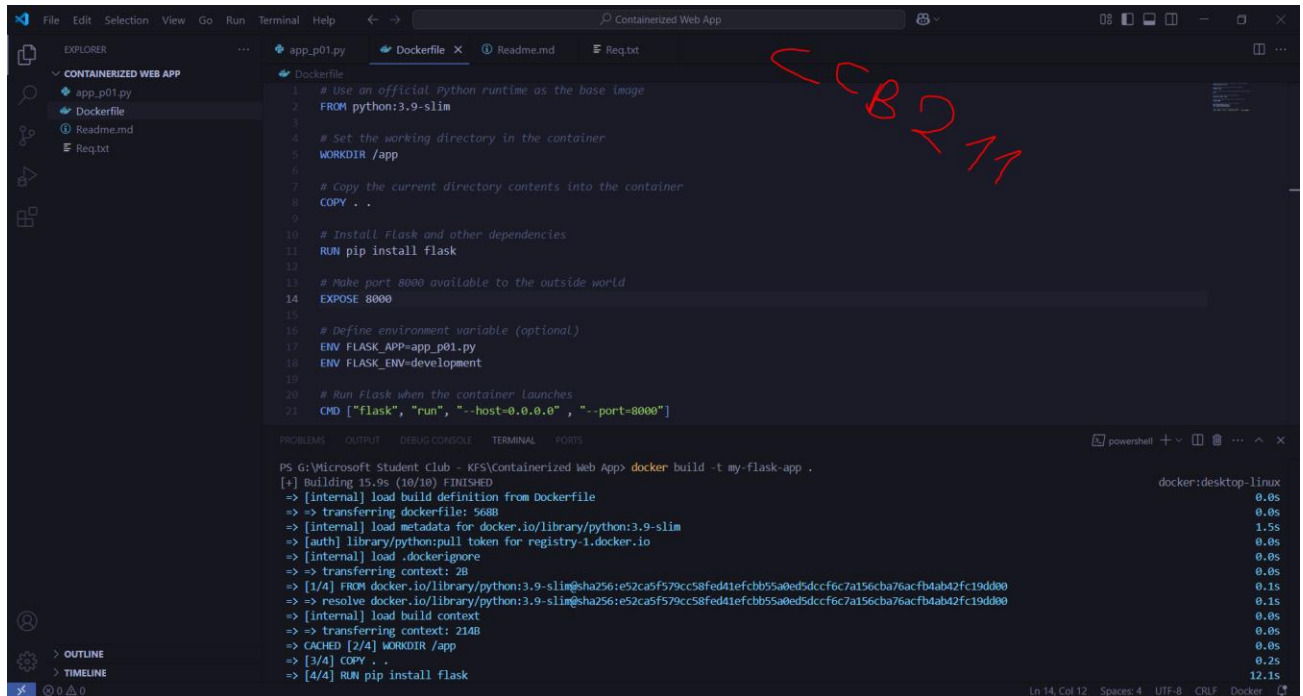
A large red handwritten watermark "CCB211" is visible across the code. The bottom panel of the editor shows the "TERMINAL" output, which displays the Docker build process:

```
=> [1/4] FROM docker.io/library/python:3.9-slim@sha256:e52ca5f579cc58fed41efcb55a0ed5dccf6c7a156cha76acfb4ab42fc19dd00 0.1s
=> => resolve docker.io/library/python:3.9-slim@sha256:e52ca5f579cc58fed41efcb55a0ed5dccf6c7a156cha76acfb4ab42fc19dd00 0.1s
=> [internal] load build context 0.0s
=> => transferring context: 214B 0.0s
=> CACHED [2/4] WORKDIR /app 0.0s
=> [3/4] COPY . . 0.2s
=> [4/4] RUN pip install flask 12.1s
```

The status bar at the bottom indicates the file encoding is UTF-8, line endings are CRLF, and the editor is running Python 3.12.9 (Microsoft Store).

Step 3: Containerize the Web App

- **Description:** Created a Dockerfile to package the web app into a Docker container.
- **Screenshot:**



The screenshot shows a Visual Studio Code editor window with a project named "Containerized Web App". The Explorer sidebar on the left shows the project structure with files: app_p01.py, Dockerfile, Readme.md, and Req.txt. The Dockerfile is open in the editor, showing the following content:

```
1 # Use an official Python runtime as the base image
2 FROM python:3.9-slim
3
4 # Set the working directory in the container
5 WORKDIR /app
6
7 # Copy the current directory contents into the container
8 COPY . .
9
10 # Install Flask and other dependencies
11 RUN pip install flask
12
13 # Make port 8000 available to the outside world
14 EXPOSE 8000
15
16 # Define environment variable (optional)
17 ENV FLASK_APP=app_p01.py
18 ENV FLASK_ENV=development
19
20 # Run Flask when the container launches
21 CMD ["flask", "run", "--host=0.0.0.0", "--port=8000"]
```

Handwritten red text "CCB211" is visible in the top right corner of the editor window.

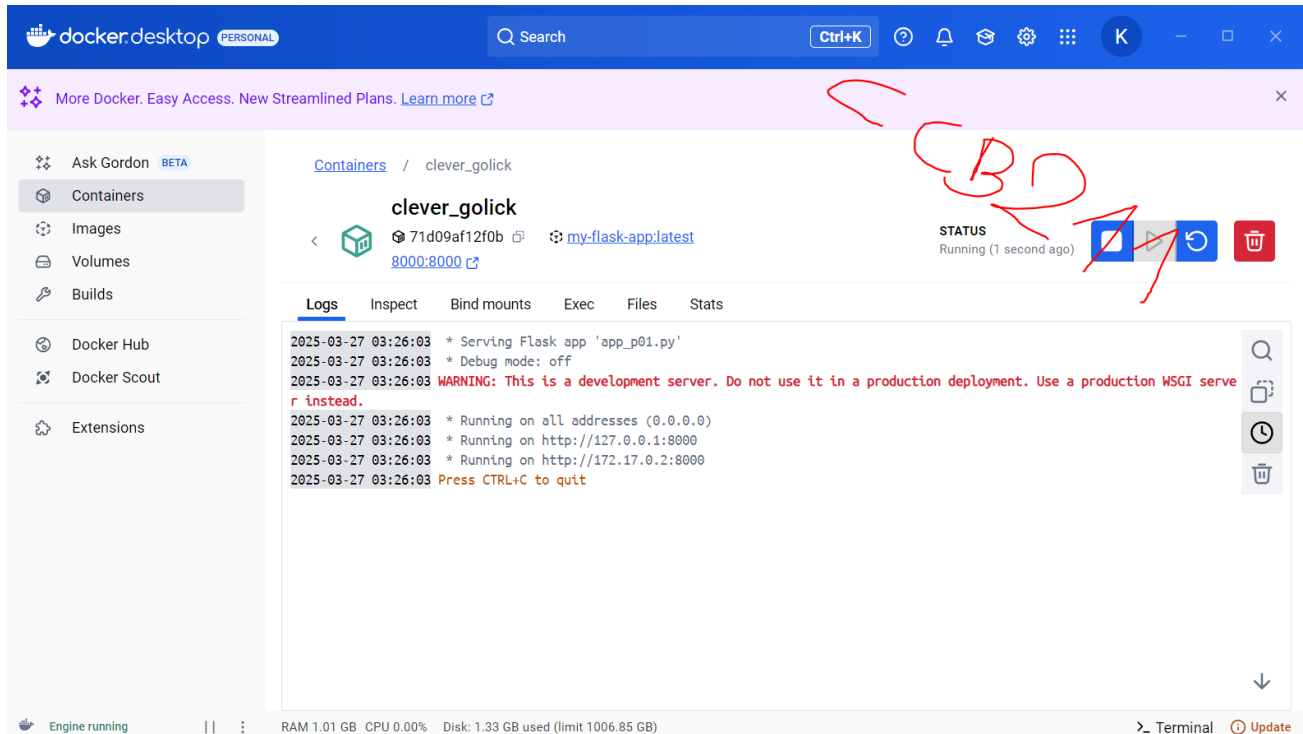
The Output window at the bottom shows the command `docker build -t my-flask-app .` and its execution progress:

```
PS G:\Microsoft Student Club - RFS\Containerized Web App> docker build -t my-flask-app .
[+] Building 15.9s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 568B
=> [internal] load metadata for docker.io/library/python:3.9-slim
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/python:3.9-slim@sha256:e52ca5f579cc58fed41efcb55a0ed5dccc6c7a156cba76acfb4ab42fc19dd00
=> => resolve docker.io/library/python:3.9-slim@sha256:e52ca5f579cc58fed41efcb55a0ed5dccc6c7a156cba76acfb4ab42fc19dd00
=> [internal] load build context
=> => transferring context: 214B
=> CACHED [2/4] WORKDIR /app
=> [3/4] COPY . .
=> [4/4] RUN pip install flask
```

The status bar at the bottom indicates the current line is 14, column 12, with 4 spaces, in UTF-8 encoding, CRLF line endings, and the Docker extension is active.

Step 4: Test the Container Locally

- **Description:** Ran the Docker container locally to verify it works.
- **Screenshot:**



Step 5: Deploy to Azure Container Apps

- **Description:** Deployed the containerized app to Azure Container Apps using a deployment method.
- **Screenshot:**

The screenshot shows the Microsoft Azure portal interface for a Container App named 'project-contapps'. The top navigation bar includes the Microsoft Azure logo, a search bar, and the user's profile. The sidebar on the left lists various navigation options under 'Overview', including Activity log, Access control (IAM), Tags, Diagnose and solve problems, Resource visualizer, Application, Revisions and replicas, Containers, Scale, Volumes, Settings, and Monitoring. The main content area displays the 'Essentials' section for the Container App, showing details such as Resource group (Containerizedwebapp), Status (Running), Location (UAE North), Subscription (Azure for Students), Subscription ID, and .NET Aspire Dashboard. Below this, there are tabs for 'Get started', 'Properties', and 'Monitoring'. The 'Get started' tab is active, showing a 'Discover Azure Container Apps Features' section with three main features: Upload artifact, Manage your app with revisions, and Set up continuous deployment. A handwritten note 'CCB 217' is visible in the top right corner of the screenshot.

Step 6: Configure Auto-Scaling

- **Description:** Set up auto-scaling based on HTTP traffic.
- **Screenshot:**

Microsoft Azure

Home > project-contapps

project-contapps | Scale

Scale rule settings

Control automatic scaling by setting the range of application replicas that'll be deployed in response to a trigger event. Use scale rules to determine the type of events that trigger scaling. [Learn more](#)

Min replicas 0 Min: 0

Max replicas 10 Max: 1000

Cooldown period 300

Polling interval 30

Current number of replicas 1 (View Details)

Scale rules

Search

+ Add

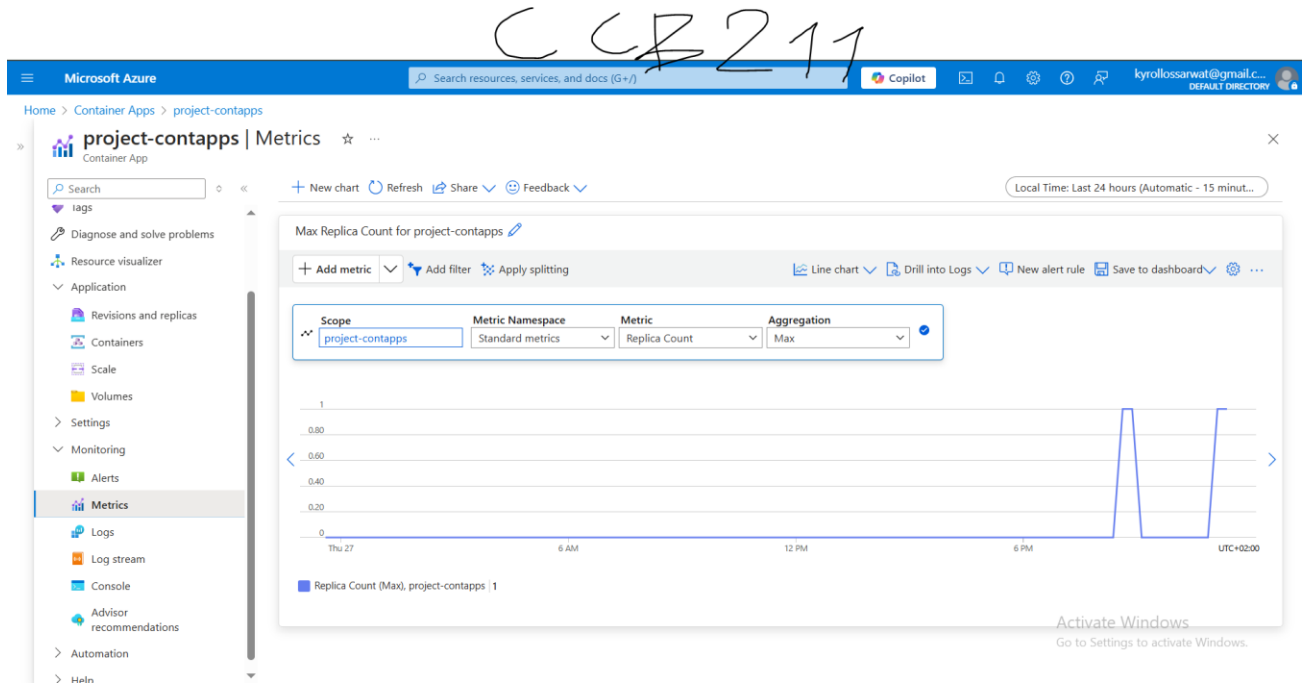
Name ↑	Type ↑	Del...
http-scaler	HTTP scaling	

Save as a new revision Cancel

Activate Windows
Go to Settings to activate Windows.

Step 7: Integrate Monitoring with Application Insights

- **Description:** Enabled Application Insights to monitor app health and performance.
- **Screenshot:**



Screenshots of Final Output:

