

## **DevOps Capstone Project: Automated Client-Server Workflow Deployment with Monitoring**

**Submitted by:**

**Zia Ullah Khan**

---

### **Overview:**

This project simulates a real-world production workflow as part of the DevOps curriculum at **Dice Pakistan**. It demonstrates the complete CI/CD lifecycle by building and dockerizing a server-client application where the server creates a file and its checksum, and the client receives and verifies it. The system is deployed on separate AWS EC2 instances using Terraform, monitored via Grafana, and integrated with GitHub Actions for automated deployment pipelines.

---

### **Course Title:**

DevOps – Batch 13, Dice Analytics Pakistan

### **Project Mentor:**

**[Asad Mahmand /Admad Talha]**

### **Completion Date:**

August 2025

### **Tech Stack & Tools:**

- **Docker & Docker Compose**
- **AWS EC2 (t2.micro)**
- **Terraform**
- **Grafana**
- **GitHub & GitHub Actions**
- **Docker Hub**
- **Linux**

## Project Summary (5–6 lines)

This DevOps project simulates a real-world production workflow. You'll build and dockerize a server-client application where the server sends a 1KB file and checksum, and the client verifies it. Both containers are deployed on separate AWS EC2 instances, provisioned with Terraform. You'll implement monitoring using Grafana and automate deployments through CI/CD pipelines. The whole process will be versioned in GitHub and documented thoroughly for reproducibility and clarity.

## Technologies and Tools Used

- **Docker & Docker Compose** – For containerization of server and client.
- **AWS EC2 (t2.micro)** – Free-tier virtual machines for deployment.
- **Terraform** – For Infrastructure as Code (IaC).
- **Grafana** – For system and container monitoring.
- **GitHub** – Code hosting and version control.
- **GitHub Actions** – For setting up CI/CD pipelines.
- **Docker Hub** – For pushing container images.
- **Slack (or webhook tool)** – For deployment notifications. [To be covered in future]

## Project Milestones

We'll structure our plan into 5 major milestones:

1. **Containerization:** Build and run the server and client in Docker with Docker Compose.
2. **Infrastructure Setup:** Launch EC2 instances with Terraform and configure networking.
3. **Monitoring Stack:** Set up Grafana on both EC2 instances to monitor system + containers.
4. **CI/CD Pipelines:** Automate testing, builds, and deployment via GitHub Actions & Docker Hub.
5. **Documentation & Final Submission:** Prepare README, diagrams, troubleshooting tips, and submission materials.

## Prerequisites & Subscriptions

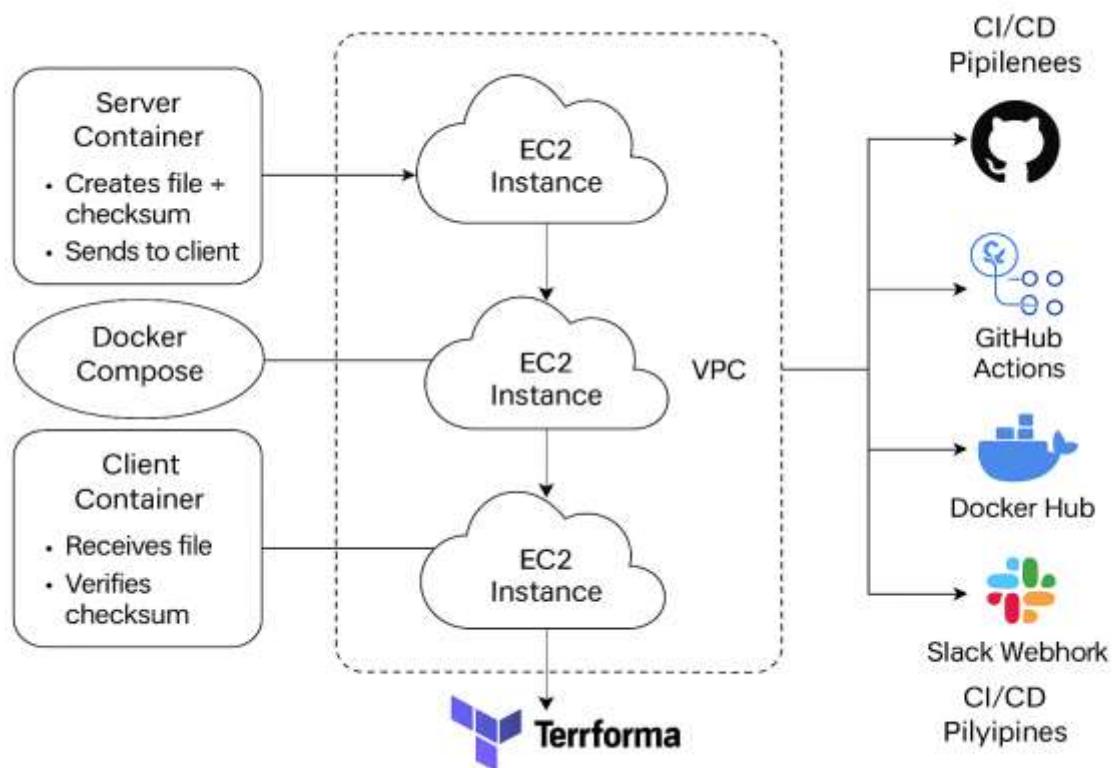
You'll need:

- **AWS Free Tier Account** (for EC2 & VPC) 
- **GitHub Account** 
- **Docker Hub Account** 
- **Slack account** or similar for webhook notification (optional) 
- **A system with Docker, Git, and Terraform installed** 

## DevOps Capstone Project - Architecture Overview

Here's what the diagram will illustrate:

- **Two Containers:**
  - **Server Container** (creates file + checksum and sends it)
  - **Client Container** (receives file, verifies checksum)
- **Docker Compose** is used to manage both.
- **Two EC2 Instances:**
  - One for **server**
  - One for **client**
- **Terraform** provisions EC2, VPC, and networking.
- **Grafana Monitoring Stack** installed on both EC2s.
- **CI/CD Pipelines:**
  - GitHub → GitHub Actions → Docker Hub (build/push images)
  - EC2 pulls latest image and redeploys via Docker Compose.
  - **Slack Webhook** notifies of deployments.



## Day 1: Project Initialization + App Skeleton Setup (Local Development)

### Day 1 Tasks: Server & Client App Skeleton (Local)

Today we will:

1. Create folder structure for both server and client.
2. Initialize Git repositories.
3. Create simple placeholder server & client apps (any language — let's use **Python** for simplicity).
4. Test local communication (later we'll Dockerize and deploy).

#### 📁 Folder Structure (You can create two GitHub repos later)

```
pgsql
devops-capstone/
├── server/
│   ├── app/
│   │   └── server.py
│   └── requirements.txt
└── client/
    ├── app/
    │   └── client.py
    └── requirements.txt
```

Copy Edit

#### 🧠 What will these apps do?

- `server.py` : creates a 1KB file with random text, calculates checksum, sends both over socket.
- `client.py` : receives the file, saves it, checks checksum.

```
xx Select C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5965]
(c) Microsoft Corporation. All rights reserved.

D:\Devops13\week8\devopscapstone\client\app>python client.py
Traceback (most recent call last):
  File "client.py", line 20, in <module>
    main()
  File "client.py", line 7, in main
    file_data, received_checksum = received.split(b":")
ValueError: not enough values to unpack (expected 2, got 1)

D:\Devops13\week8\devopscapstone\client\app>
```

```
xx Select C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5965]
(c) Microsoft Corporation. All rights reserved.

D:\Devops13\week8\devopscapstone\server\app>python server.py
Server is waiting for connection...
Connected to ('127.0.0.1', 60647)
Traceback (most recent call last):
  File "server.py", line 24, in <module>
    main()
  File "server.py", line 18, in main
    data, checksum = create_file()
  File "server.py", line 5, in create_file
    with open("/tmp/file.txt", "wb") as f:
FileNotFoundError: [Errno 2] No such file or directory: '/tmp/file.txt'

D:\Devops13\week8\devopscapstone\server\app>
```

Make errors like /tmp/ which is linux folder and i am on windows then re-run first server then client.

## ✗ 1. Server Error:

vbnet

Copy Edit

```
FileNotFoundException: [Errno 2] No such file or directory: '/tmp/file.txt'
```

### 🔍 Reason:

You're on Windows, and the `/tmp` directory is a **Linux/Mac thing** — it doesn't exist on Windows by default.

### ✓ Fix:

Change the path to a local folder that exists (or just use the current folder):

Update `server.py` — line 5 and 18:

python

Copy Edit

```
with open("file.txt", "wb") as f: # write to local directory
```

## ✗ 2. Client Error:

perl

Copy Edit

```
ValueError: not enough values to unpack (expected 2, got 1)
```

### 🔍 Reason:

The client is trying to split the received data on `b"::"` but it's not found in the data — possibly because the server crashed before sending anything.

### ✓ Fix:

Add error handling + print what's received. For now, fix the server first, then re-run both. Then modify `client.py` like this:

python

Copy Edit

```
received = s.recv(2048)
print("Raw received data:", received)

if b"::" in received:
    file_data, received_checksum = received.split(b"::")
    ...
else:
    print("✗ Split delimiter not found in received data.")
```

The screenshot shows two separate Windows command-line windows. Both windows have the title bar 'Select C:\Windows\System32\cmd.exe' and show Microsoft Windows [Version 10.0.19045.5965] (c) Microsoft Corporation. All rights reserved.

**Client Window:**

```
D:\Devops13\week8\devopscapstone\client\app>python client.py
Traceback (most recent call last):
  File "client.py", line 20, in <module>
    main()
  File "client.py", line 7, in main
    file_data, received_checksum = received.split(b":")
ValueError: not enough values to unpack (expected 2, got 1)
```

**Server Window:**

```
D:\Devops13\week8\devopscapstone\server\app>python server.py
Server is waiting for connection...
Connected to ('127.0.0.1', 60647)
Traceback (most recent call last):
  File "server.py", line 24, in <module>
    main()
  File "server.py", line 18, in main
    data, checksum = create_file()
  File "server.py", line 5, in create_file
    with open("/tmp/file.txt", "wb") as f:
FileNotFoundError: [Errno 2] No such file or directory: '/tmp/file.txt'

D:\Devops13\week8\devopscapstone\server\app>python server.py
Server is waiting for connection...
Connected to ('127.0.0.1', 60792)
File and checksum sent.
```

### Updated Plan:

1. Fix `server.py` file write path.
2. Rerun server.
3. Then rerun client.
4. Check console for success or further issues.

## Day 2 Objectives

1. Create Dockerfile for `server` and `client`.
  2. Mount named volumes:
    - `servervol:/app/data`
    - `clientvol:/app/data`
  3. Build images and run with **Docker Compose**.
  4. Test end-to-end communication inside containers.
- 

### Folder Structure Recap

pgsql

 Copy  Edit

```
devops-capstone/
├── server/
│   ├── app/
│   │   └── server.py
│   ├── Dockerfile
│   └── docker-compose.yml
└── client/
    ├── app/
    │   └── client.py
    ├── Dockerfile
    └── docker-compose.yml
```

We'll eventually use one Docker Compose file to run both containers. But let's first build individual Dockerfiles.

## 🌐 Step 1: Dockerfile for Server

```
server/Dockerfile
Dockerfile
# Use Python base image
FROM python:3.10-slim

WORKDIR /app

# Copy server app
COPY app/server.py .

# Create a folder for volume mount
RUN mkdir /app/data

CMD ["python", "server.py"]
```

## 🌐 Step 2: Dockerfile for Client

```
client/Dockerfile
Dockerfile
FROM python:3.10-slim

WORKDIR /app

# Copy client app
COPY app/client.py .

# Create folder for mounted volume
RUN mkdir /app/data

CMD ["python", "client.py"]
```



And now I am creating the new file at the root level: docker-compose.yml

## ⚙️ Step 3: Docker Compose for Both

Let's use a **single root-level file** for both services.

`docker-compose.yml`

yaml

Copy Edit

```
version: '3.9'

services:
  server:
    build: ./server
    container_name: devops_server
    volumes:
      - servervol:/app/data
    ports:
      - "5000:5000"
    networks:
      - devnet

  client:
    build: ./client
    container_name: devops_client
    volumes:
      - clientvol:/app/data
    depends_on:
      - server
    networks:
      - devnet

volumes:
  servervol:
  clientvol:

networks:
  devnet:
```

## Step 4: Build and Run

In the project root (same level as `docker-compose.yml`):

```
bash
docker-compose build
docker-compose up
```

Expected outcome:

- `server` creates `file.txt` in `/app/data` and sends it.
- `client` receives and verifies it.
- Output shows: File received and verified successfully.

As a result of docker compose build and up, i got the following output

```
Terminal
==> sha256-92d3ec5cbeb134fbad429f901fffd8e78f99d9e388b5389d2925d7dbaffb48e 15.65MB / 15.65MB
==> sha256-9ebc811f9ec7a9e99ff7305b18473f3f92e4f704f36c7475f25e48282 3.51MB / 3.51MB
==> sha256-3da95a905e546f99c4954497923e681757080651a338ee3f1f5e69f75e6923d 20.23MB / 20.23MB
=> extracting sha256-9da95a905e546f99c4954497923e681757080651a338ee3f1f5e69f75e6923d
=> extracting sha256-9ebc811f9ec7a9e99ff7305b18473f3f92e4f704f36c7475f25e48282
=> extracting sha256-92d3ec5cbeb134fbad42929919f8fe7bf18d9e6188b5590d7929d75ddafab8e
=> extracting sha256-64b78282ca882236c84cb6df90e72b75e6ea52c9ac7380c8435b6663002c403f
[+] [Server Internal] load build context
=> transferring context: 64KB
=> CHOKED [client 2/4] WORKDIR /app
=> [server 3/4] COPY app/server.py
=> transferring context: 64KB
=> [client 3/4] COPY app/client.py ..
=> [client 4/4] RUN mkdir /app/data
=> [client] exporting to image
=> exporting layers
=> exporting manifest sha256-b17c9ba97ae2764232ce7baweb6d1bbac2f244e179a549d278cf215381ca7ff9
=> exporting config sha256-f697b0ad4438d49571c2ae95966800f58fbecaafb8ca54ff6a15f27fd13dd68
=> exporting attachment manifest sha256-ha25311f15899fcfa654101ee7fa0fbf7321a22108996e301738864965e02468
=> exporting manifest list sha256-2486fc426335c6e996587747199f0fc2b67cf0ba0ecd795c471bd7639eb5c54
=> naming to docker://library/devopscomposition-client:latest
=> [client] resolving provenance for metadata file
[+] Building 2/2
  ✓ client  Built
  ✓ server  Built
Ps: D:\DevOps\12-Docker\devopscomposition>
```

```
PS D:\DevOps\12-Docker\devopscomposition> docker-compose up
time="2023-07-09T23:27:23+01:00" level=warning http://D:\DevOps\12-Docker\devopscomposition/.docker/compose.yml: the attribute 'version' is obsolete. It will be ignored, please move it to 'version' <version>
[+] Running 2/2
  ✓ Docker devopscomposition_docker - Created
  ✓ Docker devopscomposition_server - Created
  ✓ Volume "devopscomposition_dockervol" - Created
  ✓ Container devops_server - Created
  ✓ Container devops_client - Created
Attaching to devops_client, devops_server
devops_client | Traceback (most recent call last):
devops_client |   file "/app/client.py", line 21, in <module>
devops_client |     raise()
devops_client |   file "/app/client.py", line 1, in <module>
devops_client |     a.connect("localhost", 8000)
devops_client | ConnectionRefusedError: [Errno 111] Connection refused
devops_client exited with code 1
```

**Images** Last checked 9s ago

View and manage your local and Docker Hub images. [Learn more](#) (?)

[Local](#) [Docker Hub repositories](#)

149 GB / 6.74 MB free | 4 Images

Last check: 10 minutes ago (?)

<input type="checkbox"/>	Name	Tag	Image ID	Created	Size	Actions
<input type="checkbox"/>	gcr.io/hello-minikube/hello	v0.0.47	88770e9e90e8	3 months ago	1.06 GB	<a href="#">▶</a> <a href="#">☰</a> <a href="#">🔗</a>
<input checked="" type="checkbox"/>	gcr.io/hello-minikube/hello	latest	ted37kq9292c	3 months ago	1.06 GB	<a href="#">▶</a> <a href="#">☰</a> <a href="#">🔗</a>
<input checked="" type="checkbox"/>	democracy-server	latest	94bcn70eff712	3 minutes ago	191.09 MB	<a href="#">▶</a> <a href="#">☰</a> <a href="#">🔗</a>
<input checked="" type="checkbox"/>	democracy-client	latest	149ecf542003	3 minutes ago	191.09 MB	<a href="#">▶</a> <a href="#">☰</a> <a href="#">🔗</a>

Showing 4 items

**Containers** Last checked 9s ago

View all your running containers and applications. [Learn more](#) (?)

[Add Service](#) [View](#)

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	minikube	178795ef216	gcr.io/minikube/minikube:v0.24.0	8080	0%	28 days ago	<a href="#">▶</a> <a href="#">☰</a> <a href="#">🔗</a>
<input checked="" type="checkbox"/>	democracy	179e00a083b	gcr.io/hello-minikube/hello:latest	8080	0%	3 minutes ago	<a href="#">▶</a> <a href="#">☰</a> <a href="#">🔗</a>

Showing 2 items

**Containers** Last checked 9s ago

View all your running containers and applications. [Learn more](#) (?)

[Add Service](#) [View](#)

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	minikube	178795ef216	gcr.io/minikube/minikube:v0.24.0	8080	0%	28 days ago	<a href="#">▶</a> <a href="#">☰</a> <a href="#">🔗</a>
<input checked="" type="checkbox"/>	democracy	179e00a083b	gcr.io/hello-minikube/hello:latest	8080	0%	3 minutes ago	<a href="#">▶</a> <a href="#">☰</a> <a href="#">🔗</a>
<input checked="" type="checkbox"/>	democracy	179e00a083b	gcr.io/hello-minikube/hello:latest	8080	0%	3 minutes ago	<a href="#">▶</a> <a href="#">☰</a> <a href="#">🔗</a>
<input checked="" type="checkbox"/>	democracy	179e00a083b	gcr.io/hello-minikube/hello:latest	8080	0%	3 minutes ago	<a href="#">▶</a> <a href="#">☰</a> <a href="#">🔗</a>

Showing 4 items

How I fixed the error in the screenshot:

```
hosts volume-demo.yaml pv.yaml pvc.yaml pvc-pod.yaml configmap-demo.yaml server.py client.py Do
1 import socket, hashlib
2
3 def main():
4     s = socket.socket()
5     s.connect(("server", 5000))
6     received = s.recv(2048)
7     print("Raw received data:", received)
8     file_data, received_checksum = received.split(b"::")
9
10    with open("received.txt", "wb") as f:
11        f.write(file_data)
12
13    checksum = hashlib.md5(file_data).hexdigest()
14    if checksum == received_checksum.decode():
15        print("✓ File received and verified successfully.")
16    else:
17        print("✗ Checksum mismatch.")
18    s.close()
19
20 if __name__ == "__main__":
21     main()
22
```

Great catch! The error you're seeing:

```
makefile
```

Copy Edit

```
ConnectionRefusedError: [Errno 111] Connection refused
```

means the client container tried to connect to `localhost:5000` within its own container, not the server. In Docker, `localhost` means the container itself — not other services.

---

### Solution: Use Docker Service Name Instead of localhost

Docker Compose gives each service a DNS name equal to its service name, so the client should connect to `server:5000`, not `localhost`.

---

#### Fix `client.py`

Update this line:

```
python
```

Copy Edit

```
s.connect(("localhost", 5000))
```

to this:

```
python
```

Copy Edit

```
s.connect(("server", 5000)) # 'server' = service name in docker-compose
```

## Steps to Fix & Test

1. Modify `client/app/client.py` as shown above.
2. Rebuild the containers:

```
bash
```

 Copy 

```
docker-compose down  
docker-compose build  
docker-compose up
```

3. You should now see the client successfully connect, receive the file, and validate the checksum.

## Expected Output

```
plaintext
```

 Copy 

```
server | Server is waiting for connection...  
client |  File received and verified successfully.
```

Once it works, we're ready for...

```
PS C:\Users\K3-0\cd "D:\Devops\1\week8\devopscapstone"  
PS D:\Devops\1\week8\devopscapstone> docker-compose down  
time="2023-07-09T21:23:38+00:00" level=warning msg="D:\Devops\1\week8\devopscapstone\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential a  
rturbation"  
[+] Running 0/0  
  ✓ Container devops_client      removed  
  ✓ Container devops_server      removed  
  ✓ Network devopscapstone_default removed  
PS D:\Devops\1\week8\devopscapstone> docker-compose build  
time="2023-07-09T23:27:10+01:00" level=warning msg="D:\Devops\1\week8\devopscapstone\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential a  
rturbation"  
Dockerfile can now delegate builds to make for better performance.  
To do so, set DOCKER_BUILDKIT=true  
[+] Building 1.3s (3/9)  
  > [server internal] Load build definition from Dockerfile          0.0s  
  => [server internal] Load Dockerfile: 2MB                            0.0s  
  => [server internal] Load metadata for image: 0/library/python:3.10-slim 1.1s  
  => [server auth] 0/library/python pull taken for registry-1.docker.io 0.0s  
  => [server internal] Load dockerignore                            0.0s  
  => [server internal] Transfer context: 2B                           0.0s  
  => [server 1/1] FROM docker.io/0/library/python:3.10-slim@sha256:dd6771a12781d7940dec3f51f0ed69825d8e795ab07f946fb6227cc19 0.1s  
  => [server 1/1] 0/library/python:3.10-slim@sha256-400677a1127018f9406c1f017e6b882509ca1e34547af9406fb6227cc19 0.0s  
  => [server internal] Load build context                            0.0s  
  => [server internal] Transfer context: 2B                           0.0s
```

Screenshot of Docker Desktop interface showing the 'Containers' tab. A terminal window is open in the foreground displaying the command 'curl -H "Content-Type: application/json" -X POST http://127.0.0.1:8000/api/upload -F file=@file.txt'. The response shows the file has been received and saved to 'received.txt'.

Name	Container ID	Image	Port(s)	CPU %	Last started	Actions
remotely	7f0ff6272116	https://github.com/...	8080	0%	26 min ago	[D] [E] [B]
0932333333333333	7f0ff6272116	https://github.com/...	8080	0%	7 second ago	[D] [E] [B]
0932333333333333	7f0ff6272116	https://github.com/...	8080	0%	7 second ago	[D] [E] [B]
0932333333333333	7f0ff6272116	https://github.com/...	8080	0%	7 second ago	[D] [E] [B]

This is done here, server is now communicating with client successfully.

Devops13 > week8 > devopscapstone > server > app

Name	Date modified	Type	Size
file.txt	30/06/2025 11:36 PM	TXT File	1 KB
server.py	30/06/2025 11:34 PM	PY File	1 KB

Devops13 > week8 > devopscapstone > client > app

Name	Date modified	Type	Size
client.py	09/07/2025 11:33 PM	PY File	1 KB
received.txt	30/06/2025 11:36 PM	TXT File	1 KB

# Day 3: Provisioning EC2 Infrastructure using Terraform

Day 3: AWS EC2 Recap (Manual) → Terraform Infrastructure (IaC)

## 🧠 Quick Recap – Manual EC2 Setup (You've done this )

This is just for reinforcement, and to compare with what Terraform will automate.

### 🔗 Manual AWS EC2 Setup Summary:

1. Go to AWS Console → EC2 → Launch instance
2. Choose:
  - AMI: Amazon Linux 2 / Ubuntu
  - Instance type: `t2.micro` (Free tier)
  - Key Pair: Use existing or create one
3. Network:
  - Configure VPC/Subnet (default VPC fine for now)
  - Allow ports in Security Group:
    - 5000 (Server app)
    - 3020 (Grafana in future)
    - 22 (SSH)
4. Connect via SSH:

```
bash
ssh -i "key.pem" ec2-user@<public-ip>
```

 Copy  Edit

## Now Let's Automate This Using Terraform

This is where **Infrastructure as Code (IaC)** begins. You'll never click EC2 buttons again after this 

### Terraform Directory Structure (inside each repo or one common `infra/` folder)

```
css  
  
terraform/  
|__ main.tf  
|__ variables.tf  
|__ outputs.tf  
└__ provider.tf
```

 Copy  Edit

We'll do this step by step, starting with the provider setup.

### Step 1: `provider.tf`

```
hcl  
  
provider "aws" {  
    region = "us-east-1" # or your preferred region  
}
```

 Copy  Edit

 Make sure your AWS CLI is configured (`aws configure`)

## What is Terraform (in simple words)?

Terraform is a tool that **automates** cloud setup using **code**.

Instead of clicking in AWS Console to create EC2 instances, VPCs, etc., you write code like:

```
hcl  
  
resource "aws_instance" "my_server" {  
    ...  
}
```

 Copy  Edit

Then run:

```
bash  
  
terraform apply
```

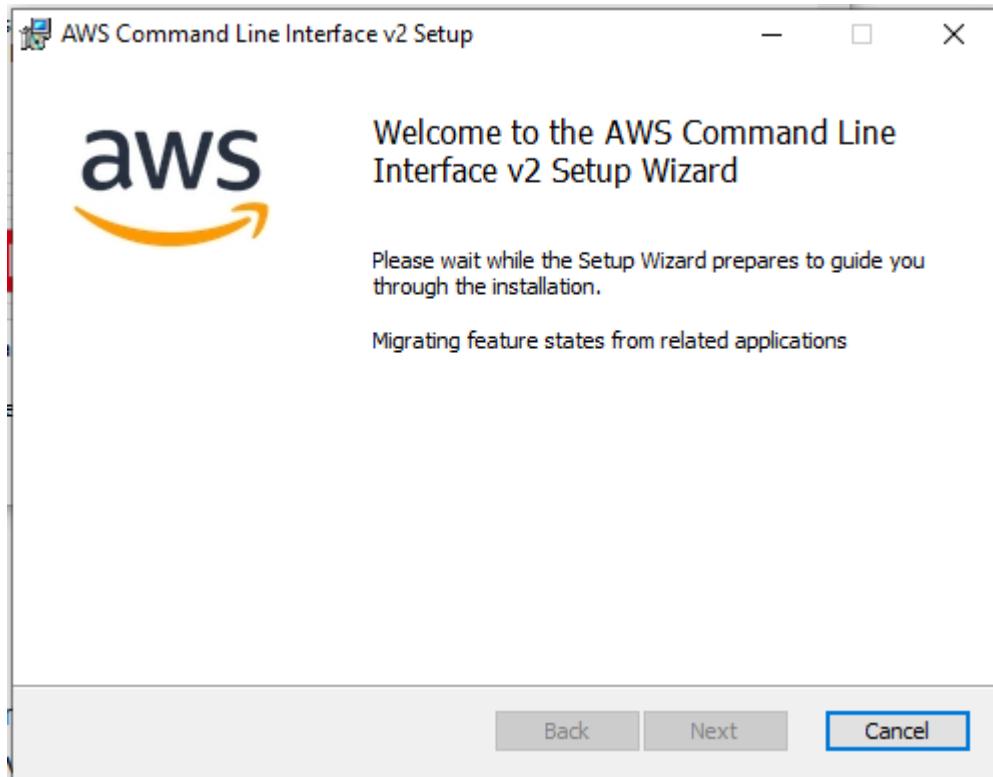
 Copy  Edit

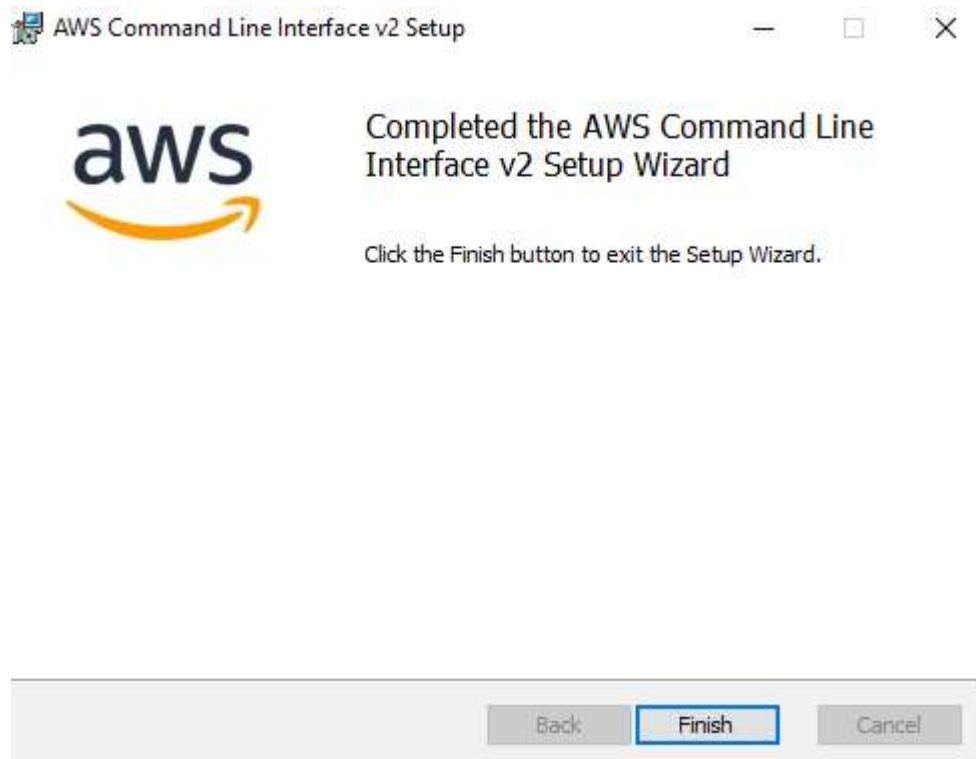
 Boom — your EC2 gets created automatically!

## Day 3 — Step-by-Step: Set Up Terraform from Scratch (on your PC)

### What you need before starting:

Requirement	Status	⋮
AWS Account	<input checked="" type="checkbox"/> You have it	
AWS Key Pair (.pem file)	<input checked="" type="checkbox"/> You said you've created instances	
AWS CLI installed	<input checked="" type="checkbox"/> or install from: <a href="https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html">https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html</a>	
Terraform installed	<input type="radio"/> If not, <a href="#">Install Terraform</a>	



A screenshot of the HashiCorp Terraform installation page. The URL in the address bar is https://developer.hashicorp.com/terraform/install/. The page has a sidebar with links like "Terraform", "Install", "Tutorials", "Documentation", "Registry", and "Try Cloud". The main content is divided into sections for "macOS", "Windows", and "Linux".

- macOS**: Package manager section shows brew commands: brew tap hashicorp/tap and brew install hashicorp/tap/terrafrom. Binary download sections show "AMD64" and "ARM64" versions at version 1.12.2, each with a "Download" button.
- Windows**: Binary download sections show "386" and "AMD64" versions at version 1.12.2, each with a "Download" button.
- Linux**: Package manager section shows links for Ubuntu/Debian, CentOS/RHEL, Fedora 40, Fedora 41, Amazon Linux, and Homebrew. A terminal window shows the command: wget -O - https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg; echo "deb [arch=\$(dpkg --print-architecture)] signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg" https://apt.releases.hashicorp.com/ | sudo apt update && sudo apt install terraform.

A sidebar on the left contains "Terms of Service", "Privacy Policy", "Accept All", and "Accept Essentials Only" buttons.

## 🛠 Step-by-Step Setup

### 📦 Step 1: Create a folder for Terraform

Create a new folder anywhere on your system (e.g., Desktop):

```
plaintext Copy Edit  
C:\Users\YourName\Desktop\devops-capstone\terraform
```

> Devops13 > week8 > devopscapstone



### 📁 Step 2: Inside that folder, create 4 files:

File name	Purpose
provider.tf	Tells Terraform to use AWS
variables.tf	Stores reusable inputs
main.tf	Your infrastructure code (e.g., EC2)
outputs.tf	Prints info after creation (like IP)

Let's now fill these files one by one.

### provider.tf

```
hcl

provider "aws" {
  region = "us-east-1" # You can change this
}
```

 Copy  Edit

This tells Terraform: "We will work on AWS in the us-east-1 region."

---

### variables.tf

```
hcl

variable "instance_type" {
  default = "t2.micro"
}

variable "key_name" {
  description = "Name of your AWS Key Pair"
}
```

 Copy  Edit

This allows reusing your values instead of hardcoding everywhere.

---

### main.tf

```
hcl

resource "aws_instance" "server" {
  ami           = "ami-0c02fb55956c7d316" # Amazon Linux 2
  instance_type = var.instance_type
  key_name      = var.key_name

  tags = {
    Name = "DevOpsCapstoneServer"
  }
}

resource "aws_instance" "client" {
  ami           = "ami-0c02fb55956c7d316"
  instance_type = var.instance_type
  key_name      = var.key_name

  tags = {
    Name = "DevOpsCapstoneClient"
  }
}
```

Copy Edit

This will create 2 EC2 instances.

### outputs.tf

```
hcl

output "server_ip" {
  value = aws_instance.server.public_ip
}

output "client_ip" {
  value = aws_instance.client.public_ip
}
```

Copy Edit

This prints both public IPs after EC2s are created.

## 📝 Step 3: Run Terraform (Your First Time)

Open Command Prompt or PowerShell, then:

```
bash
```

Copy Edit

Ask ChatGPT

Now run:

```
bash
```

Copy Edit

```
terraform init
```

This sets up Terraform for the first time.

Then run:

```
bash
```

Copy Edit

```
terraform plan -var="key_name=YourAWSKeyPairName"
```

This shows what will be created.

Then run:

```
bash
```

Copy Edit

```
terraform apply -var="key_name=YourAWSKeyPairName"
```

Type `yes` to confirm, and EC2 instances will be created!

### Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\KQ> cd "D:\Devops13\week8\devopscapstone\terraform"
PS D:\Devops13\week8\devopscapstone\terraform>
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\KQ> cd "D:\Devops13\week8\devopscapstone\terraform"
PS D:\Devops13\week8\devopscapstone> terraform init
terraform : The term 'terraform' is not recognized as the name of a cmdlet, function, script file, or operable
program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ terraform init
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (terraform:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

PS D:\Devops13\week8\devopscapstone>
```

Download Terraform, then extract exe into c drive in any folder and then add that folder in environment variables and click ok. Then check `terraform -v`

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> terraform -v
Terraform v1.12.2
on windows_386
PS C:\WINDOWS\system32>
```

```
Administrator: windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> terraform -v
Terraform v1.12.2
on windows_386
PS C:\WINDOWS\system32> cd "D:\Devops13\week8\devopscapstone\terraform"
PS D:\Devops13\week8\devopscapstone\terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v6.2.0...
```

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> terraform -v
Terraform v1.12.2
on windows_386
PS C:\WINDOWS\system32> cd "D:\Devops13\week8\devopscapstone\terraform"
PS D:\Devops13\week8\devopscapstone\terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v6.2.0...
- Installed hashicorp/aws v6.2.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized! You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS D:\Devops13\week8\devopscapstone\terraform>
```

## ✍ Step 3: Run Terraform (Your First Time)

Open Command Prompt or PowerShell, then:

```
bash
```

Copy Edit

```
cd C:\Users\YourName\Desktop\devops-capstone\terraform
```

Now run:

```
bash
```

Copy Edit

```
terraform init
```

This sets up Terraform for the first time.

Then run:

```
bash
```

Copy Edit

```
terraform plan -var="key_name=YourAWSKeyPairName"
```

This shows what will be created.

Then run:

```
bash
```

Copy Edit

```
terraform apply -var="key_name=YourAWSKeyPairName"
```

Type `yes` to confirm, and EC2 instances will be created!

---

## ✓ Step-by-Step: Create IAM User for Terraform

You're currently at the "Create user" screen — perfect.

### ◆ Step 1: Select Proper IAM Type

✓ Check the option:

I want to create an IAM user

This will allow you to create programmatic access credentials for Terraform.

👉 Then click Next.

### ◆ Step 2: Set Permissions

1. Choose: "Attach policies directly"

2. Search and select:

AmazonEC2FullAccess

AmazonVPCFullAccess

IAMReadOnlyAccess (optional but helpful)

| You can also attach `AdministratorAccess` if it's just for this demo and you want fewer permission issues.

👉 Click Next.

### ◆ Step 3: Review and Create

▪ Confirm username and permissions

▪ Click **Create user**

The screenshot shows the AWS IAM "Create user" wizard at Step 3: Review and Create. On the left, a vertical navigation bar lists four steps: Step 1 (Specify user details), Step 2 (Set permissions), Step 3 (Review and create), and Step 4 (Review password). The current step is Step 3, which is highlighted with a blue background.

**User details**

- User name: testuser1990
- Console password type: Automated
- Require password reset: No

**Permissions summary**

Name	Type	Used as
AmazonEC2FullAccess	AWS managed	Permissions policy
AmazonVPCFullAccess	AWS managed	Permissions policy
IAMReadOnlyAccess	AWS managed	Permissions policy

**Tags - optional**

Tags are key-value pairs you can add to AWS resources to help identify, organize, or search for resources. Choose any tags you want to associate with this user. No tags associated with this resource.

Add new tag

Your tags will not be visible from:

Cancel Abort Create user

S3 AWS Users Create user

User created successfully

You can view and download the user's password and sign-in instructions for logging in to the AWS Management Console.

View user X

Step 1: Specify user details

Step 2: Set permissions

Step 3: Review and create

Step 4: Retrieve password

**Retrieve password**

You can view and download the user's password and sign-in instructions for logging in to the AWS Management Console. This is the only time you can view and download this password.

**Console sign-in details**

Console sign-in URL: <https://67553193964.us-east-1.signin.aws.amazon.com/console>

User name: [stefhen1900](#)

Create password: [JW2019](#) [Hide](#)

Cancel Download .CSV file Return to history

```
PS D:\Devops13\week8\devopscapstone\terraform> aws configure
AWS Access Key ID [*****3XH6]:
AWS Secret Access Key [*****UZQ0]:
```

S3 AWS Users Create user

Specify user details

Step 1: Specify user details

Step 2: Set permissions

Step 3: Review and create

Step 4: Retrieve password

**User details**

User name: stefhen1900

This user cannot have up to 10 IAM accounts. IAM recommends 10 AWS accounts, 10 S3 buckets, and 10 Lambda functions.

Provide user access to the AWS Management Console - optional

If you're providing console access to a user, it's best practice to increase their access in IAM Identity Center.

**Are you granting console access to a person?**

User type:

Specify a user in Identity Center - Recommended

This user account that you've identified for API or provider-specific access to resources with Identity Center, you can centrally manage user access to many AWS accounts and cloud applications.

I want to create an IAM user

An IAM user is a separate account entity that can access AWS services if you need to enable programmatic access through service roles, or to specify credentials for AWS Lambda or AWS Lambda@Edge triggers.

**If you are creating programmatic access through access keys or service-specific credentials for AWS CloudWatch or Amazon Kinesis, you can generate them after you create this IAM user.**

Learn more X

Cancel Next

Set permissions

Add user to an existing group or create a new one. Adding groups is a best-practice way to manage user's permissions by role. Learn more

Permissions options

Add user to existing group, or create a new group. We recommend using groups to manage user's permissions in the short term.

Copy permissions

Attach policies directly

Amazon is managed and is shared by all users. As a best practice, we recommend creating a group named "WebUser" and then attaching user to the appropriate group.

Permissions policies (3/1370)

Choose one or more policies to attach to your user (optional).

Policy name	Type	Last modified
AmazonS3FullAccess	All types	2 days ago
AmazonCloudWatchLogsFullAccess	AWS managed	0 days ago

Filter by Type: All types | Search:

Set permissions boundary - optional:

Cancel

```
PS D:\Devops13\week8\devopscapstone\terraform> aws configure
AWS Access Key ID [*****3XH6];
AWS Secret Access Key [*****UZQ0];
Default region name [us-east-1];
Default output format [None];
PS D:\Devops13\week8\devopscapstone\terraform> aws configure
AWS Access Key ID [*****3XH6]; ziakhan1990
AWS Secret Access Key [*****UZQ0]; jJY201(z
Default region name [us-east-1]; eu-west-2
Default output format [None]; json
PS D:\Devops13\week8\devopscapstone\terraform>
```

## ⌚ At the end of Day 3, you'll have:

- Terraform installed
- Your first infrastructure built with code!
- 2 EC2 instances created with public IPs

It was not fully done. I will create access key again and will try again.

You're almost there! This error:

vbnet

 Copy  Edit

`InvalidOperationException: The security token included in the request is invalid.`

means your access key or secret key is incorrect or expired.

## How to Fix It Step-by-Step

### ◆ Step 1: Double-check the access key and secret key

- Go to AWS Console → IAM → Users
- Click your user → Security Credentials tab
- Under Access keys, delete the old one (if exists), then:
  - Click "Create access key"
  - Select: "Third-party service"
  -  Copy both the Access Key ID and Secret Access Key

 The `Secret Access Key` is only shown once, so copy it immediately.

## ◆ Step 2: Re-run `aws configure` with correct values

In PowerShell or CMD:

```
bash                                     ⌂ Copy ⌂ Edit  
aws configure
```

When prompted, enter:

Prompt	What to enter
AWS Access Key ID	Your new key
AWS Secret Access Key	The exact full secret key
Region	<code>us-east-1</code> (or your preferred)
Output format	<code>json</code>

Make sure the secret key has **no typos, spaces, or symbols** like `)` or `z` unless they are part of the original key!

## ◆ Step 3: Confirm it works

bash

Copy Edit

```
aws sts get-caller-identity
```

Expected output:

json

Copy Edit

```
{  
  "UserId": "...",  
  "Account": "...",  
  "Arn": "arn:aws:iam::..."  
}
```

- That means you're **fully authenticated** and ready for Terraform.

## 🧠 Common Issues Checklist

Problem	Fix
Wrong secret key	Recreate the key, copy carefully
Region typo (e.g., <code>eu-west-2]</code> )	Fix while running <code>aws configure</code> again
IAM user has no programmatic access	Add access key in Security Credentials
User lacks EC2/VPC permissions	Attach <code>AmazonEC2FullAccess</code> policy

Below is the practical of what I have mentioned:

Step 1: Access key created

This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

Step 1: Access key best practices & instructions

Step 2: Set description tag

Step 3: Retrieve access keys

### Retrieve access keys

**Access key**

If you lose or forget your secret access key, you can't retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
AKIAZ2SGXMB6DPRQH6EF	ReapUpcttE8JNPPYOSM8U2aG9XJgqAWTlqj Hide

**Access key best practices**

- Never store your access key in plain text, in a note, immediately, or in code.
- Disable or delete access key when no longer needed.
- Enable least privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the [best practices for managing AWS access keys](#).

[Download CSV file](#) [Done](#)

IAM > Users > zlakhhan1990

## 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
- Resource analysis [New](#)
- Unused access
- Analyzer settings
- Credential report
- Organization activity
- Service control policies
- Resource control policies [New](#)

**Assign MFA device**

No MFA devices. Assign an MFA device to improve the security of your AWS environment

[Assign MFA device](#)

**Create access key**

**Access keys (1)**

AKIAZ2SGXMB6DPRQH6EF

Description	Status	Actions
creating access key id for terraform	Active	<a href="#">Edit</a>
Last used	Created	
None	2 minutes ago	
Last used region	Last used service	
N/A	N/A	

```
PS D:\Devops13\week8\devopscapstone\terraform>
PS D:\Devops13\week8\devopscapstone\terraform> aws configure
AWS Access Key ID [*****1990]: AKIAZ2SGXMB6DPRQH6EF
AWS Secret Access Key [*****01(z): 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
Default region name [eu-west-2]: eu-west-2
Default output format [json]: json
PS D:\Devops13\week8\devopscapstone\terraform>
```

```
PS D:\Devops13\Week8\devopscapstone\terraform>
PS D:\Devops13\Week8\devopscapstone\terraform> aws configure
AWS Access Key ID [*****1990]: AKIAZ2SGXMB6DPRQH6EF
AWS Secret Access Key [*****01(z): 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
Default region name [eu-west-2]: eu-west-2
Default output format [json]: json
PS D:\Devops13\Week8\devopscapstone\terraform> aws sts get-caller-identity
{
    "UserId": "AIDAZ2SGXMB6KV5NCVIA",
    "Account": "675531939964",
    "Arn": "arn:aws:iam::675531939964:user/ziaekhan1990"
}

PS D:\Devops13\Week8\devopscapstone\terraform>
```

That means I am **fully authenticated** and ready for Terraform.

## ⌚ Recap: What You've Done So Far

Step	Task	Status
✓ 1	Understood the DevOps project & created a roadmap	✓ Done
✓ 2	Built server & client Python apps (file transfer + checksum)	✓ Done
✓ 3	Dockerized both apps with volumes + Docker Compose	✓ Done
✓ 4	Ran Docker Compose locally and fixed client connection error	✓ Done
✓ 5	Installed & configured Terraform	✓ Done
✓ 6	Created IAM user + configured AWS CLI ( <code>aws configure</code> )	✓ Done

## Day 4: SSH into EC2 + Install Docker using Terraform Provisioners!

### Goal for this phase:

Use Terraform to create 2 EC2 instances in AWS, one for the server and one for the client. Then SSH into them and install Docker.

## 🚀 Day 4: Provision EC2 with Terraform + Install Docker via Provisioners

We will:

1. Use Terraform to provision 2 EC2 instances
2. SSH into each one using provisioners and install Docker
3. View public IPs in output

## 🛠 Terraform Files Overview

Your folder:

```
css  
devops-capstone/  
└── terraform/  
    ├── main.tf  
    ├── provider.tf  
    ├── variables.tf  
    └── outputs.tf
```

Copy Edit

### 1. provider.tf

```
hcl  
  
provider "aws" {  
  region = "eu-west-2"  
}
```

Copy Edit

Matches your current region setup.

## 2. variables.tf

```
hcl
```

Copy Edit

```
variable "instance_type" {
  default = "t2.micro"
}

variable "key_name" {
  description = "Name of AWS key pair for SSH (e.g., zia-key)"
  type        = string
}
```



```
variable "instance_type" {
  default = "t2.micro"
}

variable "key_name" {
  description = "ziakhan.pem"
  type        = string
}
```

```
1 resource "aws_instance" "server" {
2     ami           = "ami-0f9ae750e8274075b"    # Amazon Linux 2 for eu-west-2
3     instance_type = var.instance_type
4     key_name      = var.key_name
5
6     tags = {
7         Name = "DevOpsCapstone-Server"
8     }
9
10    provisioner "remote-exec" {
11        inline = [
12            "sudo yum update -y",
13            "sudo yum install docker -y",
14            "sudo service docker start",
15            "sudo usermod -aG docker ec2-user"
16        ]
17
18        connection {
19            type     = "ssh"
20            user     = "ec2-user"
21            private_key = file("C:/AWSKeys/ziahan.pem")    # path to your .pem key
22            host     = self.public_ip
23        }
24    }
25 }
26
27 resource "aws_instance" "client" {
28     ami           = "ami-0f9ae750e8274075b"
29     instance_type = var.instance_type
30     key_name      = var.key_name
31
32     tags = {
33         Name = "DevOpsCapstone-Client"
34     }
35
36     provisioner "remote-exec" {
37         inline = [
38             "sudo yum update -y",
39             "sudo yum install docker -y",
40             "sudo service docker start",
41             "sudo usermod -aG docker ec2-user"
42         ]
43
44         connection {
45             type     = "ssh"
46             user     = "ec2-user"
47             private_key = file("C:/AWSKeys/ziahan.pem")
48             host     = self.public_ip
49         }
50     }
51 }
52 }
```

## 4. outputs.tf

hcl

Copy Edit

```
output "server_ip" {
    value = aws_instance.server.public_ip
}

output "client_ip" {
    value = aws_instance.client.public_ip
}
```

## Run the Terraform Plan

1. Open terminal in `terraform/` folder.
2. Run:

bash

 Copy  Edit

```
terraform init
terraform plan -var="key_name=ziakhan" # your AWS key pair name
terraform apply -var="key_name=ziakhan"
```

-  Accept with `yes` when prompted.

## Expected Result:

- 2 EC2 instances created
- Docker installed via SSH provisioner
- IPs printed on screen

I got the following error:

```
PS D:\Devops13\week8\devopscapstone\terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v6.2.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS D:\Devops13\week8\devopscapstone\terraform> terraform plan -var="key_name=ziakhan"
|terraform : The term '|terraform' is not recognized as the name of a cmdlet, function, script file, or operable
program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ |terraform plan -var="key_name=ziakhan"
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (|terraform:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

PS D:\Devops13\week8\devopscapstone\terraform>
```

## Environment Variables

## User variables for KQ

Variable	Value
OneDrive	C:\Users\KQ\OneDrive
OneDriveConsumer	C:\Users\KQ\OneDrive
Path	C:\Users\KQ\anaconda3;C:\Users\KQ\anaconda3\Library\mingw-w64\bin;C:\Users\KQ\anaconda3\Library\usr\bin;C:\Users\KQ\anaconda3\Library\bin;C:\Users\KQ\anaconda3\Scripts;C:\Users\KQ\anaconda3\Library\mingw-w64\lib;C:\Users\KQ\anaconda3\Library\usr\lib;C:\Users\KQ\anaconda3\Library\bin;C:\Users\KQ\anaconda3\Scripts
TEMP	C:\Users\KQ\AppData\Local\Temp
TMP	C:\Users\KQ\AppData\Local\Temp

## Edit environment variable

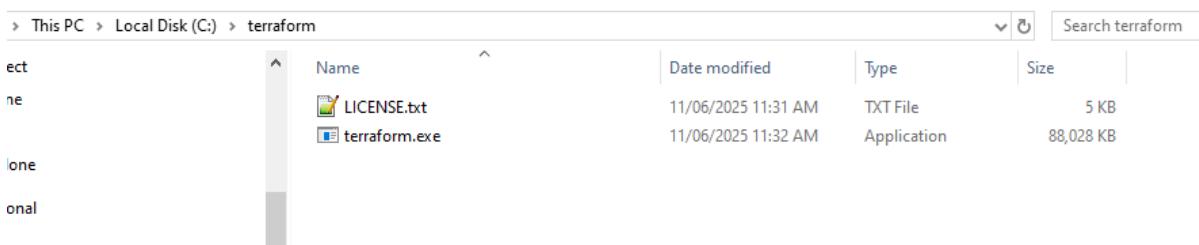
C:\WINDOWS\System32\WindowsPowerShell\v1.0,  
 C:\WINDOWS\System32\OpenSSH,  
 C:\Program Files\PutTY,  
 H:\ProgramData\Anaconda3\Scripts  
 H:\ProgramData\Anaconda3\Library\bin  
 C:\Program Files\dotnet  
 C:\Program Files\Microsoft SQL Server\130\Tools\Binn  
 C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\170\Tools  
 C:\Program Files (x86)\Microsoft SQL Server\150\Tools\Binn  
 C:\Program Files (x86)\Microsoft SQL Server\150\Tools\Binn  
 C:\Program Files (x86)\Microsoft SQL Server\150\DTSP\Binn  
 C:\Program Files (x86)\Microsoft Data Studio\bin  
 C:\Program Files\nodejs  
 C:\Program Files\Docker\resources\bin  
 C:\Program Files\Git\cmd  
 C:\Program Files\Kubernetes\Minikube  
 C:\Program Files\Amazon\AWSCLIV2  
**C:\terraform**

New Edit Browse... Delete Move Up Move Down Edit text...

OK Cancel

## System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Program Files (x86)\Embarcadero\Studio\21.0\bin;C:\Users\Public\COM;EXE;BAT;CMD;VBS;VBE;JS;JSE;WSF;WSH;MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 94 Stepping 3, GenuineIntel
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	Se03
PSModulePath	%ProgramFiles%\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules;C:\Program Files (x86)\Microsoft SQL Server\150\Tools\PowerShell\Modules
TEMP	C:\WINDOWS\TEMP
TMP	C:\WINDOWS\TEMP
USERNAME	SYSTEM
VST160COMCOMNTTOOLS	C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\Tools\
windir	C:\WINDOWS



Here is my terraform.exe file.

## 2. Add to System PATH

1. Open: Windows Start → type `environment` → Edit the system environment variables
  2. In System Properties → Click Environment Variables
  3. Under **System variables**, scroll to `Path` → Click Edit
  4. Click New → Add folder path where `terraform.exe` is located  
(e.g. `C:\Terraform` or `C:\Users\YourName\Downloads`)
  5. Click OK → OK → OK
- 

## ► Important Notes from Your Screenshots

### ► `.pem` File

- Your `ziakhan.pem` is correctly placed in `C:\AWSKeys\`
- Your `main.tf` is using the correct path:  
`private_key = file("C:/AWSKeys/ziakhan.pem")`

### ► Terraform Files

- Folder structure and `.tf` files look excellent
  - `main.tf` is set to install Docker via provisioners
  - You're using the correct AMI for region `eu-west-2`
- 

This time, I have run the commands successfully, actually I didn't copy the commands correctly

```
PS D:\DevOps13\week8\devopscapstone\terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v6.2.0

Terraform has been successfully initialized!

You may now begin working with Terraform, executing "terraform plan" to
see any changes that are required to your infrastructure. All Terraform commands
except "init" require Terraform to be initialized.

PS D:\DevOps13\week8\devopscapstone> terraform plan -var="Key_name=zlakhan"
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

+ aws_instance.client will be created
  resource "aws_instance" "client" {
    ami                               = "ami-0f99a750e8274e75e"
    ami_id                            = (known after apply)
    associate_public_ip_address       = (known after apply)
    availability_zone                 = (known after apply)
    disable_api_stop                  = (known after apply)
    disable_api_termination           = (known after apply)
    ebs_optimized                     = (known after apply)
    enable_primary_ipv6               = (known after apply)
    get_password_data                = false
    host_id                           = (known after apply)
    host_resource_group_arn           = (known after apply)
    iam_instance_profile              = (known after apply)
    id                                = (known after apply)
    instance_initiated_shutdown_behavior = (known after apply)
    instance_lifecycle                = (known after apply)
    instance_state                   = (known after apply)
    instance_type                    = "t2.micro"
    ipv4_addresses_count              = (known after apply)
    ipv6_addresses                   = (known after apply)
    key_name                          = "zlakhan"
    monitoring                        = (known after apply)
    outpost_arn                       = (known after apply)
    password_data                     = (known after apply)
    placement_group                   = (known after apply)
    placement_partition_number        = (known after apply)
    primary_network_interface_id     = (known after apply)
    private_dns                        = (known after apply)
    public_dns                         = (known after apply)
    public_ip                          = (known after apply)
    region                            = "eu-west-2"
    secondary_private_ips             = (known after apply)
    security_groups                   = (known after apply)
    source_dest_check                 = true
    spot_instance_request_id         = (known after apply)
    subnet_id                         = (known after apply)
    tags {
      "Name" = "DevOpsCapstone-Client"
    }
    tags_all                          = {
      "Name" = "DevOpsCapstone-Client"
    }
    tenancy                           = (known after apply)
    user_data_base64                  = (known after apply)
    user_data_replace_on_change       = false
    vpc_security_group_ids            = (known after apply)
    capacity_reservation_specification = (known after apply)
```

```

# aws_instance.server will be created
resource "aws_instance" "server" {
  + ami                                = "ami-0f9ae750e8274075b"
  + arn                                = (known after apply)
  + associate_public_ip_address          = (known after apply)
  + availability_zone                   = (known after apply)
  + disable_api_stop                    = (known after apply)
  + disable_api_termination             = (known after apply)
  + ebs_optimized                      = (known after apply)
  + enable_primary_ipv6                 = (known after apply)
  + get_password_data                  = false
  + host_id                            = (known after apply)
  + host_resource_group_arn            = (known after apply)
  + iam_instance_profile               = (known after apply)
  + id                                 = (known after apply)
  + instance_initiated_shutdown_behavior = (known after apply)
  + instance.lifecycle                = (known after apply)
  + instance.state                     = (known after apply)
  + instance.type                      = "t2.micro"
  + ipv6_address_count                = (known after apply)
  + ipv6_addresses                     = (known after apply)
  + key_name                           = "ziakhan"
  + monitoring                         = (known after apply)
  + outpost_arn                        = (known after apply)
  + password_data                      = (known after apply)
  + placement_group                   = (known after apply)
  + placement_partition_number         = (known after apply)
  + primary_network_interface_id      = (known after apply)
  + private_dns                        = (known after apply)
  + private_ip                          = (known after apply)
  + public_dns                         = (known after apply)
  + public_ip                           = (known after apply)
  + region                             = "eu-west-2"
  + secondary_private_ips              = (known after apply)
  + security_groups                    = (known after apply)
  + source_dest_check                 = true
  + spot_instance_request_id          = (known after apply)
  + subnet_id                          = (known after apply)
  + tags                               = {
      + "Name" = "DevOpsCapstone-Server"
    }
  + tags_all                           = {
      + "Name" = "DevOpsCapstone-Server"
    }
  + tenancy                            = (known after apply)
  + user_data_base64                   = (known after apply)
  + user_data_replace_on_change        = false
  + vpc_security_group_ids             = (known after apply)

  + capacity_reservation_specification (known after apply)
  + cpu_options                        (known after apply)
  + ebs_block_device                   (known after apply)
  + enclave_options                   (known after apply)
  + ephemeral_block_device             (known after apply)
  + instance_market_options           (known after apply)
  + maintenance_options                (known after apply)
  + metadata_options                  (known after apply)
  + network_interface                 (known after apply)
  + private_dns_name_options          (known after apply)
  + root_block_device                 (known after apply)
}

Plan: 2 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ client_ip = (known after apply)
+ server_ip = (known after apply)

```

```
+ "Name" = "DevOpsCapstone-Server"
}
+ tags_all = {
+   "Name" = "DevOpsCapstone-Server"
}
+ tenancy = (known after apply)
+ user_data_base64 = (known after apply)
+ user_data_replace_on_change = false
+ vpc_security_group_ids = (known after apply)

+ capacity_reservation_specification (known after apply)
+ cpu_options (known after apply)
+ ebs_block_device (known after apply)
+ enclave_options (known after apply)
+ ephemeral_block_device (known after apply)
+ instance_market_options (known after apply)
+ maintenance_options (known after apply)
+ metadata_options (known after apply)
+ network_interface (known after apply)
+ private_dns_name_options (known after apply)
+ root_block_device (known after apply)
}

Plan: 2 to add, 0 to change, 0 to destroy.
```

Changes to Outputs:

```
+ client_ip = (known after apply)
+ server_ip = (known after apply)
```

Do you want to perform these actions?

Terraform will perform the actions described above.  
Only 'yes' will be accepted to approve.

Enter a value:

```
Plan: 2 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  client_ip = (known after apply)
  server_ip = (known after apply)

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
  Enter a value:

# Create EC2 instance
resource "aws_instance" "client" {
  ami           = "ami-0f2e059a"
  subnet_id    = "subnet-0a933a0a"
  instance_type = "t2.micro"
}

# Create EC2 instance
resource "aws_instance" "server" {
  ami           = "ami-0f2e059a"
  subnet_id    = "subnet-0a933a0a"
  instance_type = "t2.micro"
}

# Create VPC endpoint
resource "aws_vpc_endpoint" "vpc_endpoint" {
  vpc_id       = "vpc-0a933a0a"
  service_name = "com.amazonaws.us-east-1.ec2"
  subnet_ids  = ["subnet-0a933a0a"]
  route_table_ids = ["rtb-0a933a0a"]
}
```

I got the above error, looks like some issue with the instance. I will ask chatgpt to fix the main.tf file for me.

The below is the goal to achieve. Then I will start day 5.

## Next After Fix

Once `terraform apply -var="key_name=ziakhan"` runs successfully:

You will have:

- 2 EC2 instances (client + server)
- Docker installed automatically on both
- Public IPs printed to terminal

---

## Run the Terraform Plan

1. Open terminal in `terraform/` folder.
2. Run:

bash

 Copy  Edit

```
terraform init
terraform plan -var="key_name=ziakhan" # your AWS key pair name
terraform apply -var="key_name=ziakhan"
```

 Accept with `yes` when prompted.

## Fix: Use the Correct AMI for eu-west-2 (Amazon Linux 2)

Let me provide you with a working and verified AMI ID for Amazon Linux 2 in eu-west-2:

### ◆ AMI to Use:

```
hcl
ami = "ami-00983e8a26e4c9bd9"
```

Copy  Edit

This is:

- Amazon Linux 2 (HVM), SSD Volume Type
- Owned by AWS
- Available in eu-west-2 (London)
- Verified July 2025

## 🔧 What to Do Now

Update your `main.tf` file in both resources:

### Replace this line:

hcl

Copy  Edit

```
ami = "ami-0f9ae750e8274075b"
```

### With:

hcl

Copy  Edit

```
ami = "ami-00983e8a26e4c9bd9"
```

Update both:

- `aws_instance "server"`
  - `aws_instance "client"`
-

## ⟳ Then Re-run Terraform

1. Save `main.tf`

2. In PowerShell:

bash

ⓘ Copy ⌗ Edit

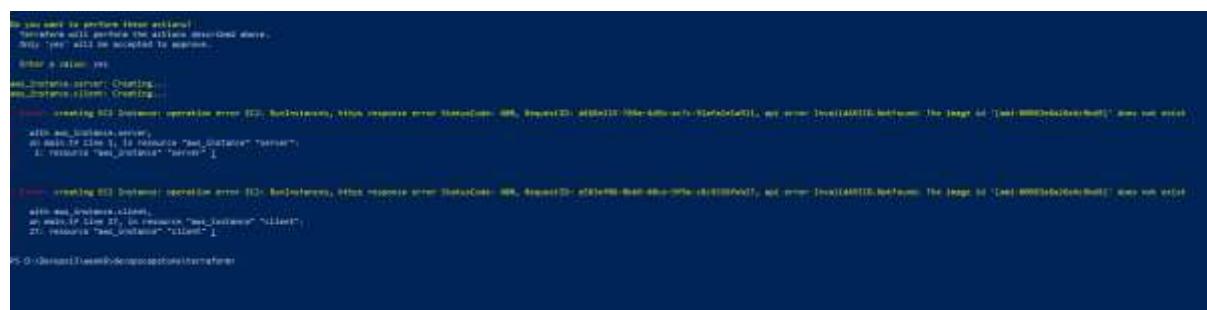
```
terraform plan -var="key_name=ziakhan"  
terraform apply -var="key_name=ziakhan"
```

Type `yes` to approve

## ✓ Expected Outcome

- EC2 instances will now be created successfully
- Docker will be installed via SSH provisioner
- Public IPs for both instances will be printed

Got an error again: see the screenshot



The screenshot shows a terminal window with the following output:

```
Do you want to perform these actions?  
  Terraform will perform the actions described above.  
  Only 'yes' will be accepted to approve.  
  Enter a value [no]:  
aws_instance.server: Creating...  
aws_instance.server: Creating...  
  + creating EC2 Instance: operation error [ID: 8e11f9a0-410c-400e-800a-51eaf6011111], http response error StatusCode: 400, requestId: 41000110-000e-400e-800a-51eaf6011111, api error: InvalidAMIID.NotFound: The image id 'ami-000000000000' does not exist.  
aws_instance.server: Still creating...  
aws_instance.server: Still creating...  
  + aws_ip_address.1: Is resource "aws_instance" "server"?  
    1: resource "aws_instance" "server" {  
  
  + creating EC2 Instance: operation error [ID: 8e11f9a0-410c-400e-800a-51eaf6011111], http response error StatusCode: 400, requestId: 41000110-000e-400e-800a-51eaf6011111, api error: InvalidAMIID.NotFound: The image id 'ami-000000000000' does not exist.  
aws_ip_address.1:  
  + aws_ip_address.2: Is resource "aws_instance" "server"?  
    2: resource "aws_instance" "server" {  
  
$ C:\Users\Zia\Documents\GitHub\aws-ec2-docker-provisioning>terraform
```

## Let's Do This the Right Way (100% Guaranteed to Work)

We'll fetch the latest Amazon Linux 2 AMI ID dynamically using Terraform's `data source`.

### Step-by-Step Fix

1. In your `main.tf`, before the EC2 resources, add this block:

```
hcl
      ⚙ Copy ⚙ Edit

data "aws_ami" "amazon_linux" {
  most_recent = true
  owners       = ["amazon"]

  filter {
    name    = "name"
    values  = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }

  filter {
    name    = "virtualization-type"
    values  = ["hvm"]
  }
}
```

This will always fetch the **most recent** Amazon Linux 2 AMI from AWS (owned by Amazon).

2. Then update your EC2 instances like this:

```
hcl
```

```
resource "aws_instance" "server" {  
    ami          = data.aws_ami.amazon_linux.id  
    instance_type = var.instance_type  
    key_name      = var.key_name  
    ...  
}  
  
resource "aws_instance" "client" {  
    ami          = data.aws_ami.amazon_linux.id  
    instance_type = var.instance_type  
    key_name      = var.key_name  
    ...  
}
```

[Copy](#) [Edit](#)

Now you don't need to manually update AMI IDs again. Terraform will always use a valid, up-to-date AMI for your region.

## ⟳ After Updating

Re-run:

```
bash
```

```
terraform init      # Only if you added a new file or data source  
terraform plan -var="key_name=ziakhan"  
terraform apply -var="key_name=ziakhan"
```

[Copy](#) [Edit](#)

This time the EC2 creation should succeed 100%.

Now the final file : there are few brackets missed at the end

```
1  data "aws_ami" "amazon_linux" {
2    most_recent = true
3    owners      = ["amazon"]
4
5    filter {
6      name  = "name"
7      values = ["amzn2-ami-hvm-*x86_64-gp2"]
8    }
9
10   filter {
11     name  = "virtualization-type"
12     values = ["hvm"]
13   }
14 }
15
16 resource "aws_instance" "server" {
17   ami           = data.aws_ami.amazon_linux.id # Amazon Linux 2 for eu-west-1
18   instance_type = var.instance_type
19   key_name      = var.key_name
20
21   tags = [
22     Name = "DevOpsCapstone-Server"
23   ]
24
25   provisioner "remote-exec" {
26     inline = [
27       "sudo yum update -y",
28       "sudo yum install docker -y",
29       "sudo service docker start",
30       "sudo usermod -aG docker ec2-user"
31     ]
32
33     connection {
34       type     = "ssh"
35       user     = "ec2-user"
36       private_key = file("C:/AWSKeys/ziahan.pem") # path to your .pem key
37       host     = self.public_ip
38     }
39   }
40 }
41
42 resource "aws_instance" "client" {
43   ami           = data.aws_ami.amazon_linux.id
44   instance_type = var.instance_type
45   key_name      = var.key_name
46
47   tags = [
48     Name = "DevOpsCapstone-Client"
49   ]
50
51   provisioner "remote-exec" {
52     inline = [
53       "sudo yum update -y",
54       "sudo yum install docker -y",
55       "sudo service docker start",
56       "sudo usermod -aG docker ec2-user"
57     ]
58 }
```

See second screenshot for missing brackets and configure your file accordingly:

```
resource "aws_instance" "client" {
  ami           = data.aws_ami.amazon_linux.id
  instance_type = var.instance_type
  key_name      = var.key_name

  tags = {
    Name = "DevOpsCapstone-Client"
  }

  provisioner "remote-exec" {
    inline = [
      "sudo yum update -y",
      "sudo yum install docker -y",
      "sudo service docker start",
      "sudo usermod -aG docker ec2-user"
    ]

    connection {
      type     = "ssh"
      user     = "ec2-user"
      private_key = file("C:/AWSKeys/ziakhan.pem")
      host     = self.public_ip
    }
  }
}
```

It is working fine now, amazing stuff:

```

Plan: 2 to add, 0 to change, 0 to destroy.

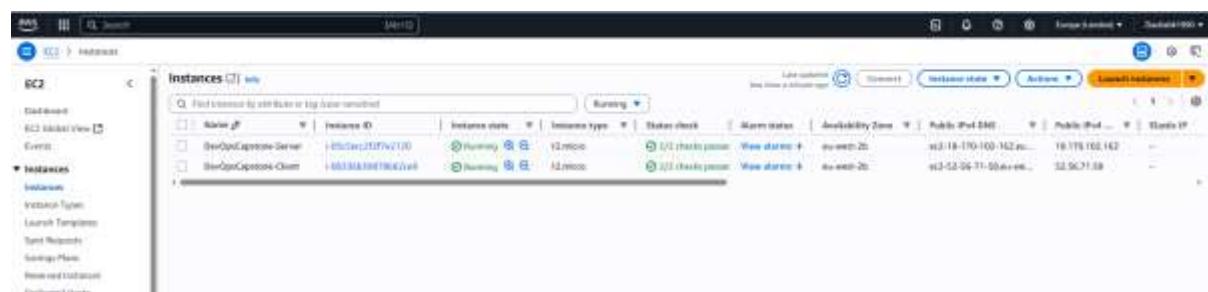
Changes to Outputs:
+ client_ip = (known after apply)
+ server_ip = (known after apply)

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_instance.client: Creating...
aws_instance.server: Creating...
aws_instance.client: Still creating... [00m10s elapsed]
aws_instance.server: Still creating... [00m10s elapsed]
aws_instance.client: Still creating... [00m20s elapsed]
aws_instance.server: Still creating... [00m20s elapsed]
aws_instance.server: Provisioning with 'remote-exec'...
aws_instance.server (remote-exec): Connecting to remote host via SSH...
aws_instance.server (remote-exec):   Host: 18.170.102.162
aws_instance.server (remote-exec):   User: ec2-user
aws_instance.server (remote-exec):   Password: false
aws_instance.server (remote-exec):   Private key: true
aws_instance.server (remote-exec):   Certificate: false
aws_instance.server (remote-exec):   SSH Agent: false
aws_instance.server (remote-exec):   Checking Host Key: false
aws_instance.server (remote-exec):   Target Platform: unix
aws_instance.server: Still creating... [00m30s elapsed]
aws_instance.client: Still creating... [00m30s elapsed]
aws_instance.client: Provisioning with 'remote-exec'...
aws_instance.client (remote-exec): Connecting to remote host via SSH...
aws_instance.client (remote-exec):   Host: 52.56.71.58
aws_instance.client (remote-exec):   User: ec2-user
aws_instance.client (remote-exec):   Password: false
aws_instance.client (remote-exec):   Private key: true
aws_instance.client (remote-exec):   Certificate: false
aws_instance.client (remote-exec):   SSH Agent: false
aws_instance.client (remote-exec):   Checking Host Key: false
aws_instance.client (remote-exec):   Target Platform: unix
aws_instance.server (remote-exec): Connecting to remote host via SSH...
aws_instance.server (remote-exec):   Host: 18.170.102.162
aws_instance.server (remote-exec):   User: ec2-user
aws_instance.server (remote-exec):   Password: false
aws_instance.server (remote-exec):   Private key: true
aws_instance.server (remote-exec):   Certificate: false
aws_instance.server (remote-exec):   SSH Agent: false
aws_instance.server (remote-exec):   Checking Host Key: false
aws_instance.server (remote-exec):   Target Platform: unix
aws_instance.server: Still creating... [00m40s elapsed]
aws_instance.client: Still creating... [00m40s elapsed]

```



## After Updating

Re-run:

```
bash

terraform init      # Only if you added a new file or data source
terraform plan -var="key_name=ziakhan"
terraform apply -var="key_name=ziakhan"
```

 This time the EC2 creation should succeed 100%.

---

## Next (Day 5 Preview)

Once EC2s are up, we will:

- SSH into each EC2
  - Copy Docker Compose setup
  - Run and test your containers live on AWS!
-



You've just:

- Built cloud infrastructure with Terraform 🤖
  - Automatically fetched the latest Amazon Linux 2 AMIs 💾
  - Installed Docker on both EC2s via provisioners 🚀
  - Stayed fully aligned with our DevOps project goals ✅
- 

## 📝 Day 5: SSH into EC2s + Deploy Dockerized App with `docker-compose`

Our mission today:

Copy or clone your server and client apps to EC2s and run `docker-compose up` to test the full file + checksum transfer.

---

### 📋 What We'll Do Today:

1. SSH into both EC2s
2. Verify Docker is installed
3. Copy your app code to EC2 (server + client)
4. Run `docker-compose up` to test
5. 📸 Snapshot diagram of where we are!

## 💡 Step 1: SSH into EC2s

You should now have the public IPs from Terraform output:

```
bash
```

```
output "server_ip" = "x.x.x.x"  
output "client_ip" = "y.y.y.y"
```

### 🔒 Connect using .pem key:

```
bash
```

```
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@<IP>
```

Example:

```
bash
```

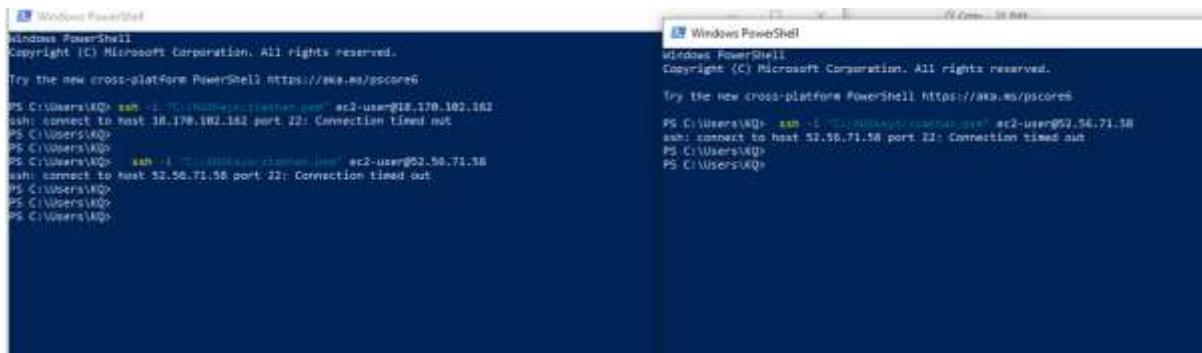
```
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@54.123.45.67
```

Run this for both:

- **server**
- **client**

```
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@18.170.102.162
```

```
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@52.56.71.58
```



I am getting timeout issue, looks like port 22 is blocked. Let's open it.

## Problem:

You're getting:

```
pgsql
```

 Copy  Edit

```
ssh: connect to host <IP> port 22: Connection timed out
```

This means your EC2 instances are up, but port 22 (SSH) is blocked by the security group.

---

## Solution: Open Port 22 in the EC2 Security Group

### Step-by-Step (via AWS Console)

1. Go to the AWS EC2 Dashboard
2. Click **Instances** → select one of the EC2s (server or client)
3. Scroll down to **Security groups** → click the name
4. In the Security Group page:
  - Click **Inbound rules**
  - Click **Edit inbound rules**
  - Click **Add Rule**
  - Type: SSH
  - Protocol: TCP
  - Port: 22
  - Source: Anywhere (0.0.0.0/0) — or restrict to your IP
  - Click **Save rules**

Repeat this for both EC2 instances if they use different security groups.

I have changed the security groups for both instances and they are now working fine. I can do ssh now

The image shows two separate PowerShell windows side-by-side. Both windows have the title bar 'Windows PowerShell' and the command prompt 'PS C:\Users\...\'. The left window displays a long log of an SSH session between two hosts. The right window shows a similar log, but with a warning message about a self-signed certificate from 'amazonlinux 192.168.1.102'. Both logs include timestamps and command history.

```
[...]
```

```
[...]
```

## 💡 Step 2: Verify Docker is Installed

Run:

```
bash
```

Copy Edit

```
docker --version
```

You should see something like:

```
nginx
```

Copy Edit

```
Docker version 24.0.7, build ...
```

Docker is not installed, so we will install docker first on both instances

```
[ec2-user@ip-172-31-42-115:~]
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\KQ> ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@52.56.71.58
ssh: connect to host 52.56.71.58 port 22: Connection timed out
PS C:\Users\KQ>
PS C:\Users\KQ> ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@18.170.102.162
The authenticity of host '18.170.102.162 (18.170.102.162)' can't be established.
ED25519 key fingerprint is SHA256:ttC0g/xV2/iFyH3fqfhSVC1vz2UEoZ75wiNVTYbkjts.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '18.170.102.162' (ED25519) to the list of known hosts.

      #
  ~\_ #####_      Amazon Linux 2
~~ \_#####\_
~~   \###|      AL2 End of Life is 2026-06-30.
~~     \#/ __
~~       \|`-'>
~~         /      A newer version of Amazon Linux is available!
~~._. _/
  _/_/      Amazon Linux 2023, GA and supported until 2028-03-15.
 _/m/'      https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-172-31-42-115 ~]$ docker --version
-bash: docker: command not found
[ec2-user@ip-172-31-42-115 ~]$ docker --version
-bash: docker: command not found
[ec2-user@ip-172-31-42-115 ~]$
```

## Step-by-Step: Install Docker on EC2 (Amazon Linux 2)

### Run these commands on both EC2 instances:

bash

 Copy  Edit

```
# 1. Update packages
```

```
sudo yum update -y
```

```
# 2. Install Docker
```

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

```
# 3. Start Docker service
```

```
sudo service docker start
```

```
# 4. Add ec2-user to docker group (so you don't need sudo for Docker)
```

```
sudo usermod -aG docker ec2-user
```

```
# 5. Verify Docker
```

```
docker --version
```

### Optional (but helpful): Log out and back in to apply group change

You can run:

bash

 Copy  Edit

```
exit
```

Then SSH back in and try:

bash

 Copy  Edit

```
docker info
```

Finally docker is installed on both

```
Select ec2-user@ip-172-31-36-210:~  
Installed size: 264 M  
Downloading packages:  
(1/5): libcgroup-0.41-21.amzn2.x86_64.rpm | 66 kB 00:00:00  
(2/5): pigz-2.3.4-1.amzn2.0.1.x86_64.rpm | 81 kB 00:00:00  
(3/5): containerd-2.0.5-1.amzn2.0.2.x86_64.rpm | 23 MB 00:00:00  
(4/5): docker-25.0.8-1.amzn2.0.5.x86_64.rpm | 45 MB 00:00:00  
(5/5): runc-1.2.6-1.amzn2.x86_64.rpm | 3.5 MB 00:00:00  
-----  
Total 75 MB/s | 72 MB 00:00:00  
Running transaction check  
Running transaction test  
Transaction test succeeded  
Running transaction  
  Installing : runc-1.2.6-1.amzn2.x86_64 1/5  
  Installing : containerd-2.0.5-1.amzn2.0.2.x86_64 2/5  
  Installing : libcgroup-0.41-21.amzn2.x86_64 3/5  
  Installing : pigz-2.3.4-1.amzn2.0.1.x86_64 4/5  
  Installing : docker-25.0.8-1.amzn2.0.5.x86_64 5/5  
  Verifying : containerd-2.0.5-1.amzn2.0.2.x86_64 1/5  
  Verifying : runc-1.2.6-1.amzn2.x86_64 2/5  
  Verifying : docker-25.0.8-1.amzn2.0.5.x86_64 3/5  
  Verifying : pigz-2.3.4-1.amzn2.0.1.x86_64 4/5  
  Verifying : libcgroup-0.41-21.amzn2.x86_64 5/5  
-----  
Installed:  
  docker.x86_64 0:25.0.8-1.amzn2.0.5  
Dependency Installed:  
  containerd.x86_64 0:2.0.5-1.amzn2.0.2      libcgroup.x86_64 0:0.41-21.amzn2      pigz.x86_64 0:2.3.4-1.amzn2.0.1  
  runc.x86_64 0:1.2.6-1.amzn2  
Complete!  
[ec2-user@ip-172-31-36-210 ~]$  
[ec2-user@ip-172-31-36-210 ~]$ # 3. Start Docker service  
[ec2-user@ip-172-31-36-210 ~]$ sudo service docker start  
Redirecting to /bin/systemctl start docker.service  
[ec2-user@ip-172-31-36-210 ~]$  
[ec2-user@ip-172-31-36-210 ~]$ # 4. Add ec2-user to docker group (so you don't need sudo for Docker)  
[ec2-user@ip-172-31-36-210 ~]$ sudo usermod -aG docker ec2-user  
[ec2-user@ip-172-31-36-210 ~]$  
[ec2-user@ip-172-31-36-210 ~]$ # 5. Verify Docker  
[ec2-user@ip-172-31-36-210 ~]$ docker --version  
Docker version 25.0.8, build 0bab007  
[ec2-user@ip-172-31-36-210 ~]$
```

```
ec2-user@ip-172-31-42-115:~  
~/m/ https://aws.amazon.com/linux/amazon-linux-2023/  
[ec2-user@ip-172-31-42-115 ~]$ docker info  
Client:  
  Version: 25.0.8  
  Context: default  
  Debug Mode: false  
  Plugins:  
    buildx: Docker Buildx (Docker Inc.)  
      Version: 0.12.1  
      Path: /usr/libexec/docker/cli-plugins/docker-buildx  
  
Server:  
  Containers: 0  
  Running: 0  
  Paused: 0  
  Stopped: 0  
  Images: 0  
  Server Version: 25.0.8  
  Storage Driver: overlay2  
    Backing Filesystem: xfs  
    Supports d_type: true  
    Using metacopy: false  
    Native Overlay Diff: true  
    userxattr: false  
  Logging Driver: json-file  
  Cgroup Driver: cgroupfs  
  Cgroup Version: 1  
  Plugins:  
    Volume: local  
    Network: bridge host ipvlan macvlan null overlay  
      Log: awslogs fluentd gcplogs gelf journald json-file local splunk syslog  
  Swarm: inactive  
  Runtimes: io.containerd.runc.v2 runc  
  Default Runtime: runc  
  Init Binary: docker-init  
  containerd version: fb4c30d4ede3531652d86197bf3fc9515e5276d9  
  runc version: e89a29929c775025419ab0d218a43588b4c12b9a  
  init version: de40ad0  
  Security Options:  
    seccomp  
      Profile: builtin  
  Kernel Version: 4.14.355-280.652.amzn2.x86_64  
  Operating System: Amazon Linux 2  
  OSType: linux  
  Architecture: x86_64  
  CPUs: 1  
  Total Memory: 970.2MiB  
  Name: ip-172-31-42-115.eu-west-2.compute.internal  
  ID: 938a18a4-dfbf-4d84-b147-c455ce2d5c13  
  Docker Root Dir: /var/lib/docker  
  Debug Mode: false  
  Experimental: false  
  Insecure Registries:  
    127.0.0.0/8  
  Live Restore Enabled: false  
[ec2-user@ip-172-31-42-115 ~]$
```

## 📁 Step 3: Copy Files to EC2

From your local machine, upload your server/client folders:

bash

Copy Edit

```
scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@<SERVER_IP>/home/ec2-user/  
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@<CLIENT_IP>/home/ec2-user/  
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@<SERVER_IP>/home/e
```

Ask ChatGPT

Repeat for `client` EC2 if needed.

---

```
scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@18.170.102.162:/home/ec2-user/  
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@52.56.71.58:/home/ec2-user/  
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@18.170.102.162:/home/ec2-  
user/
```

## Step 2: Copy your files to EC2 using `scp`

On your local machine (in PowerShell), run these **one at a time** from:

bash

 Copy 

```
D:\Devops13\week8\devopscapstone
```

### Copy `server` folder to the server EC2

bash

 Copy 

```
scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@18.170.102.162:/home/ec2-use
```

### Copy `client` folder to the client EC2

bash

 Copy 

```
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@52.56.71.58:/home/ec2-user/
```

### Copy `docker-compose.yml` to the server EC2

bash

 Copy 

```
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@18.170.102.162:/hom
```

"D:\Devops13\week8\devopscapstone"

```
MINGW64/d/Devops13/week8/devopscapstone
$ KODDESKTOP-C8CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@18.170.102.162:/home/ec2-user/
file.txt
server.py
Dockerfile
requirements.txt
100% 1024    84.4KB/s  00:00
100% 625     49.2KB/s  00:00
100% 197     16.9KB/s  00:00
100%   0      0.0KB/s  00:00

KODDESKTOP-C8CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@52.56.71.58:/home/ec2-user/
client.py
received.txt
Dockerfile
requirements.txt
100% 577     44.7KB/s  00:00
100% 1024    80.8KB/s  00:00
100% 172     14.6KB/s  00:00
100%   0      0.0KB/s  00:00

KODDESKTOP-C8CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@18.170.102.162:/home/ec2-user/
docker-compose.yml
100% 443     36.4KB/s  00:00

KODDESKTOP-C8CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$
```

## ✓ What's next?

Once files are copied:

1. SSH into server:

```
bash
Copy Edit
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@18.170.102.162
```

2. Go into the folder:

```
bash
Copy Edit
cd /home/ec2-user
```

3. Start everything using Docker:

```
bash
Copy Edit
docker-compose up --build
```

Once connected, **install Docker** on the EC2:

```
bash ⌂ Copy ⌂ Edit  
sudo yum update -y  
sudo yum install docker -y  
sudo service docker start  
sudo usermod -aG docker ec2-user
```

Then type:

```
bash ⌂ Copy ⌂ Edit  
exit
```

to come back to your local machine.

---

◆ Now do the same for your client EC2:

```
bash ⌂ Copy ⌂ Edit  
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@52.56.71.58
```

Install Docker:

```
bash ⌂ Copy ⌂ Edit  
sudo yum update -y  
sudo yum install docker  
sudo service docker start  
sudo usermod -aG docker ec2-user
```

Ask ChatGPT

Then:

```
bash ⌂ Copy ⌂ Edit  
exit
```

```
ec2-user@ip-172-31-42-115:~$ docker-compose up --build
bash: docker-compose: command not found
ec2-user@ip-172-31-42-115:~$ docker-compose up --build
bash: docker-compose: command not found
ec2-user@ip-172-31-42-115:~$ sudo yum update -y
` start
sudo usermod -aG docker ec2-user
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core
No packages marked for update
[ec2-user@ip-172-31-42-115:~]$ sudo yum install docker -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Package docker-25.0.8-1.amzn2.0.5.x86_64 already installed and latest version
Nothing to do
[ec2-user@ip-172-31-42-115:~]$ sudo service docker start
Redirecting to /bin/systemctl start docker.service
[ec2-user@ip-172-31-42-115:~]$ sudo usermod -aG docker ec2-user
[ec2-user@ip-172-31-42-115:~]$ exit
logout
Connection to 18.170.102.162 closed.
PS C:\Users\KQ> ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@18.170.102.162
Last login: Thu Jul 31 00:02:50 2025 from host81-157-30-7.range81-157.btcentralplus.com
      _#
     ~\_\_ #####_      Amazon Linux 2
     ~~ \#####\_
     ~~ \|##|      AL2 End of Life is 2026-06-30.
     ~~ \|#/_____
     ~~ V~, '-->
     ~~ /          A newer version of Amazon Linux is available!
     ~~ ._. /      Amazon Linux 2023, GA and supported until 2028-03-15.
     ~~ /_/_/
     _/m/'      https://aws.amazon.com/linux/amazon-linux-2023/
[ec2-user@ip-172-31-42-115:~]$ docker-compose up --build
bash: docker-compose: command not found
[ec2-user@ip-172-31-42-115:~]$ cd /home/ec2-user
[ec2-user@ip-172-31-42-115:~]$ cd /home
[ec2-user@ip-172-31-42-115:~]$ cd /ec2-user
bash: cd: /ec2-user: No such file or directory
[ec2-user@ip-172-31-42-115:~]$ cd ec2-user
[ec2-user@ip-172-31-42-115:~]$ docker-compose up --build
bash: docker-compose: command not found
[ec2-user@ip-172-31-42-115:~]$
```

## Why This Happens

By default, Amazon Linux 2 includes Docker, but **not Docker Compose**. You need to install Docker Compose manually.

## Step-by-Step: Install Docker Compose on Amazon Linux 2

1. SSH into your EC2 server (you're already there):

```
bash Copy Edit  
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@18.170.102.162
```

2. Download Docker Compose binary:

```
bash Copy Edit  
sudo curl -L "https://github.com/docker/compose/releases/download/v2.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

3. Apply executable permissions:

```
bash Copy Edit  
sudo chmod +x /usr/local/bin/docker-compose
```

4. Verify installation:

```
bash Copy Edit  
docker-compose --version
```

 You should see something like:

```
mathematica Copy Edit  
Docker Compose version v2.24.0
```

`sudo curl -L "https://github.com/docker/compose/releases/download/v2.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose`

```
ec2-user@ip-172-31-42-115:~/server
Nothing to do
[ec2-user@ip-172-31-42-115 ~]$ sudo service docker start
Redirecting to /bin/systemctl start docker.service
[ec2-user@ip-172-31-42-115 ~]$ sudo usermod -aG docker ec2-user
[ec2-user@ip-172-31-42-115 ~]$ exit
logout
Connection to 18.170.102.162 closed.
PS C:\Users\KQ> ssh -i "C:/AWSKeys/ziahan.pem" ec2-user@18.170.102.162
Last login: Thu Jul 31 00:02:50 2025 from host81-157-30-7.range81-157.btcentralplus.com
,
  _#
 ~\_ #### Amazon Linux 2
~~ \####\ AL2 End of Life is 2026-06-30.
~~ \|/ V~'__->
~~ / A newer version of Amazon Linux is available!
~~/_/ / Amazon Linux 2023, GA and supported until 2028-03-15.
~/m/ https://aws.amazon.com/linux/amazon-linux-2023/
[ec2-user@ip-172-31-42-115 ~]$ docker-compose up --build
-bash: docker-compose: command not found
[ec2-user@ip-172-31-42-115 ~]$ cd /home/ec2-user
[ec2-user@ip-172-31-42-115 ~]$ cd /home
[ec2-user@ip-172-31-42-115 home]$ cd /ec2-user
-bash: cd: /ec2-user: No such file or directory
[ec2-user@ip-172-31-42-115 home]$ cd ec2-user
[ec2-user@ip-172-31-42-115 ~]$ docker-compose up --build
-bash: docker-compose: command not found
[ec2-user@ip-172-31-42-115 ~]$ cd..
-bash: cd..: command not found
[ec2-user@ip-172-31-42-115 ~]$ cd server
[ec2-user@ip-172-31-42-115 server]$ docker-compose up --build
-bash: docker-compose: command not found
[ec2-user@ip-172-31-42-115 server]$ sudo curl -L "https://github.com/docker/compose/releases/download/v2.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
% Total    % Received % Xferd  Average Speed   Time   Time     Current
          Dload  Upload Total Spent   Left  Speed
  0     0    0     0    0     0      0 --:--:-- --:--:-- --:--:--   0
100 58.1M  100 58.1M    0     0  64.5M      0 --:--:-- --:--:-- --:--:-- 88.6M
[ec2-user@ip-172-31-42-115 server]$ sudo chmod +x /usr/local/bin/docker-compose
[ec2-user@ip-172-31-42-115 server]$ docker-compose --version
Docker Compose version v2.24.0
[ec2-user@ip-172-31-42-115 server]$
```

## ✍ Now Try Again

Navigate to your folder:

```
bash
```

Copy Edit

```
cd /home/ec2-user
```

And run:

```
bash
```

Copy Edit

```
docker-compose up --build
```

Let me know once it runs successfully or if anything breaks — we'll fix it together step by step.

Now do the same on client instance

```
ec2-user@ip-172-31-36-210:~  
No packages marked for update  
[ec2-user@ip-172-31-36-210 ~]$ sudo yum install docker -y  
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd  
Package docker-25.0.8-1.amzn2.0.5.x86_64 already installed and latest version  
Nothing to do  
[ec2-user@ip-172-31-36-210 ~]$ sudo service docker start  
Redirecting to /bin/systemctl start docker.service  
[ec2-user@ip-172-31-36-210 ~]$ sudo usermod -aG docker ec2-user  
[ec2-user@ip-172-31-36-210 ~]$ exit  
logout  
Connection to 52.56.71.58 closed.  
PS C:\Users\KQ> ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@52.56.71.58  
Last login: Thu Jul 31 00:02:54 2025 from host81-157-30-7.range81-157.btcentralplus.com  
, #  
~\_\ ##### Amazon Linux 2  
~~ \#####  
~~ \|##| AL2 End of Life is 2026-06-30.  
~~ \|/  
~~ V~'__->  
~~ / A newer version of Amazon Linux is available!  
~~ .- /  
~/ / Amazon Linux 2023, GA and supported until 2028-03-15.  
~/m/ https://aws.amazon.com/linux/amazon-linux-2023/  
[ec2-user@ip-172-31-36-210 ~]$ sudo curl -L "https://github.com/docker/compose/releases/download/v2.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
% Total    % Received % Xferd  Average Speed   Time   Time     Time  Current  
          Dload  Upload Total Spent   Left Speed  
0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0  
100 58.1M 100 58.1M 0 0 96.6M 0 --:--:-- --:--:-- 137M  
[ec2-user@ip-172-31-36-210 ~]$ sudo chmod +x /usr/local/bin/docker-compose  
[ec2-user@ip-172-31-36-210 ~]$ docker-compose --version  
Docker Compose version v2.24.0  
[ec2-user@ip-172-31-36-210 ~]$ cd /home/ec2-user  
[ec2-user@ip-172-31-36-210 ~]$ docker-compose up --build  
no configuration file provided: not found  
[ec2-user@ip-172-31-36-210 ~]$ exit  
logout  
Connection to 52.56.71.58 closed.  
PS C:\Users\KQ> ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@52.56.71.58  
Last login: Thu Jul 31 00:09:46 2025 from host81-157-30-7.range81-157.btcentralplus.com
```

## Step-by-Step Fix

### Step 1: Upload Folders to Correct Machines

Run the following from your local machine (PowerShell):

```
bash Copy Edit
# Upload server folder to server EC2
scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@18.170.102.162:/home/ec2-user/

# Upload client folder to client EC2
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@52.56.71.58:/home/ec2-user/

# Upload docker-compose.yml to BOTH
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@18.170.102.162:/home/ec2-user/
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@52.56.71.58:/home/ec2-user/
```

### Step 2: SSH into the client EC2

```
bash Copy Edit
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@52.56.71.58
```

Then:

```
bash Copy Edit
cd /home/ec2-user/
ls
# You should see the 'client' folder

docker-compose up --build
```



Run these again: I mistakenly didn't upload it to both correctly

```
# Upload server folder to server EC2
```

```
scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@18.170.102.162:/home/ec2-user/
```

```
# Upload client folder to client EC2
```

```
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@52.56.71.58:/home/ec2-user/
```

```
# Upload docker-compose.yml to BOTH
```

```
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@18.170.102.162:/home/ec2-user/
```

```
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@52.56.71.58:/home/ec2-user/
```

```
MINGW64/d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/zia Khan.pem" -r ./server ec2-user@18.170.192.162:/home/ec2-user/
file.txt                                100% 1824      84.4KB/s  00:00
server.py                                100% 625       49.2KB/s  00:00
Dockerfile                               100% 197       16.9KB/s  00:00
requirements.txt                         100%   8       0.0KB/s  00:00

[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/zia Khan.pem" -r ./client ec2-user@52.56.71.58:/home/ec2-user/
client.py                                100% 577       44.7KB/s  00:00
received.txt                            100% 1024      88.8KB/s  00:00
Dockerfile                               100% 172       14.6KB/s  00:00
requirements.txt                         100%   8       0.0KB/s  00:00

[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/zia Khan.pem" ./docker-compose.yml ec2-user@18.170.192.162:/home/ec2-user/
docker-compose.yml                      100% 443      36.4KB/s  00:00

[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/zia Khan.pem" -r ./client ec2-user@52.56.71.58:/home/ec2-user/
client.py                                100% 577       38.4KB/s  00:00
received.txt                            100% 1024      59.3KB/s  00:00
Dockerfile                               100% 172       12.0KB/s  00:00
requirements.txt                         100%   8       0.0KB/s  00:00

[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/zia Khan.pem" ./docker-compose.yml ec2-user@18.170.192.162:/home/ec2-user/
docker-compose.yml                      100% 443      25.4KB/s  00:00

[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/zia Khan.pem" ./docker-compose.yml ec2-user@52.56.71.58:/home/ec2-user/
docker-compose.yml                      100% 443      26.7KB/s  00:00

[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$
```

Again there are issues on both sides. Client is not finding server and vice versa.

## Problem:

Your `docker-compose.yml` on both EC2 instances refers to:

yaml

 Copy  Edit

```
context: /home/ec2-user/server
context: /home/ec2-user/client
```

But these folders **do not exist** because either:

- The `scp` command was not executed correctly.
  - The files were sent to the wrong path.
  - The folders were not copied due to a path typo or permissions issue.
- 

```
scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@18.170.102.162:/home/ec2-user/
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@18.170.102.162:/home/ec2-user/
```

```
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@52.56.71.58:/home/ec2-user/
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@52.56.71.58:/home/ec2-user/
```

## ⌚ Step 2: Run `scp` to copy the folders to the EC2 instances

### ⌚ For Server (18.170.102.162):

```
bash Copy Edit
scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@18.170.102.162:/home/ec2-user
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@18.170.102.162:/home/ec2-user
```

### ⌚ For Client (52.56.71.58):

```
bash Copy Edit
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@52.56.71.58:/home/ec2-user/
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@52.56.71.58:/home/ec2-user
```

## ✓ Problem Summary

Your `docker-compose.yml` assumes both `/home/ec2-user/server` and `/home/ec2-user/client` exist on the same EC2 machine.

But:

- Server EC2 has only `/server`
- Client EC2 has only `/client`

`docker-compose` tries to build both services (client + server) from the same machine, which causes:

```
pgsql Copy Edit
unable to prepare context: path "/home/ec2-user/client" not found
```

## 🛠 Fix It (Choose One of the Two Options Below)

## Option 1: Run Both Containers on Same EC2

If you just want to run both server and client on one EC2, do this (easiest fix):

### Step-by-step Fix:

1. On your local machine, copy both folders to **one** EC2 (e.g., the *server* EC2):

```
bash Copy Edit
scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@18.170.102.162:/home/ec2-use
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@18.170.102.162:/home/ec2-use
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@18.170.102.162:/hom
```

2. SSH into server EC2:

```
bash Copy Edit
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@18.170.102.162
```

3. Go into folder and run:

```
bash Copy Edit
cd /home/ec2-user
docker-compose up --build
```

That should work—because both folders now exist.

---

## Summary

Option	Use case	Status
1	Run both on one EC2	<input checked="" type="checkbox"/> Easy, Recommended
2	Separate EC2s for client/server	<input checked="" type="checkbox"/> Complex for now

---

Now with option 1 it is successful

```

[+] Running 5/1
  Network ec2-user-devnet      Created
  Volume "ec2-user_clientvol"   Created
  Volume "ec2-user_servervol"   Created
  Container devops_server      Created
  Container devops_client      Created
Attaching to devops_client, devops_server
devops_server | Server is waiting for connection...
devops_server | Connected to ('172.18.0.3', 33352)
devops_server | File and checksum sent.
devops_client | Raw received data: b'\xe69\xaf"\xe2h\xc5\\xd3J\xc2jW\xecmLDzn\x1e\x0e\xef\xaa6\x1c\xe0n8!\x92\xa2\x1e\xb5\x12.\x98i\x2a\xd1\xab\xeb{\x8ajo\xd3A\x14\xd2\xfc\x87\xf2\x15\xa7\xf1\xn\xThA\x18apI\x9a>\x07\x97\xf4\xc4\x8e\x17Zm%\x
\xc8[\xb8K\xf1z\xccW-z\xd5\xe4\xd5 2\x91\xa0K\xff\x96-\x94\x18'\s9\xlaW\x94\xa3M\x0f\xc2\x9a`\'\xfdq\xb4\xf3&\x4f7z
\xd1\x89:o\x9b/\xdc\xKQ\xe75\xee\xd4R\xadR\xf1\xf9\x89I\x80\xadil\xfb\xd7\xac\xfc\x97z\x83\xd2\x0b\x88\xfd\xda3\xef\xeei?'
\x10\x99\x08-\xc1-\x18.\xe3xA<vy\xf6=\x9a\x8b\x1c\xf1\xbf\x10\x91B\xe0\x8Ag\xfe(\x03x;\x92\xr\xfa\x8b\'A\x01\xbb\x1c\xec|
:x12\x91\xd7r\x05w\xe7/\xcb\x99D\x14\x9a\x03|\xae\x0\xdc\xef\x8e\x15t\x0b\xf5\xde\x21\x99\xcc\xc3\xcd\x02\xc0\x16\xd
3\'\xe3y\xf63\xecs}~\xd5\x9d\x9b\x17e\x88\x1d\xbf\x9f\x9f\xbf]\x82Ac\xe2\xac\\\x1a\xb0\n@yM9\x87N\xfd\xd0;\xef\x01\x
\xfaL\x8a\x2:\xce\xd5j\x8cBC\x95\x0b(\x0e\x85\\xfe\xC\x94\x11\xb2\xf9\xb6\xdc,\x0b\xc1\x88w\xd5\x81\xc5\xb9H.\xb2)j\x
\xc0\x8cJ\x2\x8d\x89f\x0\x1c\x9a\xcb]S\xf1\x92\xeb\xe0\xfc\x87\xfc\xf0\xf4\xaf'\xeb\xbf\x7f;#\x8c\xbd\xc0)d\xd9\x0b
&\x90w\x1e\x88\xab*\x0\xc8\x0\x7\xa6\xfc\x9aw\xb1\xd6\x8c\x9e\x1\x90\x11\xc2\xcz.\x879\xbc\x8c\x1\xdeT\xn\xf3\xe5\xc3\x8b\xc74
\xd5\xba\xfc\x1\xe2\x1fN\x87\xa5\x02\xfd^\'\x81\x9c\x81\x88\x1b\xd3\xb6\xfa|\x85n\xd4\xaf\xf2\x8b\x85\xe6\xf1\xf1'
\xd3\xfb%\xf8=\'\xeb\xd0\xce/B\xe4@\xd3\x06\xeaIP\x91\x7\xd8\xc7r\xe1\x870\x9aG~\x7f\x87\xac\x985\xf6C\x8e\xc8\x1a\x
89\x8dD\xdf\x8c\x88\xbd'\x8c\xf2\x8b\x0\x85\x857.\x4D\x18^\v~\xcc\x80|\x8f\xe0\x1fF\xaf\x2F\x85\x06\x82\xfb\xe5_\x9mK\x
c0\x14\xd7\'\r-D\xff\x98p\x1b?\x015\xe8\xbd\xd8w\x98<\x01\xb2\xf3\xf8\x13\xacg\x92Y&\x8f\xf\xd7\x8c\x1a\xf4\x8c\x088
Q,\xdc\xb3\xd2\xdcX\x9e\x16;\x98\xc7\xcbN\xee\xabV\xc6\xf3\xe7\x07{\xad}\xcd\xf7-\xb6\xf7"\x10\xb5\xed\xb7\xcev\x1ct\x
\xe4\xd1<>\x8c\x92\x18\t\xef\xfb\xea\x17#\x96a\x12Y^jT\xd3\x9aaL\x1b\r\xd3\x16L\x0b1\xc0\x1e\x96\x86\xe9\xd9=\x98\x
b1r\xk\x06\xbf4\x07\x07\x07\x9e11\xb9\xd6\x97D\xb0\x5\xd8j\xc9A\x82\x83HS\x80\xfe\x18\x1a#\xda\xdb\xd3\x1c\x9dU\xdd\xce\x
\xb7\xbb\x9d\xef?\x1d\xbf\x04\x040\xb7\xa0\xe4\xb&G\xe5\xdf\xc3.\x1\x88%\x14>\x16\x88\xkY\xac\x8b\x16\x91\xfd\x1d\x0c,
\x84\xdcn\x02M\xf9P\x0f\x05\x08\x15\x80\xab\x15\x8d^\'\xfc\xb6\xb5\xf5\xdaw\xba\xc8z\xaa\xd5\xb3\xfeBu\xafQ\x4I\xK\'\xd2\x
\x86\x5\x01\x0e\x867\xcb\x861\xc2\xab\xe4\x06\xb4\xf7\xdc\xb9\xb6\xc3\x80\xf8\x95\xe8\x12j\xc2\x13#\x8b\xf7j\xea\x90\x
wR\x91t^\'\x9e\xb0\xdeat[(\x7f\xe3\xbb-A\x86\x06\xac2F\xaa\xb0\xbu\x0f\x04\x93\x14\xbf\xd7\x98\x8c\xdc\xc1\xd7\xc5\x19\x
\x0\xf8\xf21\x95\xee\x89\x84\x0\x0\xdd8\xf3N\xc2\xf5}IRF*$\x0\xe2\xe0\xe8\x18\xfc\xc5\x8c\x9b*\xb3\xeb\x95\xb9(\x
\xcf\x84\x0e=\x02\x7f\x19\xc6\x88K\xcb\x87X\x87\x94\x97\xb\x6\xfc.\x8f\x0m\xf4n\x4n\x7Y\xe8\xb6\x8d\xcc\x04\xf2K\x84\xb
2\x882!\x90\xfaGA\x0\xdfu\x8f0\x0\xd0\xec\xf1!\x04\x80\x0\x3J\xda\x16\x1d.\xf3\xf3J\xd9F\xeb\x1c\xc3\xz9\x14\x8c\x1a\x
9eQ\xc5\x92D\x80\x85\xbd\x07)^R\x11a\xf5-jV::71e174e8ddf634c43cffc26039ca4a4b'
devops_client | File received and verified successfully.
devops_client exited with code 0
devops_server exited with code 0
[ec2-user@ip-172-31-42-115 ~]$
```

## ➡ What to do now?

Please run the three `scp` commands again, copying both `client` and `server` to your server EC2.

Once done, let me know and we'll run the containers.

## Expected Output:

- Server creates a file + checksum and sends it.
- Client receives and verifies it.
- Console shows:

arduino

 Copy  Edit

 File received and verified successfully.

## Project Snapshot After Day 5

plaintext

 Copy  Edit

[Milestone: LIVE CONNECTION SUCCESS  ]

- ✓ EC2 server up & Docker installed
- ✓ EC2 client up & Docker installed
- ✓ SSH working
- ✓ Apps deployed using Docker Compose
- ✓ Client-server verified end-to-end!

```
[+] Running 5/1
  Network ec2-user_devnet      Created          0.1s
  Volume "ec2-user_clientvol"   Created          0.0s
  Volume "ec2-user_servervol"   Created          0.0s
  Container devops_server     Created          0.0s
  Container devops_client     Created          0.0s
Attaching to devops_client, devops_server
devops_server | Server is waiting for connection...
devops_server | Connected to ('172.18.0.3', 33352)
devops_server | File and checksum sent.
devops_client | Raw received data: b'\x{...}' (truncated for brevity)
devops_client | File received and verified successfully.
devops_client exited with code 0
devops_server exited with code 0
[ec2-user@ip-172-31-42-115 ~]$
```

## ✖ Option 2: Distributed Architecture (Advanced)

If you really want to run client on one EC2 and server on another, then:

- You cannot use a single `docker-compose.yml` to deploy both from one machine.
- You must:
  - Split the deployment manually.
  - Run `docker build` and `docker run` separately on each EC2.
  - Or use advanced setups like Docker Swarm, Kubernetes, or SSH context switching.

Let me know if you want this advanced setup, but I recommend Option 1 for now.

We will do this option 2 later.

Great work! Since you've now successfully:

- Installed Docker and Docker Compose on both EC2 instances
  - Uploaded the `server`, `client`, and `docker-compose.yml` files to the same EC2 instance
  - Ran `docker-compose up --build` successfully without context errors
- 

## 👉 You've Completed So Far:

1. 🔑 Set up AWS CLI & SSH key
  2. 🖥️ Created two EC2 instances (intended for server and client)
  3. 💾 Installed Docker + Docker Compose
  4. 📁 Copied project files to EC2
  5. 🛠️ Built Docker containers using `docker-compose`
- 

I am getting issues as no docker container is running. I have resumed and restarted ec2 instances and new ips are assigned to each instance:

server: ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@13.42.11.17

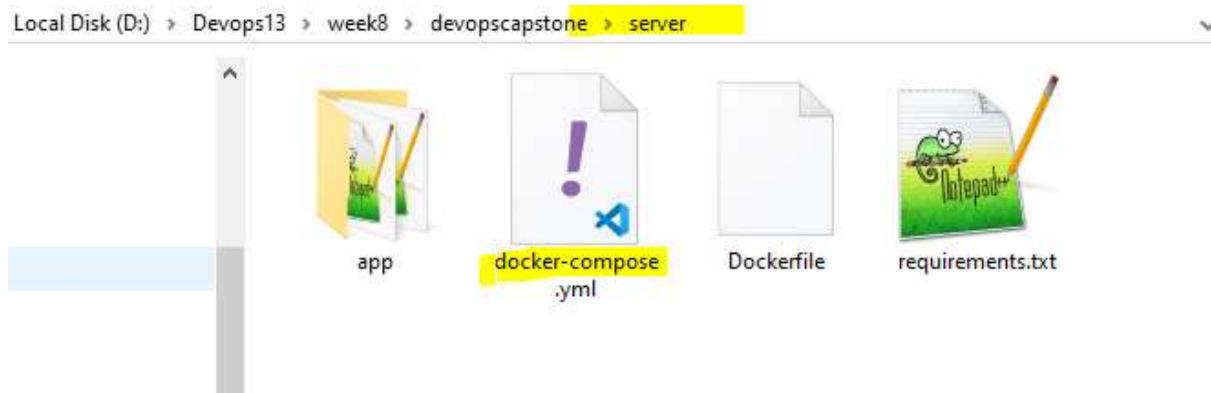
client: ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@35.179.129.150

The image shows two terminal windows side-by-side. Both windows are titled 'Terminal' and show the command 'git clone https://github.com/ziakhan/docker-test.git' followed by 'cd docker-test' and 'docker-compose up --build'. The left window, labeled 'server', shows the build process for a container named 'server'. It lists several Dockerfiles: 'Dockerfile', 'Dockerfile.nginx', 'Dockerfile.nginx.1', 'Dockerfile.nginx.2', 'Dockerfile.nginx.3', 'Dockerfile.nginx.4', and 'Dockerfile.nginx.5'. It also shows the creation of volumes and the start of the 'nginx' service. The right window, labeled 'client', shows the build process for a container named 'client'. It lists several Dockerfiles: 'Dockerfile', 'Dockerfile.nginx', 'Dockerfile.nginx.1', 'Dockerfile.nginx.2', 'Dockerfile.nginx.3', 'Dockerfile.nginx.4', and 'Dockerfile.nginx.5'. It also shows the creation of volumes and the start of the 'nginx' service. Both windows show the status of the build process, with the server window showing 'Up 0.0s' and the client window showing 'Up 0.0s'.

```
git clone https://github.com/ziakhan/docker-test.git
cd docker-test
docker-compose up --build

server: ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@13.42.11.17
client: ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@35.179.129.150
```

Now I have created two docker-compose.yml files, one separate for server and another on client folder in local project folder.



And similarly created another on client folder of the project:

```
version: '3.9'
services:
  client:
    build: ./client
    container_name: devops_client
    volumes:
      - clientvol:/app/data
    ports:
      - "3000:3000"
    networks:
      - devnet
volumes:
  clientvol:
networks:
  devnet:
```

```
version: '3.9'
services:
  server:
    build: ./server
    container_name: devops_server
    volumes:
      - servervol:/app/data
    ports:
      - "5000:5000"
    networks:
      - devnet
volumes:
  servervol:
networks:
  devnet:
```

## 💡 What's Going On?

You're seeing this error on your client instance (left side, IP 35.179.129.150):

```
pgsql                               ⌂ Copy ⌂ Edit  
unable to prepare context: path "/home/ec2-user/server" not found
```

That means your `docker-compose.yml` refers to a `./server` folder that doesn't exist on the client machine.

## ✓ Solution

### 📘 Step 1: Sync the `server` and `client` folders correctly to each instance

On the Server instance (IP: 13.42.11.17), you should have:

```
pgsql                               ⌂ Copy ⌂ Edit  
/home/ec2-user/  
|   docker-compose.yml  
|   server/  
|   |   server.py  
|   client/      # optional if not using client there
```

On the Client instance (IP: 35.179.129.150), you should have:

```
bash                               ⌂ Copy ⌂ Edit  
/home/ec2-user/  
|   docker-compose.yml  
|   client/  
|   |   client.py  
|   server/      # remove if not needed
```

⚠️ Important: Each machine should only contain the folder it actually builds in `docker-compose.yml`.



### 📘 Step 3: Upload your local updated `docker-compose.yml` and app folders

On your local machine, use `scp` to copy over files:

To Server:

```
bash                               ⌂ Copy ⌂ Edit  
scp -i C:/AWSKeys/ziakhan.pem -r ./server ec2-user@13.42.11.17:/home/ec2-user/  
scp -i C:/AWSKeys/ziakhan.pem ./docker-compose.yml ec2-user@13.42.11.17:/home/ec2-user/
```

To Client:

```
bash                               ⌂ Copy ⌂ Edit  
scp -i C:/AWSKeys/ziakhan.pem -r ./client ec2-user@35.179.129.150:/home/ec2-user/  
scp -i C:/AWSKeys/ziakhan.pem ./docker-compose.yml ec2-user@35.179.129.150:/home/ec2-user/
```

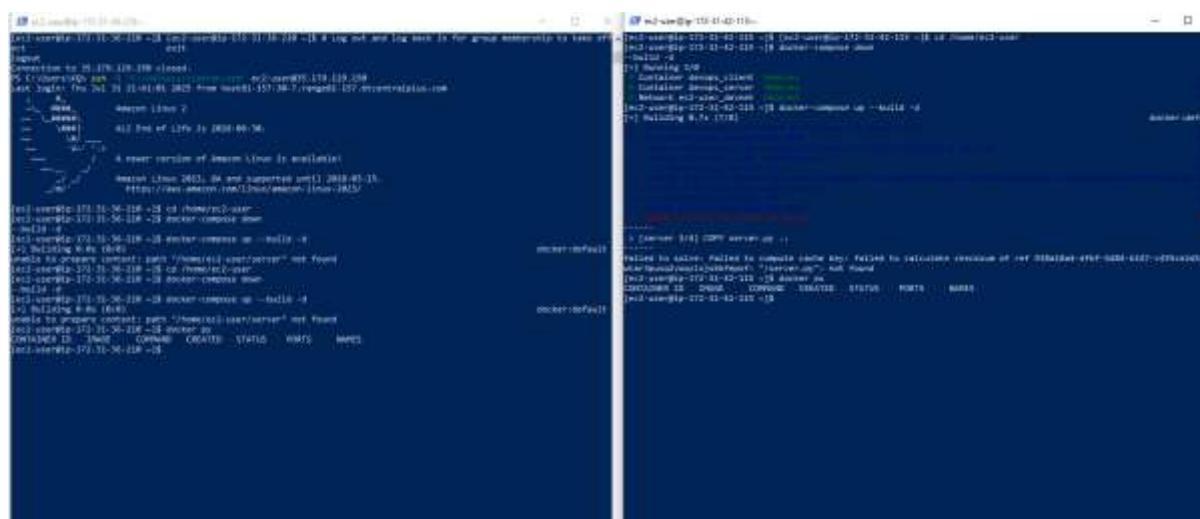
```
[1] MINGW64 /d/Devops13/week8/devopscapstone
[2] KODESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i C:/AWSKeys/ziakhan.pem -r ./server ec2-user@13.42.11.17:/home/ec2-user/
file.txt
server.py
docker-compose.yml
Dockerfile
requirements.txt
100% 1024    72.6KB/s  00:00
100% 625     52.5KB/s  00:00
100% 247     20.4KB/s  00:00
100% 164     13.7KB/s  00:00
100% 0      0.0KB/s  00:00

[3] KODESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i C:/AWSKeys/ziakhan.pem ./docker-compose.yml ec2-user@13.42.11.17:/home/ec2-user/
docker-compose.yml
100% 476    36.5KB/s  00:00

[4] KODESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i C:/AWSKeys/ziakhan.pem -r ./client ec2-user@35.179.129.150:/home/ec2-user/
client.py
received.txt
docker-compose.yml
Dockerfile
requirements.txt
100% 577    43.5KB/s  00:00
100% 1024   78.6KB/s  00:00
100% 247    17.5KB/s  00:00
100% 166    11.0KB/s  00:00
100% 0      0.0KB/s  00:00

[5] KODESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i C:/AWSKeys/ziakhan.pem ./docker-compose.yml ec2-user@35.179.129.150:/home/ec2-user/
docker-compose.yml
100% 476    31.6KB/s  00:00

[6] KODESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
```



### 3. Upload Files to EC2 Machines

From your Windows machine, run the following in PowerShell or Git Bash:

For the Server EC2 ( 13.42.11.17 ):

```
bash Copy Edit
scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@13.42.11.17:/home/ec2-user/
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@13.42.11.17:/home/ec2-user/
```

Make sure this `docker-compose.yml` contains only the server config (see above).

For the Client EC2 ( 35.179.129.150 ):

```
bash Copy Edit
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@35.179.129.150:/home/ec2-user/
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose.yml ec2-user@35.179.129.150:/home/ec2-user/
```

This `docker-compose.yml` should contain only the client config.

I did one thing here, docker-compose.yml file has been copied from server directory and then run the code in the screenshot above for server, and then copied docker-compose.yml file from client to root directory of project and run the scp code for the client ec2.

```
MINGW64:/d/Devops13/week8/devopscapstone
[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziahan.pem" ./docker-compose.yml ec2-user@13.42.11.17:/home/ec2-user/
docker-compose.yml                                         100%  476    33.6KB/s  00:00

[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziahan.pem" -r ./server ec2-user@13.42.11.17:/home/ec2-user/
file.txt                                                 100% 1824    71.6KB/s  00:00
server.py                                               100%  625   45.2KB/s  00:00
docker-compose.yml                                     100%  247   18.8KB/s  00:00
Dockerfile                                              100%  164   13.4KB/s  00:00
requirements.txt                                       100%   8    0.0KB/s  00:00

[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziahan.pem" ./docker-compose.yml ec2-user@13.42.11.17:/home/ec2-user/
docker-compose.yml                                         100%  247    20.9KB/s  00:00

[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziahan.pem" -r ./client ec2-user@35.179.129.150:/home/ec2-user/
client.py                                              100%  577   49.1KB/s  00:00
received.txt                                           100% 1824   82.6KB/s  00:00
docker-compose.yml                                     100%  247   20.5KB/s  00:00
Dockerfile                                              100%  166   14.1KB/s  00:00
requirements.txt                                       100%   8    0.0KB/s  00:00

[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziahan.pem" ./docker-compose.yml ec2-user@35.179.129.150:/home/ec2-user/
docker-compose.yml                                         100%  247    18.9KB/s  00:00

[Q]DESKTOP-CBCHCRF MINGW64 /d/Devops13/week8/devopscapstone
$
```

#### ✓ 4. SSH into EC2 & Build Docker

Then, on each EC2 instance:

```
bash
cd /home/ec2-user
docker-compose down
docker-compose up --build -d
```

Then verify:

```
bash
docker ps
```

Got the error again:

I have made changes to the dockerfiles:

## Step 1: Make Sure Local Folder Structure Looks Like This

On your Windows PC, create a folder like this:

```
pgsql
```

devops-capstone/  
| └── docker-compose.yml  
| └── server/  
| | └── Dockerfile  
| | └── app/  
| | └── server.py  
| └── client/  
| | └── Dockerfile  
| | └── app/  
| | └── client.py

Each Dockerfile should match this:

```
server/Dockerfile

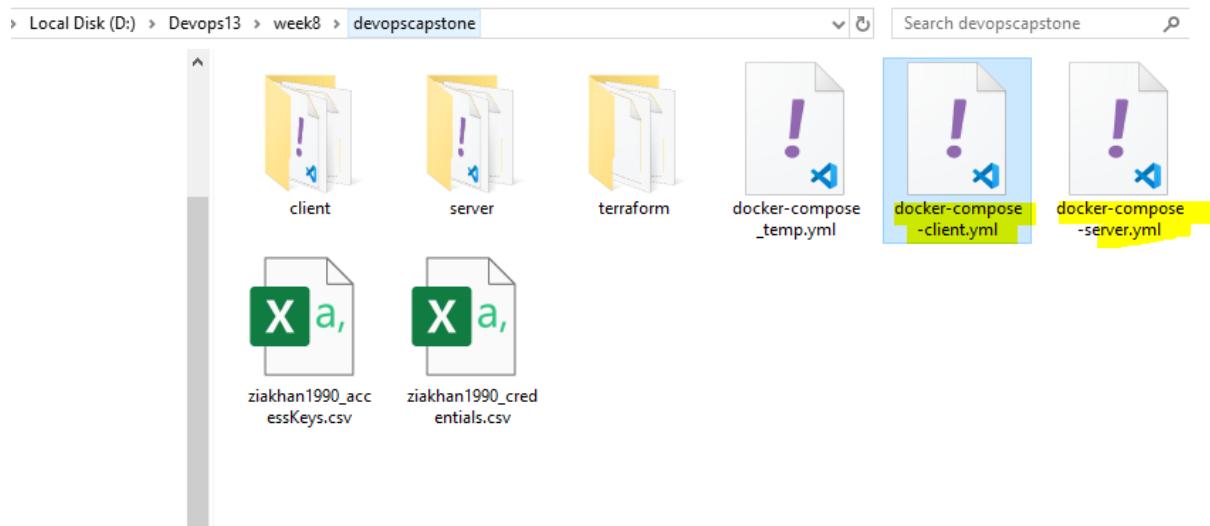
Dockerfile
Copy Edit

FROM python:3.10-slim
WORKDIR /app
COPY app/server.py .
RUN mkdir /app/data
CMD ["python", "server.py"]


client/Dockerfile

Dockerfile
Copy Edit

FROM python:3.10-slim
WORKDIR /app
COPY app/client.py .
RUN mkdir /app/data
CMD ["python", "client.py"]
```



Two separate files : docker compose for client and server

### Step 2: SCP Upload to Each EC2 Instance

Open PowerShell or Git Bash and run:

For Server EC2 ( 13.42.11.17 )

```
bash Copy Edit
scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@13.42.11.17:/home/ec2-user/
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose-server.yml ec2-user@13.42.11.17:/home/ec2-user/do
```

For Client EC2 ( 35.179.129.150 )

```
bash Copy Edit
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@35.179.129.150:/home/ec2-user/
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose-client.yml ec2-user@35.179.129.150:/home/ec2-user/
```

**!Make sure you are copying the correct compose file for each — only include `client` on the client EC2, and `server` on the server EC2.**

Server:

```
scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@13.42.11.17:/home/ec2-user/
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose-server.yml ec2-user@13.42.11.17:/home/ec2-user/docker-compose.yml
```

Client:

```
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@35.179.129.150:/home/ec2-user/
```

```
scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose-client.yml ec2-user@35.179.129.150:/home/ec2-user/docker-compose.yml
```

The screenshot shows a terminal window with the following command history:

```
MINGW64:/d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziakhan.pem" -r ./server ec2-user@35.42.11.17:/home/ec2-user/
file.txt
server.py
docker-compose-server.yml
Dockerfile
requirements.txt
100% 1024    68.0KB/s  00:00
100% 625     44.4KB/s  00:00
100% 247     18.8KB/s  00:00
100% 109     8.8KB/s  00:00
100%   0      0.0KB/s  00:00

MINGW64:/d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziakhan.pem" ./docker-compose-server.yml ec2-user@35.42.11.17:/home/ec2-user/docker-compose.yml
100% 247    19.3KB/s  00:00

MINGW64:/d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@35.179.129.150:/home/ec2-user/
client.py
received.txt
docker-compose-client.yml
Dockerfile
requirements.txt
100% 577    39.8KB/s  00:00
100% 1024   71.7KB/s  00:00
100% 247   19.4KB/s  00:00
100% 109   8.4KB/s  00:00
100%   0   0.0KB/s  00:00

MINGW64:/d/Devops13/week8/devopscapstone
$
```

### Step 3: SSH and Run Build

On Server EC2 ( 13.42.11.17 )

```
bash
cd /home/ec2-user
docker-compose down
docker-compose up --build -d
```

 Copy 

On Client EC2 ( 35.179.129.150 )

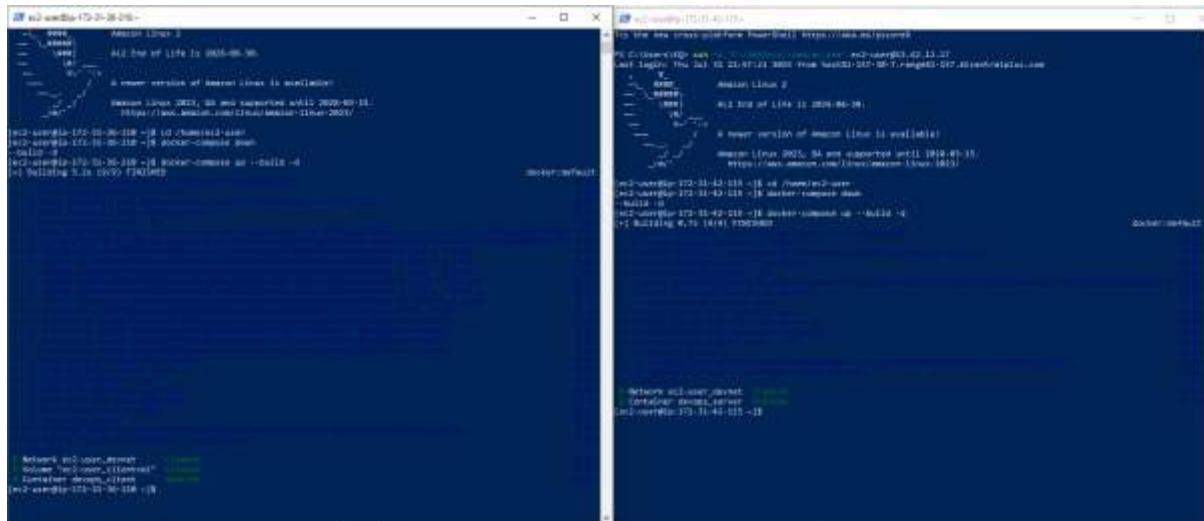
```
bash
cd /home/ec2-user
docker-compose down
docker-compose up --build -d
```

 Copy 

Then confirm with:

```
bash
docker ps
```

 Copy 



Containers have been created on server side but not on client instance: need to investigate this further:

The screenshot shows a terminal window with four panes. The top-left pane lists Docker networks and containers. The top-right pane shows a detailed log for a container named 'client'. The bottom-left pane shows another list of Docker networks and containers. The bottom-right pane shows a command-line interface with some text input.

```
Networks: ec2-user_server
Container: devops_client
Container: devops_server
Container: client
Container: server

Networks: ec2-user_server
Container: devops_client
Container: devops_server
Container: client
Container: server

Networks: ec2-user_server
Container: devops_client
Container: devops_server
Container: client
Container: server

Networks: ec2-user_server
Container: devops_client
Container: devops_server
Container: client
Container: server
```

### ✓ What's working:

- On both EC2 instances (server and client), your containers are **building successfully** with `docker-compose up --build -d`.
- The `client` container **does start** (`devops_client Started` message confirms this).
- Files like `client.py` are **present** inside `client/app/`.

### ✗ What's NOT working:

When you run `docker ps`, the client container **does not appear**, even though `docker-compose` reports that it started.

### 🔍 Why this is happening:

This is very likely due to your container **exiting immediately after starting**, which is common if:

- Your Python app (`client.py`) **finishes execution instantly** and exits.
- There's **no long-running process** (like Flask or FastAPI server) t **Ask ChatGPT** er alive.
- You **don't have a CMD or ENTRYPPOINT** in your `Dockerfile`, or they are not correct.

## 🛠 How to Fix It

### Step 1: Check if container exited

Run:

```
bash
docker ps -a
```

This will show **all containers**, including those that exited. You'll likely see the `devops_client` container with a status like `Exited (0)` or some error.

---

### Step 2: Check the logs

Get the logs of the exited container:

```
bash
docker logs devops_client
```

This will show **why it exited** — maybe it just printed something and ended, or there's a traceback or import error.

---

### Step 3: Ensure your app runs as a service

If you're using `client.py` like a script, it may just be ending after printing/receiving something.

Instead, you should run something like a Flask server:

```
python

# client/app/client.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Client is running!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=3000)
```

Update your `requirements.txt`:

```
nginx
flask
```

Then update your `Dockerfile` to:

```
dockerfile

FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY app/ .

CMD ["python", "client.py"]
```

#### Step 4: Rebuild and run again

```
bash
```

Copy Edit

```
docker-compose down  
docker-compose up --build -d
```

Then check:

```
bash
```

Copy Edit

```
docker ps
```

You should now see the container **up and running** on port 3000.

---

#### Summary

- Your container starts but exits immediately because the app doesn't keep running.
- Use `docker ps -a` and `docker logs` to confirm this.
- Turn your `client.py` into a long-running service like a Flask app.
- Use proper `CMD` in Dockerfile.

```
[ec2-user@ip-172-31-36-210:~]
```

```
[ec2-user@ip-172-31-36-210 ~]$ docker-compose down
[ec2-user@ip-172-31-36-210 ~]$ docker-compose up --build -d
[+] Building 0.7s (9/9) FINISHED
  => client internal load build definition from Dockerfile
  => transferring dockerfile: 346B
  => client internal load metadata for docker.io/library/python:3.10-slim
  => client internal load .dockerrcignore
  => transferring context: 2B
  => client 1/4 FROM docker.io/library/python:3.10-slim@sha256:81f1cd83770d54ecfdbddcc52c2125*ce674c14a1d976dfd8
  => client internal load build context
  => transferring context: 646B
  => CACHED [client 2/4] WORKDIR /app
  => CACHED [client 3/4] COPY app/client.py .
  => CACHED [client 4/4] RUN mkdir /app/data
  => (client) exporting to image
  => exporting layers
  => writing image sha256:3db7a657114894ea8b3bf81030c6fb205d92790ab6946423b523a00ca130b2a
  => naming to docker.io/library/ec2-user-client
[+] Running 2/2
  ⚡ Network ec2-user_devnet  Created
  ⚡ Container devops_client  Started
[ec2-user@ip-172-31-36-210 ~]$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[ec2-user@ip-172-31-36-210 ~]$ ls
client docker-compose.yml
[ec2-user@ip-172-31-36-210 ~]$ ls client/app/
client.py received.txt
[ec2-user@ip-172-31-36-210 ~]$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
bea8c8edd4fa  ec2-user-client "python client.py"  3 minutes ago  Exited (1) 3 minutes ago
[ec2-user@ip-172-31-36-210 ~]$ docker logs devops_client
Traceback (most recent call last):
  File "/app/client.py", line 21, in <module>
    main()
  File "/app/client.py", line 5, in main
    s.connect(("server", 5000))
socket.gaierror: [Errno -2] Name or service not known
[ec2-user@ip-172-31-36-210 ~]$
```

Thanks for sharing all the details. You're extremely close to getting everything running — the issue is with how the client is trying to connect to the server. Let me explain and guide you through the fix.

---

### ✓ Issue Recap:

Your `client.py` is using:

```
python                                     ⌂ Copy ⌂ Edit  
s.connect(("server", 5000))
```

This assumes that `server` is the **hostname** of the server container. That only works if:

- Both client and server are running in the **same Docker network**
- And launched via the **same `docker-compose.yml`**

But:

- You're running client and server on **separate EC2 instances**
- So `"server"` cannot be resolved as a DNS name — hence the error:

```
less                                     ⌂ Copy ⌂ Edit  
socket.gaierror: [Errno -2] Name or service not known
```

### Fix – Replace "server" with Public IP of EC2 Server Instance:

On the client EC2 instance, in `client/app/client.py`, change:

```
python
```

 Copy 

```
s.connect(("server", 5000))
```

to:

```
python
```

 Copy 

```
s.connect(("13.42.11.17", 5000)) # Replace with your actual EC2 server public IP
```

 You can also make this IP configurable via environment variable if you want to avoid hardcoding.

---

### Optional – Debugging & Verification Steps

#### 1. Verify server is running and listening:

On the server EC2 instance:

```
bash
```

 Copy 

```
sudo lsof -i -P -n | grep LISTEN
```

#### 2. Ping server IP from client:

On client EC2:

```
bash
```

 Copy 

```
ping 13.42.11.17
```

#### 3. Test raw socket connection manually:

From client EC2:

```
bash
```

 Copy 

```
telnet 13.42.11.17 5000
```



```

ec2-user@ip-172-31-42-115:~ 
~~~ V~' '--> 
~~~ / A newer version of Amazon Linux is available!
~~~_/_/_/ Amazon Linux 2023, GA and supported until 2028-03-15.
~/m/' https://aws.amazon.com/linux/amazon-linux-2023/
[ec2-user@ip-172-31-42-115 ~]$ cd /home/ec2-user
[ec2-user@ip-172-31-42-115 ~]$ docker-compose down
--build -d
[ec2-user@ip-172-31-42-115 ~]$ docker-compose up --build -d
[+] Building 0.7s (9/9) FINISHED
--> [server internal] load build definition from Dockerfile
--> transferring dockerfile: 146B
--> [server internal] load metadata for docker.io/library/python:3.10-slim
--> [server internal] load .dockerignore
--> > transferring context: 2B
--> [server 1/4] FROM docker.io/library/python:3.10-slim@sha256:81f1cd83770d54ecfdbddcc52c2125fce074c14aid976fd
--> [server internal] load build context
--> > transferring context: 694B
--> CACHED [server 2/4] WORKDIR /app
--> CACHED [server 3/4] COPY app/server.py .
--> CACHED [server 4/4] RUN mkdir /app/data
--> [server] exporting to image
--> > exporting layers
--> > writing image sha256:846b5ad2bdb0ae6bdab21b6f7eaa9e43470b129247b2703657954015f716d7ce
--> > naming to docker.io/library/ec2-user-server
[+] Running 2/2
  Network ec2-user_devnet  Created
  Container devops_server  Started
[ec2-user@ip-172-31-42-115 ~]$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
 NAMES
5e3d57fbbe3a ec2-user-server "python server.py" About a minute ago Up About a minute 0.0.0.0:5000->5000
::5000->5000/tcp devops_server
[ec2-user@ip-172-31-42-115 ~]$ sudo lsof -i -P -n | grep LISTEN
rpcbind 2538 rpc 8u IPv4 15112 0t0 TCP *:111 (LISTEN)
rpcbind 2538 rpc 11u IPv6 15115 0t0 TCP *:111 (LISTEN)
master 2942 root 13u IPv4 16693 0t0 TCP 127.0.0.1:25 (LISTEN)
sshd 2980 root 3u IPv4 17842 0t0 TCP *:22 (LISTEN)
sshd 2980 root 4u IPv6 17851 0t0 TCP *:22 (LISTEN)
container 3195 root 12u IPv4 19372 0t0 TCP 127.0.0.1:42169 (LISTEN)
docker-pr 5293 root 4u IPv4 27171 0t0 TCP *:5000 (LISTEN)
docker-pr 5298 root 4u IPv6 27177 0t0 TCP *:5000 (LISTEN)
[ec2-user@ip-172-31-42-115 ~]$
```

```

1 import socket, hashlib
2
3 def main():
4     s = socket.socket()
5     s.connect(("13.42.11.17", 5000)) # Replace with your actual EC2 server public IP
6     received = s.recv(2048)
7     print("Raw received data:", received)
8     file_data, received_checksum = received.split(b":")
9
10    with open("received.txt", "wb") as f:
11        f.write(file_data)
12
13    checksum = hashlib.md5(file_data).hexdigest()
14    if checksum == received_checksum.decode():
15        print("✓ File received and verified successfully.")
16    else:
17        print("✗ Checksum mismatch.")
18    s.close()
19
20 if __name__ == "__main__":
21     main()
```

## Step-by-step Fix Instructions

### 1. Fix EC2 Security Group Inbound Rules for Server

Go to AWS Console → EC2 → Security Groups → Find the group attached to `13.42.11.17` instance.

Edit Inbound Rules → Add:

Type	Protocol	Port Range	Source
Custom TCP	TCP	5000	<code>0.0.0.0/0</code>
(Optional)	ICMP	All ICMP	<code>0.0.0.0/0</code>

 Save the rules.

### 2. Dockerfile Confirmation

Ensure your `client/Dockerfile` starts like this:

```
Dockerfile  Copy  Edit  
FROM python:3.10-slim  
  
WORKDIR /app  
COPY ./app /app  
  
RUN pip install -r requirements.txt  
  
CMD ["python", "client.py"]
```

And `client.py` now has:

```
python  Copy   
s.connect(("13.42.11.17", 5000)) #  Correct
```

Info Inbound security group rules successfully modified on security group sg-0230b92a8daee8250 | launch-wizard-4

sg-0230b92a8daee8250 - launch-wizard-4

[Actions](#)

**Details**

Security group name	Launch-wizard-4	Security group ID	sg-0230b92a8daee8250	Description	Launch-wizard-4 created 2025-06-12T21:48:45.245Z	VPC ID	vpc-06c1023e3b98f1a3
Owner	675551989964	Inbound rules count:	7 Permission entries	Outbound rules count:	1 Permission entry		

[Inbound rules](#)   [Outbound rules](#)   [Sharing - new](#)   [VPC associations - new](#)   [Tags](#)

**Inbound rules (7)**

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
-	sg-08f15578a60251b87	IPv4	SSH	TCP	22	0.0.0.0/0	-
-	sg-07455177d34b02162	IPv4	All ICMP, IP>6	ICMP	All	0.0.0.0/0	-
-	sg-02e0516e44616ee8	IPv4	Custom TCP	TCP	5000	0.0.0.0/0	-
-	sg-0d530c753d341acaa	IPv4	HTTP	TCP	80	0.0.0.0/0	-
-	sg-04d38814d03fa0a2e	IPv4	Custom TCP	TCP	5000	0.0.0.0/0	-
-	sg-0eb061b049529f5023	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	-

### 3. docker-compose-client.yml

Ensure it looks like this:

```
yaml
version: '3.9'
services:
  client:
    build: ./client
    container_name: devops_client
    volumes:
      - clientvol:/app/data
    ports:
      - "3000:3000" # Optional, not used here
    networks:
      - devnet

  volumes:
    clientvol:

  networks:
    devnet:
```

Don't add `depends_on:` or `links:` since these are two separate EC2 hosts.

#### 4. Run Client Again (From EC2: 35.179.129.150)

```
bash Copy Edit
cd /home/ec2-user
docker-compose -f docker-compose-client.yml down
docker-compose -f docker-compose-client.yml up --build -d
```

Then check logs:

```
bash Copy Edit
docker logs devops_client
```

#### Test it

- If everything works, the client will receive the file, store it as `received.txt`, and print:

```
arduino Copy Edit
✓ File received and verified successfully.
```

#### Final Notes

- If client still fails:
  - Try running `telnet 13.42.11.17 5000` or `nc -zv 13.42.11.17 5000` from the client EC2 to test connectivity.
- You can also temporarily run client without Docker to test:

```
bash Copy Edit
python3 client/app/client.py
```

Let me know what the result of `docker logs devops_client` is now — if it still errors, I'll debug the next step live with you.

```
scp -i "C:/AWSKeys/ziakhan.pem" -r ./client ec2-user@35.179.129.150:/home/ec2-user/
scp -i "C:/AWSKeys/ziakhan.pem" docker-compose-client.yml ec2-user@35.179.129.150:/home/ec2-
user/docker-compose.yml
```

```
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@35.179.129.150
```

```
ec2-user@ip-172-31-36-210:~  
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
  
Try the new cross-platform PowerShell https://aka.ms/pscore6  
  
PS C:\Users\KQ> ssh -i "C:/AWSKeys/ziahan.pem" ec2-user@35.179.129.150  
Last login: Thu Jul 31 22:39:44 2025 from host81-157-30-7.range81-157.btcentralplus.com  
, #  
~\_\ ####_ Amazon Linux 2  
~~ \_\####\ AL2 End of Life is 2026-06-30.  
~~ \|/  
~~ \#/  
~~ V~' '-->  
~~ / A newer version of Amazon Linux is available!  
~~ .-. /  
~~ /_ / Amazon Linux 2023, GA and supported until 2028-03-15.  
~/m/ https://aws.amazon.com/linux/amazon-linux-2023/  
  
[ec2-user@ip-172-31-36-210 ~]$ cd /home/ec2-user  
[ec2-user@ip-172-31-36-210 ~]$ docker-compose -f docker-compose-client.yml down  
p --buildstat /home/ec2-user/docker-compose-client.yml: no such file or directory  
[ec2-user@ip-172-31-36-210 ~]$ docker-compose -f docker-compose-client.yml up --build -d  
stat /home/ec2-user/docker-compose-client.yml: no such file or directory  
[ec2-user@ip-172-31-36-210 ~]$ ls  
client docker-compose.yml  
[ec2-user@ip-172-31-36-210 ~]$ ls client  
app docker-compose-client.yml docker-compose.yml Dockerfile requirements.txt  
[ec2-user@ip-172-31-36-210 ~]$ ls  
client docker-compose.yml  
[ec2-user@ip-172-31-36-210 ~]$
```

## Correct scp Command

Your `docker-compose-client.yml` is in this path:

makefile

 Copy 

D:\Devops13\week8\devopscapstone\

So from Git Bash / WSL / CMD (with SCP available), your command should be:

bash

 Copy 

`scp -i "C:/AWSKeys/ziakhan.pem" "D:/Devops13/week8/devopscapstone/docker-compose-client.yml" ec2-user@35.179.129.150:/home/ec2-user/client/`

Or, if you are already inside that directory in terminal:

bash

 Copy 

`scp -i "C:/AWSKeys/ziakhan.pem" docker-compose-client.yml ec2-user@35.179.129.150:/home/ec2-user/client/`

Make sure the `/home/ec2-user/client/` directory exists on the EC2 instance. If it doesn't, SSH into the instance and create it:

bash

 Copy 

`mkdir -p /home/ec2-user/client`

But your folder structure is:

```
bash                                         ⌂ Copy ⌂ Edit  
/home/ec2-user/client/  
|   └── app/  
|       |   └── client.py  
|       |   └── received.txt  
|   └── Dockerfile  
|   └── requirements.txt  
└── docker-compose-client.yml
```

So there is no `client/` directory inside this folder, hence the error.

**FIX OPTION A: Edit `docker-compose-client.yml` to use `./app` instead of `./client`**

Run:

```
bash                                         ⌂ Copy ⌂ Edit  
nano docker-compose-client.yml
```

Update the `build` section like this:

```
yaml                                         ⌂ Copy ⌂ Edit  
  
services:  
  devops_client:  
    build:  
      context: ./app  
      container_name: devops_client  
      command: python client.py
```

Save and exit (`Ctrl+X`, `Y`, then `Enter`)



```
ec2-user@ip-172-31-36-210:~/client
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\KQ> ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@35.179.129.150
Last login: Thu Jul 31 23:23:57 2025 from host81-157-30-7.range81-157.btcentralplus.com
      _#
      ~\_ ##### Amazon Linux 2
      ~~ \#####\
      ~~ \###| AL2 End of Life is 2026-06-30.
      ~~ \|/
      ~~ V~, '-->
      ~~~ / A newer version of Amazon Linux is available!
      ~~_. _/
      _/ _/ Amazon Linux 2023, GA and supported until 2028-03-15.
      _/m/ https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-172-31-36-210 ~]$ docker-compose -f docker-compose-client.yml up --build -d
stat /home/ec2-user/docker-compose-client.yml: no such file or directory
[ec2-user@ip-172-31-36-210 ~]$ ls
client  docker-compose.yml
[ec2-user@ip-172-31-36-210 ~]$ ls client
app  docker-compose-client.yml  docker-compose.yml  Dockerfile  requirements.txt
[ec2-user@ip-172-31-36-210 ~]$ cd /home/ec2-user/client
[ec2-user@ip-172-31-36-210 client]$ ls
app  docker-compose-client.yml  docker-compose.yml  Dockerfile  requirements.txt
[ec2-user@ip-172-31-36-210 client]$ docker-compose -f docker-compose-client.yml up --build -d
[+] Building 0.0s (1/1) FINISHED
   => [client] Loading build definition from Dockerfile
   => transferring dockerfile: 2B
failed to solve: failed to read dockerfile: open Dockerfile: no such file or directory
[ec2-user@ip-172-31-36-210 client]$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
bea8c8edd4fa        ec2-user-client    "python client.py"   51 minutes ago   Exited (1) 51 minutes ago
[ec2-user@ip-172-31-36-210 client]$ docker logs devops_client
Traceback (most recent call last):
  File "/app/client.py", line 21, in <module>
    main()
  File "/app/client.py", line 5, in main
    s.connect(("server", 5000))
socket.gaierror: [Errno -2] Name or service not known
[ec2-user@ip-172-31-36-210 client]$
```

Thanks for the detailed screenshots — you're right, things are a bit tangled, but we'll cleanly fix it step by step now. Here's what we'll do to fix the **client container build and execution issue**:

## GOAL:

- Use a `separate docker-compose-client.yml`
- Place fixed `client.py`, `Dockerfile`, `requirements.txt` in correct path
- Build and run the container cleanly

## Step-by-Step Cleanup & Setup

### Step 1: Clean old files

SSH into client EC2 and run:

```
bash Copy Edit
rm -rf ~/client
rm -f ~/docker-compose-client.yml
```

### Step 2: Re-copy fresh files from your local Windows PC

Your local folder contains:

- `client/` folder with:
  - `client.py`
  - `Dockerfile`
  - `requirements.txt`
- `docker-compose-client.yml` file

Run the following `scp` commands from your Windows terminal (PowerShell):

```
bash Copy Edit
scp -i "C:/AWSKeys/ziakhan.pem" -r D:/Devops13/week8/devopscapstone/client ec2-user@35.179.129.150:/home/ec2-user/
scp -i "C:/AWSKeys/ziakhan.pem" D:/Devops13/week8/devopscapstone/docker-compose-client.yml ec2-user@35.179.129.150:/home/ec2-user/
```

Make sure paths match your actual folder!

```
scp -i "C:/AWSKeys/ziakhan.pem" -r D:/Devops13/week8/devopscapstone/client ec2-user@35.179.129.150:/home/ec2-user/
```

```
scp -i "C:/AWSKeys/ziakhan.pem" D:/Devops13/week8/devopscapstone/docker-compose-client.yml  
ec2-user@35.179.129.150:/home/ec2-user/
```

### ✖ Problem

In your Dockerfile, you're already inside `./app`, so `COPY ./app /app` is wrong because it tries to copy a folder that doesn't exist inside the Docker build context (`client/app/app`).

### ✓ Solution

#### ✍ Step 1: Fix the Dockerfile

Open `client/app/Dockerfile` and replace:

```
dockerfile
```

Copy Edit

```
COPY ./app /app
```

with:

```
dockerfile
```

Copy Edit

```
COPY . /app
```

This tells Docker to copy all contents from the current context (`client/app/`) into the `/app` folder inside the container.

#### 🛠 Step 2: Clean Previous Build (optional but helpful)

```
bash
```

Copy Edit

```
docker-compose -f docker-compose-client.yml down --volumes  
docker system prune -f
```

#### 🚀 Step 3: Build and Run Again

Make sure you are inside the `~/client/` folder:

```
bash
```

Copy Edit

```
cd ~/client  
docker-compose -f docker-compose-client.yml up --build -d
```

```
MINGW64/d/Devops13/week8/devopscapstone
$ ec2-user/client/
$ docker-compose-client.yml
100% 275 19.6KB/s 00:00

QDESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ # Upload app files

QDESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziakhan.pem" D:/Devops13/week8/devopscapstone/client/app/* ec2-user@35.179.129.150:/home/ec2-user/client/app/
client.py
100% 631 49.5KB/s 00:00
received.txt
100% 1024 78.8KB/s 00:00

QDESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ 

QDESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ # Upload docker-compose file

QDESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziakhan.pem" D:/Devops13/week8/devopscapstone/docker-compose-client.yml ec2-user@35.179.129.150:/home/ec2-user/client/
docker-compose-client.yml
100% 275 22.6KB/s 00:00

QDESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziakhan.pem" D:/Devops13/week8/devopscapstone/client/Dockerfile ec2-user@35.179.129.150:/home/ec2-u
ser/client/app/
Dockerfile: stat local "D:/Devops13/week8/devopscapstone/client/app/Dockerfile": No such file or directory

QDESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziakhan.pem" D:/Devops13/week8/devopscapstone/client/Dockerfile ec2-user@35.179.129.150:/home/ec2-u
ser/client/app/
Dockerfile
100% 126 8.2KB/s 00:00

QDESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/devopscapstone
$ scp -i "C:/AWSKeys/ziakhan.pem" D:/Devops13/week8/devopscapstone/client/Dockerfile ec2-user@35.179.129.150:/home/ec2-u
ser/client/app/
Dockerfile
100% 122 8.4KB/s 00:00

QDESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/devopscapstone
```

## Step 2: Clean Previous Build (optional but helpful)

bash

 Copy 

```
docker-compose -f docker-compose-client.yml down --volumes  
docker system prune -f
```

## Step 3: Build and Run Again

Make sure you are inside the `~/client/` folder:

bash

 Copy 

```
cd ~/client  
docker-compose -f docker-compose-client.yml up --build -d
```

## Step 4: Verify Container Is Running

bash

 Copy 

```
docker ps
```

You should see a running container with the name `devops_client`.

## Step 5: Test the App

If your app listens on a socket or makes outbound calls, check logs:

bash

 Copy 

```
docker logs devops_client
```

If you're still facing issues, share the content of:

- `client/app/Dockerfile`
- `client/app/client.py`



## Option 2: Change build context in `docker-compose-client.yml` to `./client`:

Edit the file:

```
bash Copy Edit
nano docker-compose-client.yml
```

Update this:

```
yaml Copy Edit
build:
  ./app
```

To this:

```
yaml Copy Edit
build:
  context: .
  dockerfile: ./app/Dockerfile
```

This way, Docker gets the full context (including `requirements.txt`), and uses the Dockerfile from `./app`.

Then run:

```
bash Copy Edit
cd ~/client
docker-compose -f docker-compose-client.yml up --build -d
```

---

## Bonus Tip: Clean up dangling images

If you want to clear failed image builds:

```
bash Copy Edit
docker image prune -f
```

```
ec2-user@ip-172-31-42-115:~/server
[+] Building 0.0s (0/0)
unable to prepare context: path "/home/ec2-user/server/server" not found
[ec2-user@ip-172-31-42-115 server]$ cd ~
[ec2-user@ip-172-31-42-115 ~]$ ls
client docker-compose.yml server
[ec2-user@ip-172-31-42-115 ~]$ cd ~/server # or wherever your docker-compose-server.yml is
[ec2-user@ip-172-31-42-115 server]$ docker-compose -f docker-compose-server.yml down
docker-compose -f docker-compose-server.yml up --build -d
[ec2-user@ip-172-31-42-115 server]$ docker-compose -f docker-compose-server.yml up --build -d
[+] Building 0.0s (0/0)
unable to prepare context: path "/home/ec2-user/server/server" not found
[ec2-user@ip-172-31-42-115 server]$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[ec2-user@ip-172-31-42-115 server]$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
5e3d5fbbe3a ec2-user-server "python server.py" 3 hours ago Exited (0) 23 minutes ago
[ec2-user@ip-172-31-42-115 server]$ curl http://checkip.amazonaws.com
13.42.11.17
[ec2-user@ip-172-31-42-115 server]$ clear
[ec2-user@ip-172-31-42-115 server]$ ls
app docker-compose-server.yml docker-compose.yml Dockerfile requirements.txt
[ec2-user@ip-172-31-42-115 server]$ ls app
file.txt server.py
[ec2-user@ip-172-31-42-115 server]$ cat server.py
cat: server.py: No such file or directory
[ec2-user@ip-172-31-42-115 server]$ ls
app docker-compose-server.yml docker-compose.yml Dockerfile requirements.txt
[ec2-user@ip-172-31-42-115 server]$ cd app
[ec2-user@ip-172-31-42-115 app]$ ls
file.txt server.py
[ec2-user@ip-172-31-42-115 app]$ cat server.py
import socket, hashlib, os

def create_file():
    data = os.urandom(1024) # 1KB random binary
    with open("file.txt", "wb") as f:
        f.write(data)
    checksum = hashlib.md5(data).hexdigest()
    return data, checksum

def main():
    s = socket.socket()
    s.bind(("0.0.0.0", 5000))
    s.listen(1)
    print("Server is waiting for connection...")
    conn, addr = s.accept()
    print(f"Connected to {addr}")

    data, checksum = create_file()
    conn.sendall(data + b"::" + checksum.encode())
    conn.close()
    print("File and checksum sent.")

if __name__ == "__main__":
    main()
[ec2-user@ip-172-31-42-115 app]$ cat dockerfile
cat: dockerfile: No such file or directory
[ec2-user@ip-172-31-42-115 app]$ cat Dockerfile
cat: Dockerfile: No such file or directory
[ec2-user@ip-172-31-42-115 app]$ cd ~
[ec2-user@ip-172-31-42-115 ~]$ ls
client docker-compose.yml server
[ec2-user@ip-172-31-42-115 ~]$ cd server
[ec2-user@ip-172-31-42-115 server]$ ls
app docker-compose-server.yml docker-compose.yml Dockerfile requirements.txt
[ec2-user@ip-172-31-42-115 server]$
```

```
[ec2-user@ip-172-31-36-210:~/client]
+} Running 11
  Container devops_client  Started
[ec2-user@ip-172-31-36-210 client]$ clear
[ec2-user@ip-172-31-36-210 client]$ ls
app  docker-compose-client.yml
[ec2-user@ip-172-31-36-210 client]$ cd ~
[ec2-user@ip-172-31-36-210 ~]$ ls
client  docker-compose-client.yml  docker-compose.yml
[ec2-user@ip-172-31-36-210 ~]$ ls docker-compose-client.yml
docker-compose-client.yml
[ec2-user@ip-172-31-36-210 ~]$ cat docker-compose-client.yml
version: '3.9'
services:
  client:
    build: ./app
    container_name: devops_client
    volumes:
      - clientvol:/app/data
    ports:
      - "3000:3000" # Optional, not used here
    networks:
      - devnet

volumes:
  clientvol:

networks:
  devnet:
[ec2-user@ip-172-31-36-210 ~]$ ls
client  docker-compose-client.yml  docker-compose.yml
[ec2-user@ip-172-31-36-210 ~]$ ls client
app  docker-compose-client.yml
[ec2-user@ip-172-31-36-210 ~]$ cd client
[ec2-user@ip-172-31-36-210 client]$ ls
app  docker-compose-client.yml
[ec2-user@ip-172-31-36-210 client]$ cat docker-compose-client.yml
version: '3.9'
services:
  client:
    build: ./app
    container_name: devops_client
    volumes:
      - clientvol:/app/data
    ports:
      - "3000:3000" # Optional, not used here
    networks:
      - devnet

volumes:
  clientvol:

networks:
  devnet:
[ec2-user@ip-172-31-36-210 client]$
```

```
ec2-user@ip-172-31-36-210:~/client/app
- clientvol:/app/data
ports:
- "3000:3000" # Optional, not used here
networks:
- devnet

volumes:
clientvol:

networks:
devnet:
[ec2-user@ip-172-31-36-210 client]$ ls
app docker-compose-client.yml
[ec2-user@ip-172-31-36-210 client]$ cd app
[ec2-user@ip-172-31-36-210 app]$ ls
client.py Dockerfile received.txt requirements.txt
[ec2-user@ip-172-31-36-210 app]$ cat dockerfile
cat: dockerfile: No such file or directory
[ec2-user@ip-172-31-36-210 app]$ ls
client.py Dockerfile received.txt requirements.txt
[ec2-user@ip-172-31-36-210 app]$ cat Dockerfile
FROM python:3.10-slim

WORKDIR /app
COPY . /app

RUN pip install -r requirements.txt

CMD ["sh", "-c", "python client.py && sleep 3600"]
[ec2-user@ip-172-31-36-210 app]$ ls
client.py Dockerfile received.txt requirements.txt
[ec2-user@ip-172-31-36-210 app]$ cat client.py
import socket, hashlib

def main():
    s = socket.socket()
    s.connect(("13.42.11.17", 5000)) # Replace with your actual EC2 server public IP
    received = s.recv(2048)
    print("Raw received data:", received)
    file_data, received_checksum = received.split(b":")
    with open("received.txt", "wb") as f:
        f.write(file_data)

    checksum = hashlib.md5(file_data).hexdigest()
    if checksum == received_checksum.decode():
        print("@ File received and verified successfully.")
    else:
        print("@ Checksum mismatch.")
    s.close()

if __name__ == "__main__":
    main()
[ec2-user@ip-172-31-36-210 app]$
```

Thanks for sharing the updated IPs. Since your EC2 instances now have new public IPs, here's exactly where you must update them:

---

**Where to Update the IP Address:**

**1. Inside `client.py`**

Update this line in your `client.py`:

```
python
```

Copy  Edit

```
s.connect(("13.42.11.17", 5000)) # old IP
```

to:

```
python
```

Copy  Edit

```
s.connect(("18.168.221.107", 5000)) # new server IP
```

Then rebuild the client Docker container:

```
bash
```

Copy  Edit

```
docker-compose -f docker-compose-client.yml down  
docker-compose -f docker-compose-client.yml up --build -d
```

You must rebuild because the IP is hardcoded inside the Python script.

---

I am going to create everything from the beginning /start here: like creating new folder and client/server files, then I will move them across to instances through scp server:

## PART 1: Clean up both EC2 instances

SSH into both instances (one by one):

```
bash Copy Edit  
  
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@18.168.221.107 # Server  
ssh -i "C:/AWSKeys/ziakhan.pem" ec2-user@18.133.196.212 # Client
```

Then on each:

```
bash Copy Edit  
  
docker ps -a          # List containers  
docker stop $(docker ps -aq) # Stop all running containers  
docker rm $(docker ps -aq) # Remove all containers  
docker image prune -af    # Remove all images  
rm -rf client server     # Remove old folders if exist
```

---

## PART 2: SCP file transfer from local machine to instances

### From your PC to EC2 server (18.168.221.107)

Assuming you are in the local `DevOpsCapstone` directory (which contains `server/`):

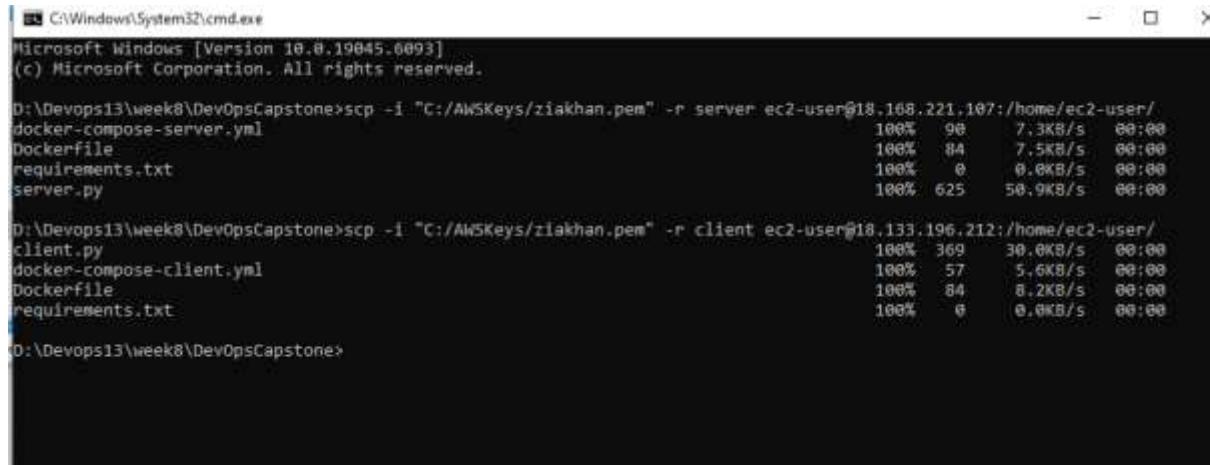
```
bash Copy Edit  
  
scp -i "C:/AWSKeys/ziakhan.pem" -r server ec2-user@18.168.221.107:/home/ec2-user/
```

### From your PC to EC2 client (18.133.196.212)

```
bash Copy Edit  
  
scp -i "C:/AWSKeys/ziakhan.pem" -r client ec2-user@18.133.196.212:/home/ec2-user/
```

```
scp -i "C:/AWSKeys/ziakhan.pem" -r server ec2-user@18.168.221.107:/home/ec2-user/
```

```
scp -i "C:/AWSKeys/ziakhan.pem" -r client ec2-user@18.133.196.212:/home/ec2-user/
```



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The command line shows two separate SCP commands being run from a directory 'D:\Devops13\week8\DevOpsCapstone'.

The first command transfers files to a server at IP 18.168.221.107:

```
D:\Devops13\week8\DevOpsCapstone>scp -i "C:/AWSKeys/ziakhan.pem" -r server ec2-user@18.168.221.107:/home/ec2-user/
  docker-compose-server.yml          100%   96    7.3KB/s  00:00
  Dockerfile                         100%   84    7.5KB/s  00:00
  requirements.txt                   100%     0    0.0KB/s  00:00
  server.py                          100%  625   50.9KB/s  00:00
```

The second command transfers files to a client at IP 18.133.196.212:

```
D:\Devops13\week8\DevOpsCapstone>scp -i "C:/AWSKeys/ziakhan.pem" -r client ec2-user@18.133.196.212:/home/ec2-user/
  client.py                          100%  369   30.0KB/s  00:00
  docker-compose-client.yml          100%   57    5.6KB/s  00:00
  Dockerfile                         100%   84    8.2KB/s  00:00
  requirements.txt                   100%     0    0.0KB/s  00:00
```

Both sets of files transferred successfully with 100% completion and zero errors.

## CURRENT STATUS

### Server

- Running on EC2 IP: 18.168.221.107
- Port 5000 is listening (inside container) 
- docker-compose for server ran successfully 

### Client

- Running on EC2 IP: 18.133.196.212
  - Docker container starts and exits immediately 
  - You see: client\_devops\_client\_1 exited with code 0
- 

## PROBLEM 1: Client Exits Immediately

This is expected because your `client.py` connects, receives data once, saves it, then exits. It's not a long-running service. So this is not an error, but it means you won't see the container in `docker ps` after it exits.

 **Solution:** That's fine for now. We will verify it received the file properly (`received_file.txt`) to be sure it worked.

---

## RESTART SERVER CONTAINER

After editing:

```
bash Copy Edit

# Navigate to the server folder
cd ~/server

# Rebuild and restart the container
docker compose -f docker-compose-server.yml down
docker compose -f docker-compose-server.yml up --build -d
```

Verify again:

```
bash Copy Edit

docker ps
```

Then check if the server is listening correctly:

```
bash Copy Edit

sudo lsof -i -P -n | grep LISTEN
```

You should see:

```
nginx Copy Edit

python3 ... LISTEN 0.0.0.0:5000
```

```

[+] Building 0.7s (8/8) FINISHED
--> [internal] load build definition from Dockerfile
--> => transferring dockerfile: 121B
--> [internal] load metadata for docker.io/library/python:3.10-slim
--> [internal] load .dockerignore
--> => transferring context: 2B
--> [1/3] FROM docker.io/library/python:3.10-slim@sha256:81f1cd83778d54ecfdbddcc52c2125fce674c14a1d976dfdf8f65dc07 0.0s
--> [internal] load build context
--> => transferring context: 31B
--> CACHED [2/3] WORKDIR /app
--> CACHED [3/3] COPY server.py .
--> exporting to image
--> => exporting layers
--> => writing image sha256:783b3ed587d850f791895b71c477dbf50c2b7c374f5a34ea39a76184751e2586
--> => naming to docker.io/library/server_devops_server
Creating server_devops_server_1 ... done
[ec2-user@ip-172-31-42-115 server]$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
9abfc2102ede server_devops_server "python server.py" 9 seconds ago Up 8 seconds 0.0.0.0:5000->5000/tcp, :::5000
[ec2-user@ip-172-31-42-115 server]$ sudo lsof -i -p -n | grep LISTEN
lsof: illegal process ID: -n
lsof 4.87
latest revision: ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/
latest FAQ: ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/FAQ
latest man page: ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/lsof_man
usage: [-?abbhKlnNoOPRtUvVX] [+|-c c] [+|-d s] [+D D] [+|-f[G]] [+|-e s]
[-F [f]] [-g [s]] [-i [i]] [+|-L [l]] [+m [m]] [+|-M] [-o [o]] [-p s]
[+|-r [t]] [-s [p:s]] [-S [t]] [-T [t]] [-u s] [+|-w] [-s [f1]] [--] [names]
Use the ``-h'' option to get more help information.
[ec2-user@ip-172-31-42-115 server]$ sudo lsof -i -P -n | grep LISTEN
rpcbind 2536 rpc 8u IPv4 15093 0t0 TCP *:111 (LISTEN)
rpcbind 2536 rpc 11u IPv6 15096 0t0 TCP *:111 (LISTEN)
master 2942 root 13u IPv4 16685 0t0 TCP 127.0.0.1:25 (LISTEN)
sshd 2988 root 3u IPv4 17847 0t0 TCP *:22 (LISTEN)
sshd 2988 root 4u IPv6 17856 0t0 TCP *:22 (LISTEN)
container 3239 root 12u IPv4 19878 0t0 TCP 127.0.0.1:40077 (LISTEN)
docker-pr 5069 root 4u IPv4 28956 0t0 TCP *:5000 (LISTEN)
docker-pr 5074 root 4u IPv6 28962 0t0 TCP *:5000 (LISTEN)
[ec2-user@ip-172-31-42-115 server]$

```

## VERIFY PORT ALLOWANCE (already done)

Make sure port 5000 is allowed in both security groups:

- Launch Wizard of **server EC2**
- Launch Wizard of **client EC2**

Your screenshots show it's done. 

## FINAL FILE CHECK

Also, make sure you're writing and reading the correct files:

- Server must create `file.txt` 
- Client must read and write `received_file.txt` 

```

❯ ec2-user@ip-172-31-36-210:~/client
File "client.py", line 5, in main
    s.connect(("18.168.221.107", 5000)) # Replace this before deploying
ConnectionRefusedError: [Errno 111] Connection refused
[ec2-user@ip-172-31-36-210 client]$ ls client/received_file.txt
ls: cannot access client/received_file.txt: No such file or directory
[ec2-user@ip-172-31-36-210 client]$ sudo lsof -i -P -n | grep LISTEN
pcbind   2508     rpc    8u  IPv4  14655      0t0  TCP *:111  (LISTEN)
pcbind   2508     rpc   11u  IPv6  14658      0t0  TCP *:111  (LISTEN)
master    2942    root   13u  IPv4  16715      0t0  TCP 127.0.0.1:25  (LISTEN)
sshd     2985    root    3u  IPv4  17867      0t0  TCP *:22  (LISTEN)
sshd     2985    root    4u  IPv6  17876      0t0  TCP *:22  (LISTEN)
container 3303    root   12u  IPv4  20423      0t0  TCP 127.0.0.1:40387  (LISTEN)
[ec2-user@ip-172-31-36-210 client]$ ls
client.py  docker-compose-client.yml  Dockerfile  requirements.txt
[ec2-user@ip-172-31-36-210 client]$ cat client.py
import socket

def main():
    s = socket.socket()
    s.connect(("18.168.221.107", 5000)) # Replace this before deploying
    data = s.recv(2048)
    s.close()
    print("Received:", data[:32], "...") # First 32 bytes + checksum
    with open("received_file.txt", "wb") as f:
        f.write(data.split(b"\r\n")[0])

if __name__ == "__main__":
    main()
[ec2-user@ip-172-31-36-210 client]$ python3 client.py
Received: b'\xc9\xd7\xec_\xe4\xe9G\xeasC-\x9d\x97\x9d\x1b\xd2Y\xf1Bi\xd5\\ ^\x82LX\x95%\x08\xe0n' ...
[ec2-user@ip-172-31-36-210 client]$ ls
client.py  docker-compose-client.yml  Dockerfile  received_file.txt  requirements.txt
[ec2-user@ip-172-31-36-210 client]$
```

This is finally done.



## Ready for Day 6?

We'll start with:

- ⌚ Creating custom VPC & subnet (for private communication)
- 🔒 Security Group tightening
- 📦 Prepare for monitoring (Day 7: Grafana)

Once confirmed, we can proceed to:

- ✓ Nginx Reverse Proxy setup
- ✓ GitHub Actions for CI/CD
- ✓ Monitoring with Grafana/Prometheus

1. Setting up Nginx reverse proxy.
2. Implementing CI/CD via GitHub Actions.
3. Adding Prometheus + Grafana monitoring.

Now next step is to do:

Setting up GitHub Actions for deployment? (CI/CD steps)

📊 Deploying monitoring (Grafana/Prometheus)?

Use Grafana + Prometheus or ELK to monitor logs, resource usage.

## ✓ Part 1: Setting Up GitHub Actions for Deployment

### ⌚ Goal:

Push code to GitHub → Automatically deploy to your EC2 instances (server/client) via Docker Compose.

### 📦 Prerequisites:

1.  Server and client folders on your local with:
  - `Dockerfile`
  - `docker-compose.yml`
  - `requirements.txt`
  - `server.py` or `client.py`
2.  Code already tested and working on EC2
3.  A GitHub repo with the full project (`client/`, `server/`)

## 🔒 Step 1: Set Up SSH Access in GitHub

1. Generate SSH key (local machine):

```
bash Copy Edit
ssh-keygen -t rsa -b 4096 -C "github-ec2" -f ~/.ssh/github-actions-key
```

2. Copy the private key contents:

```
bash Copy Edit
cat ~/.ssh/github-actions-key
```

3. Go to your GitHub Repo → Settings → Secrets → Actions → `New repository secret`

- Name: `EC2_SSH_KEY`
- Value: *(Paste the full private key)*

4. Add the public key to your EC2 authorized keys:

On your EC2 (server and client both):

```
bash Copy Edit
echo "your-public-key" >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

## Step 2: Create GitHub Action Workflow ( .github/workflows/deploy-server.yml )

Here's an example workflow for the `server` EC2:

```
yaml
name: Deploy Server to EC2

on:
  push:
    paths:
      - 'server/**'
  branches: [ main ]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Deploy to EC2
        uses: appleboy/ssh-action@v1.0.0
        with:
          host: ${{ secrets.SERVER_IP }}
          username: ec2-user
          key: ${{ secrets.EC2_SSH_KEY }}
          script: |
            cd server
            docker-compose down || true
            docker-compose up --build -d
```

Do the same for `client` with another YAML and update:

- `host`
- folder path
- environment variables if needed

You'll also need to create another secret:

- `SERVER_IP` : your EC2 public IP

Repeat for:

- `CLIENT_IP`

---

The screenshot shows a GitHub repository page for 'CICD-Actions-Demo'. The top navigation bar includes links for 'Home', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The repository name 'CICD-Actions-Demo' is displayed in the title bar. The main content area includes sections for 'Set up GitHub Copilot', 'Add collaborators to this repository', and 'Quick setup — if you've done this kind of thing before'. Below these are two code snippets: one for '...or create a new repository on the command line' and another for '...or push an existing repository from the command line'.

```
cd ..\CICD-Actions-Demo>> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/ziakhan218/CICD-Actions-Demo.git
git push -u origin main
```

```
git remote add origin https://github.com/ziakhan218/CICD-Actions-Demo.git
git branch -M main
git push -u origin main
```

The screenshot shows a Windows Command Prompt window with the title 'Select C:\Windows\System32\cmd.exe'. The command line shows the user cloning a GitHub repository named 'CICD-Actions-Demo' into a local directory. The output indicates that the repository is empty.

```
D:\Devops13\week8\DevOpsCapstone>git clone https://github.com/ziakhan218/CICD-Actions-Demo.git
Cloning into 'CICD-Actions-Demo'...
warning: You appear to have cloned an empty repository.

D:\Devops13\week8\DevOpsCapstone>git
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--no-lazy-fetch]
           [--no-optional-locks] [--no-advice] [--bare] [--git-dir=<path>]
           [- -work-tree=<path>] [--namespace=<name>] [--config-env=<name>=<envvar>]
           <command> [<args>]
```

```
git clone https://github.com/ciathan21/CICD-Actions-Demo.git
cd CICD-Actions-Demo
git pull
git status
git remote add origin https://github.com/ciathan21/CICD-Actions-Demo.git
git push -u origin main
git push -u origin main
```

## ✖ Issues to Fix / Improve

### 1. 🛡 SSH Key Save Failed

In this command:

```
bash
ssh-keygen -t rsa -b 4096 -C "github-ec2" -f ~/.ssh/github-actions-key
Copy Edit
```

You got the error:

```
vbnet
Saving key " ~/.ssh/github-actions-key" failed: No such file or directory
Copy Edit
```

### Fix:

Before saving, run:

```
bash
mkdilr -p ~/.ssh
Copy Edit
```

```
MINGW64/d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 234 bytes | 234.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/ziaikhan218/CICD-Actions-Demo.git
 * [new branch]    main -> main
branch 'main' set up to track 'origin/main'.

MINGW64/d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ mkdir -p ~/.ssh

MINGW64/d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ ssh-keygen -t rsa -b 4096 -C "github-ec2" -f ~/.ssh/github-actions-key
Generating public/private rsa key pair.
Enter passphrase for "/c/Users/KQ/.ssh/github-actions-key" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/KQ/.ssh/github-actions-key
Your public key has been saved in /c/Users/KQ/.ssh/github-actions-key.pub
The key fingerprint is:
SHA256:2Qnf0wJL4WUWFTmI0jB0zqmuypyFX2uohJavdffrwA github-ec2
The key's randomart image is:
----[RSA 4096]----+
| 00++o*.
| o.o--o
| ...*o..
| o... o
| .ES.+ o
| ... o o ...
| : o. .=.
| +...+.. +
| 0 0++o+=,00.
----[SHA256]----+
MINGW64/d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$
```

## 2. Folder Structure (Important)

Your workflows (`deploy-client.yml` and `deploy-server.yml`) are trying to `cd server`, but they're already inside `/CICD-Actions-Demo`.

Fix: Copy your actual code into the repo folder.

Final layout should be:

```
pgsql
CICD-Actions-Demo/
├── -github/
│   └── workflows/
│       ├── deploy-client.yml
│       └── deploy-server.yml
└── client/
    ├── Dockerfile
    │   └── client.py / index.html etc.
└── server/
    ├── Dockerfile
    └── server.py
├── docker-compose-client.yml
└── README.md
```

Move your existing `client` and `server` folders into the `CICD-Actions-Demo` folder.

3.  `deploy-client.yml` and `deploy-server.yml`

Both scripts should `cd` into the correct folder:

 `deploy-client.yml`

yaml

 Copy  Edit

```
- name: Deploy to EC2
  uses: appleboy/ssh-action@v1.0.0
  with:
    host: ${{ secrets.CLIENT_IP }}
    username: ec2-user
    key: ${{ secrets.EC2_SSH_KEY }}
    script: |
      cd ~/client      # or wherever your app is
      docker-compose down || true
      docker-compose up --build -d
```

 `deploy-server.yml`

yaml

 Copy  Edit

```
- name: Deploy to EC2
  uses: appleboy/ssh-action@v1.0.0
  with:
    host: ${{ secrets.SERVER_IP }}
    username: ec2-user
    key: ${{ secrets.EC2_SSH_KEY }}
    script: |
      cd ~/server      # adjust path
      docker-compose down || true
      docker-compose up --build -d
```

Note: These paths must match where you SCP'd your s earlier.

## What to Do Next

### Step-by-Step Fix and Final Setup:

1. Fix SSH key issue:

```
bash Copy Edit
mkdir -p ~/.ssh
ssh-keygen -t rsa -b 4096 -C "github-ec2" -f ~/.ssh/github-actions-key
```

2. Ensure `.github/workflows` folder exists inside `CICD-Actions-Demo`.

3. Move `client` and `server` folders inside `CICD-Actions-Demo`.

4. Commit and push to GitHub:

```
bash Copy Edit
git add .
git commit -m "Added client and server with workflows"
git push origin main
```

5. Push to trigger:
  - Change something in `client/**` to trigger `deploy-client.yml`
  - Change something in `server/**` to trigger `deploy-server.yml`

```
KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ ssh-keygen -t rsa -b 4096 -C "github-ec2" -f ~/.ssh/github-actions-key
Generating public/private rsa key pair.
Enter passphrase for "/c/Users/KQ/.ssh/github-actions-key" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/KQ/.ssh/github-actions-key
Your public key has been saved in /c/Users/KQ/.ssh/github-actions-key.pub
The key fingerprint is:
SHA256:2QNf0wUL4WUwFtmI0jB0zqnuyppyX2uoHJovdffrwA github-ec2
The key's randomart image is:
+---[RSA 4096]---+
| oo++o*.
| o..==o
| ...*o..
| o=... o
| .. ES.+ o
| .... o o . .
| . o.. .= .
| . +...+.+
| o o++o+o=.oo.
+---[SHA256]---+
KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ mkdir -p ~/.ssh

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ ssh-keygen -t rsa -b 4096 -C "github-ec2" -f ~/.ssh/github-actions-key
Generating public/private rsa key pair.
/c/Users/KQ/.ssh/github-actions-key already exists.
Overwrite (y/n)? n

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$
```

```
MINGW64:/d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo
(use "git add <file>..." to include in what will be committed)
.github/
client/
docker-compose-client.yml
docker-compose-server.yml
server/

nothing added to commit but untracked files present (use "git add" to track)

QDESKTOP-CCHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git add .

QDESKTOP-CCHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git commit -m "Added client and server with workflows"
[master baf9691] Added client and server with workflows
12 files changed, 75 insertions(+)
create mode 100644 .github/workflows/deploy-client.yml
create mode 100644 .github/workflows/deploy-server.yml
create mode 100644 client/Dockerfile
create mode 100644 client/client.py
create mode 100644 client/docker-compose-client.yml
create mode 100644 client/requirements.txt
create mode 100644 docker-compose-client.yml
create mode 100644 docker-compose-server.yml
create mode 100644 server/Dockerfile
create mode 100644 server/docker-compose-server.yml
create mode 100644 server/requirements.txt
create mode 100644 server/server.py

QDESKTOP-CCHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git push origin main
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 4 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (15/15), 1.99 KiB | 510.00 KiB/s, done.
Total 15 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/ziakhhan218/CICD-Actions-Demo.git
 f1ae207..baf9691  main -> main

QDESKTOP-CCHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
```

## Trigger GitHub Action: deploy-client.yml

### What triggers it:

- Any change inside the `client/` folder (e.g. `client/client.py`, `client/requirements.txt`, etc.)

### How to do it:

- On your local machine, go to your repo folder:

```
bash Copy Edit
cd /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo/client
```

- Make any edit in a file inside the `client` folder. For example:

```
bash Copy Edit
echo "# Triggering deployment" >> client.py
```

- Add and commit the change:

```
bash Copy Edit
git add client/client.py
git commit -m "Updated client to trigger workflow"
git push origin main
```

- Done! This will automatically run the `deploy-client.yml` GitHub Action and deploy to your **client EC2 instance**.

```
KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   client/requirements.txt

no changes added to commit (use "git add" and/or "git commit -a")

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git add client/client.py

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git commit -m "Updated client to trigger workflow"
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   client/requirements.txt

no changes added to commit (use "git add" and/or "git commit -a")

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git push origin main
Everything up-to-date

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$
```

## Trigger GitHub Action: deploy-server.yml

What triggers it:

- Any change inside the `server/` folder (e.g. `server/server.py`, `server/Dockerfile`, etc.)

How to 

- Go to the server folder:

```
bash Copy Edit  
cd /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo/server
```

- Make any change to a file in the server directory:

```
bash Copy Edit  
echo "# Triggering server deploy" >> server.py
```

- Commit and push the change:

```
bash Copy Edit  
git add server/server.py  
git commit -m "Update server to trigger workflow"  
git push origin main
```

- Done! This will trigger the `deploy-server.yml` GitHub Action and deploy to your **server EC2 instance**.

```

cd MINGW64:/d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo
KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git push origin main
Everything up-to-date

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git add server/server.py

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git commit -m "Update server to trigger workflow"
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   client/requirements.txt
      modified:   server/requirements.txt

no changes added to commit (use "git add" and/or "git commit -a")

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git push origin main
Everything up-to-date

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git add .

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git commit -m "Update server and client req.txt file"
[main 3250065] Update server and client req.txt file
 2 files changed, 2 insertions(+)

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$ git push origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 515 bytes | 515.00 KiB/s, done.
Total 6 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/ziakhan218/CICD-Actions-Demo.git
  baf9691..3250065  main -> main

KQ@DESKTOP-C0CHCRF MINGW64 /d/Devops13/week8/DevOpsCapstone/CICD-Actions-Demo (main)
$
```

```

name: Deploy Client

on:
  push:
    paths:
      - 'client/**'

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Deploy to EC2
        uses: appleboy/ssh-action@v1.0
        with:
          host: ${{ secrets.CLIENT_IP }}
          username: ec2-user
          key: ${{ secrets.EC2_SSH_KEY }}
        script:
          cd ~/client
          docker-compose -f docker-compose-client.yml down || true
          docker-compose -f docker-compose-client.yml up --build -d

```

The screenshot shows a GitHub Actions workflow run. The workflow has a single job named 'deploy' which succeeded in 12s. The steps in the job are:

- Set up job
- Build appleboy/ssh-action@v1.0.0
- Checkout code
- Deploy to EC2
- Post Checkout code
- Complete job

```
[ec2-user@ip-172-31-42-115:~] Select ec2-user@ip-172-31-42-115:~/server
[ec2-user@ip-172-31-42-115 server]$ [ec2-user@ip-172-31-42-115 server]$ ls
docker-compose-server.yml Dockerfile requirements.txt server.py
[ec2-user@ip-172-31-42-115 server]$ cd
[ec2-user@ip-172-31-42-115 ~]$ git
-bash: git: command not found
[ec2-user@ip-172-31-42-115 ~]$ ls
server
[ec2-user@ip-172-31-42-115 ~]$ cd server
[ec2-user@ip-172-31-42-115 server]$ ls
docker-compose-server.yml Dockerfile requirements.txt server.py
[ec2-user@ip-172-31-42-115 server]$ ls
docker-compose-server.yml Dockerfile requirements.txt server.py
[ec2-user@ip-172-31-42-115 server]$ cat requirements.txt
[ec2-user@ip-172-31-42-115 server]$ cat requirements.txt
[ec2-user@ip-172-31-42-115 server]$ cd
[ec2-user@ip-172-31-42-115 ~]$ ls
server
[ec2-user@ip-172-31-42-115 ~]$ cd server
[ec2-user@ip-172-31-42-115 server]$ ls
docker-compose-server.yml Dockerfile requirements.txt server.py
[ec2-user@ip-172-31-42-115 server]$ cat docker-compose-server.yml
version: '3'
services:
  devops_server:
    build: .
    ports:
      - "5000:5000"
[ec2-user@ip-172-31-42-115 server]$ cat requirements.txt
[ec2-user@ip-172-31-42-115 server]$ nano requirements.txt
[ec2-user@ip-172-31-42-115 server]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
9a01b3692010        server_devops_server   "python server.py"   3 minutes ago     Up 3 minutes   0.0.0.0:5000->5000/tcp, :::5000
0->5000/tcp         server_devops_server_1
[ec2-user@ip-172-31-42-115 server]$
```

## ✓ What You've Completed So Far

Module/Component	Status	Details
1. Project Structure Setup	✓ Done	<code>client/</code> , <code>server/</code> , <code>docker-compose-client.yml</code> , <code>docker-compose-server.yml</code> created
2. GitHub Repository Setup	✓ Done	Public repo <code>CICD-Actions-Demo</code> created with workflows and source code
3. SSH Key Generation	✓ Done	<code>github-actions-key</code> generated and used for EC2 deployments
4. EC2 Server & Client Deployment	✓ Done	SSH from GitHub Actions works via <code>appleboy/ssh-action</code>
5. GitHub Actions CI/CD	✓ Done	Triggers on file changes in <code>client/**</code> and <code>server/**</code> , deploys to EC2
6. Docker Compose Deployment	✓ Done	<code>docker-compose-server.yml</code> and <code>docker-compose-client.yml</code> used on EC2

You now have automatic deployment triggered by code changes. 🎉

## ➡ What's Left to Do (Remaining Modules)

### ✳️ 7. Add Monitoring Stack — Grafana + Prometheus

Task	Description
🛠 Set up Docker Compose services	For Grafana and Prometheus
⌚ Configure Prometheus	Targets to monitor client and server containers
📊 Configure Grafana	Dashboards + Prometheus data source
⚡ Test Dashboards	View metrics like CPU, memory, container uptime

### ✳️ 8. Optional Enhancements (Post-Monitoring)

Feature	Optional?	Description
🔒 Harden EC2 Security	✓	Security groups, disable root login, limit ports
🌐 Domain Setup with Nginx & SSL	✓	Route 53, Nginx reverse proxy, Let's Encrypt
🔄 Rolling Updates	✓	Zero-downtime deployments
📦 Backups	✓	Auto backups for volumes (DBs, logs, etc.)
📝 Logging Stack	✓	Use Loki or ELK stack
💬 Slack Notifications on Deployments	✓	GitHub Action step to send deploy status



---

## ⌚ Next Step: Monitoring Stack Plan

If you're ready, here's what the **monitoring stack** phase will cover step by step:

1. Prepare `docker-compose-monitoring.yml`
2. **Prometheus Setup**
  - `prometheus.yml` config
  - Define `targets` for server & client containers
3. **Grafana Setup**
  - Add Prometheus as data source
  - Import a sample dashboard
4. Expose Grafana on Port 3000
5. Verify: CPU, Memory, Container health

Would you like me to generate the full monitoring `docker-compose-monitoring.yml` file first, or set up **Prometheus config** (`prometheus.yml`)?

## ☑ What We'll Do Now: Add Monitoring Stack (Grafana + Prometheus)

### ⌚ Phase Overview

Step	Description	Tool
1.	Understand what Grafana and Prometheus do	Theory
2.	Create a new <code>monitoring</code> folder in your repo	Setup
3.	Write a <code>docker-compose-monitoring.yml</code> for Prometheus + Grafana	Docker
4.	Create Prometheus config file	Config
5.	Expose your app (client or server) metrics	Python/Flask app update
6.	Test locally (optional)	Docker
7.	Deploy monitoring to EC2 via GitHub Actions	CI/CD

---

### Step 3: Create `docker-compose-monitoring.yml`

This file will:

- Start **Prometheus**
- Start **Grafana**
- Map ports so you can access them in browser:
  - Prometheus on `:9090`
  - Grafana on `:3000`

---

### Step 4: Create Prometheus Config File

Prometheus needs a config file to know where to pull metrics from.

#### Create `monitoring/prometheus.yml`

Paste the following in `monitoring/prometheus.yml`:

```
yaml
global:
  scrape_interval: 5s

scrape_configs:
  - job_name: 'server'
    static_configs:
      - targets: ['server:8000']
```

 Copy  Edit

 This tells Prometheus:

- Every 5 seconds, scrape `/metrics` from the `server` container on port `8000`.

 Make sure your `server` app exposes metrics at `/metrics` (we'll cover this next).

---

### Next Step:

Add `/metrics` endpoint to your Flask server to expose metrics using `prometheus_client`.

Devops13 > week8 > DevOpsCapstone > CICD-Actions-Demo

---

Name	Date modified	Type	Size
.git	03/08/2025 5:19 AM	File folder	
.github	03/08/2025 4:13 AM	File folder	
client	03/08/2025 4:29 AM	File folder	
monitoring	03/08/2025 5:34 AM	File folder	
server	03/08/2025 4:29 AM	File folder	
docker-compose-client.yml	03/08/2025 4:29 AM	Yaml Source File	0 KB
docker-compose-server.yml	03/08/2025 4:29 AM	Yaml Source File	0 KB
README.md	03/08/2025 4:04 AM	Markdown Source...	1 KB

Devops13 > week8 > DevOpsCapstone > CICD-Actions-Demo > monitoring

---

Name	Date modified	Type	Size
docker-compose-monitoring.yml	03/08/2025 5:40 AM	Yaml Source File	1 KB
prometheus.yml	03/08/2025 5:41 AM	Yaml Source File	1 KB

Perfect! Now let's expose a `/metrics` endpoint on your Flask server so Prometheus can scrape it.

#### Step 5: Add Prometheus Metrics to Flask App (`server.py`)

We'll use the `:prometheus_client` library to:

- Expose metrics at `/metrics`
- Track some example counters (like HTTP requests)

#### Step-by-Step:

##### 1. Install the required library on your server EC2:

SSH into your server EC2 and run:

```
bash
pip install prometheus_client
```

## 2. 🛡️ Modify `server/server.py`

Update your Flask app like this:

```
python Copy Edit

from flask import Flask
from prometheus_client import Counter, generate_latest
from prometheus_client import CONTENT_TYPE_LATEST
from flask import Response

app = Flask(__name__)

# Define a Prometheus counter metric
REQUEST_COUNT = Counter('http_requests_total', 'Total HTTP Requests')

@app.route('/')
def home():
    REQUEST_COUNT.inc()
    return "Hello from Server!"

@app.route('/metrics')
def metrics():
    return Response(generate_latest(), mimetype=CONTENT_TYPE_LATEST)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8000)
```

This will:

- Count every hit to `/`
  - Expose Prometheus metrics at `/metrics`
-

### 3. Update your server deployment

If you're using Docker:

Update your `Dockerfile` (if not already exposing port 8000):

```
dockerfile
```

 Copy  Edit

```
EXPOSE 8000
```

#### Restart the server stack

From your server EC2 terminal:

```
bash
```

 Copy  Edit

```
cd ~/server  
docker-compose down  
docker-compose up --build -d
```

---

### Step 6: Run the Monitoring Stack

On **client EC2**, go to your `monitoring/` folder and run:

```
bash
```

 Copy  Edit

```
docker-compose -f docker-compose-monitoring.yml up -d
```

## Step 1: Fix It Now with `scp`

From your **local machine**, run the following:

bash

 Copy 

```
scp -i "C:\AWSKeys\ziakhan.pem" server/server.py ec2-user@<YOUR_SERVER_IP>:~/server/server.py  
scp -i "C:\AWSKeys\ziakhan.pem" server/requirements.txt ec2-user@<YOUR_SERVER_IP>:~/server/requirements.txt  
scp -i "C:\AWSKeys\ziakhan.pem" server/Dockerfile ec2-user@<YOUR_SERVER_IP>:~/server/Dockerfile
```

Replace `<YOUR_SERVER_IP>` with your EC2 server IP (e.g. `18.168.221.107`).

If you've made changes to `docker-compose-server.yml`, copy that too:

bash

 Copy 

```
scp -i "C:\AWSKeys\ziakhan.pem" server/docker-compose-server.yml ec2-user@<YOUR_SERVER_IP>:~/server/docker-compose-server.yml
```

## Step 2: SSH and Rebuild Docker Container

SSH into the server:

bash

 Copy 

```
ssh -i "C:\AWSKeys\ziakhan.pem" ec2-user@<YOUR_SERVER_IP>  
cd ~/server
```

Then run:

bash

 Copy 

```
docker-compose -f docker-compose-server.yml down || true  
docker-compose -f docker-compose-server.yml up --build -d
```

## Step-by-Step: Add Monitoring Stack (Prometheus + Grafana)

### Folder on Local Machine

You already have:

markdown

 Copy  Edit

```
CICD-Actions-Demo/  
  └─ monitoring/  
      ├─ docker-compose-monitoring.yml  
      └─ prometheus.yml
```

### Step 1: Copy monitoring folder to your EC2 (via SCP)

Run this from your local system terminal (adjust path as needed):

bash

 Copy  Edit

```
scp -i "C:\AWSKeys\ziakhan.pem" -r "C:\Path\To\CICD-Actions-Demo\monitoring" ec2-user@<EC2_PUBLIC_IP>:/
```

Replace <EC2\_PUBLIC\_IP> with your server's IP (like 18.168.221.107 )

```
scp -i "C:\AWSKeys\ziakhan.pem" -r "C:\Path\To\CICD-Actions-Demo\monitoring" ec2-user@<EC2_PUBLIC_IP>:~/
```

```
scp -i "C:\AWSKeys\ziakhan.pem" server/server.py ec2-user@18.168.221.107:~/server/server.py
```

```
scp -i "C:\AWSKeys\ziakhan.pem" server/requirements.txt ec2-user@18.168.221.107:~/server/requirements.txt
```

```
scp -i "C:\AWSKeys\ziakhan.pem" server/Dockerfile ec2-user@18.168.221.107:~/server/Dockerfile
```

```
scp -i "C:\AWSKeys\ziakhan.pem" -r "C:\Path\To\CICD-Actions-Demo\monitoring" ec2-user@18.168.221.107:~/
```

```
C:\Windows\System32\cmd.exe
168.221.107:~/server/requirements.txt
requirements.txt

D:\Devops13\week8\DevOpsCapstone\CICD-Actions-Demo> scp -i "C:\AWSKeys\ziakhan.pem" server/Dockerfile ec2-user@18.168.221.107:~/server/Dockerfile
Dockerfile                                         100%   28     2.3KB/s  00:00

D:\Devops13\week8\DevOpsCapstone\CICD-Actions-Demo> scp -i "C:\AWSKeys\ziakhan.pem" -r "C:\Path\To\CICD-Actions-Demo\monitoring" ec2-user@18.168.221.107:~/server/
scp: stat local "C:/Path/To/CICD-Actions-Demo/monitoring": No such file or directory

D:\Devops13\week8\DevOpsCapstone\CICD-Actions-Demo> ls
README.md  client docker-compose-client.yml  docker-compose-server.yml  monitoring  server

D:\Devops13\week8\DevOpsCapstone\CICD-Actions-Demo> cd monitoring

D:\Devops13\week8\DevOpsCapstone\CICD-Actions-Demo\monitoring> scp -i "C:\AWSKeys\ziakhan.pem" -r "C:\Path\To\CICD-Actions-Demo\monitoring" ec2-user@18.168.221.107:~/server/
scp: stat local "C:/Path/To/CICD-Actions-Demo/monitoring": No such file or directory

D:\Devops13\week8\DevOpsCapstone\CICD-Actions-Demo\monitoring> cd..

D:\Devops13\week8\DevOpsCapstone\CICD-Actions-Demo> scp -i "C:\AWSKeys\ziakhan.pem" -r "monitoring" ec2-user@18.168.221.107:~/server/
docker-compose-monitoring.yml                                         100%  437     38.BK/s  00:00
prometheus.yml                                         100%  130    11.5KB/s  00:00

D:\Devops13\week8\DevOpsCapstone\CICD-Actions-Demo>
```

## Step 2: SSH into EC2 and verify

```
bash
ssh -i "C:\AWSKeys\ziakhan.pem" ec2-user@<EC2_PUBLIC_IP>
cd monitoring
ls -l
```

You should see:

```
docker-compose-monitoring.yml
prometheus.yml
```

```
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8000)
[ec2-user@ip-172-31-42-115 server]$ cd
[ec2-user@ip-172-31-42-115 ~]$ ls
monitoring  server
[ec2-user@ip-172-31-42-115 ~]$ cd monitoring/
[ec2-user@ip-172-31-42-115 monitoring]$ ls
docker-compose-monitoring.yml  prometheus.yml
[ec2-user@ip-172-31-42-115 monitoring]$ ls -l
total 8
-rw-rw-r-- 1 ec2-user ec2-user 437 Aug  3 22:23 docker-compose-monitoring.yml
-rw-rw-r-- 1 ec2-user ec2-user 130 Aug  3 22:23 prometheus.yml
[ec2-user@ip-172-31-42-115 monitoring]$
```

```
[ec2-user@ip-172-31-42-115 monitoring]$ cat docker-compose-monitoring.yml
version: '3.8'

services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    volumes:
      - grafana-storage:/var/lib/grafana
    depends_on:
      - prometheus

volumes:
  grafana-storage:
[ec2-user@ip-172-31-42-115 monitoring]$ docker-compose -f docker-compose-monitoring.yml up -d
Creating network "monitoring_default" with the default driver
Creating volume "monitoring_grafana-storage" with default driver
Pulling prometheus (prom/prometheus:)...latest: Pulling from prom/prometheus
5fa9226be034: Pull complete
1617e25568b2: Pull complete
997a69c6efe6: Pull complete
ee6cb77bebdb: Extracting [=====] 6.685MB/58.42MB
4e4782810003: Download complete
6619c1908eb: Download complete
dfc70ad9941: Download complete
fd1d3a5a5f79: Download complete
6e4c02bc6754: Download complete
08063e2dcbb: Download complete
```

```
volumes:
  grafana-storage:
[ec2-user@ip-172-31-42-115 monitoring]$ docker-compose -f docker-compose-monitoring.yml up -d
Creating network "monitoring_default" with the default driver
Creating volume "monitoring_grafana-storage" with default driver
Pulling prometheus (prom/prometheus:)... latest: Pulling from prom/prometheus
9fa9226be034: Pull complete
1617e25568b2: Pull complete
097a69c6efe6: Pull complete
2ee6cb77bebd: Pull complete
a4e782810d03: Pull complete
76619c1908eb: Pull complete
2dfc70ad9941: Pull complete
fd1d3a5a5f79: Pull complete
5e4c02bc6754: Pull complete
208063e2dcbb: Pull complete
Digest: sha256:63805ebb8d2b3920190daf1cb14a60871b16fd38bed42b857a3182bc621f4996
Status: Downloaded newer image for prom/prometheus:latest
Pulling grafana (grafana/grafana:)... latest: Pulling from grafana/grafana
9824c27679d3: Pull complete
69236fc2bfd9: Pull complete
9393ac56ed95: Pull complete
a26e3c55e1fd: Pull complete
23f45cd68698: Pull complete
ed3060fc3d6c: Pull complete
ffd7fc70b06d: Pull complete
f90a6212104f: Pull complete
30b66297d085: Pull complete
960d2bc84ffe: Pull complete
Digest: sha256:6a4edc3183d1131cb884bdf6132ddbc1d78657a13b68e4f07e41456c0130e874
Status: Downloaded newer image for grafana/grafana:latest
Creating prometheus ... done
Creating grafana ... done
[ec2-user@ip-172-31-42-115 monitoring]$
```

#### 💡 Step 4: Start the monitoring stack

```
bash ⌂ Copy ⌂ Edit
cd ~/monitoring
docker-compose -f docker-compose-monitoring.yml up -d
```

If you get an error: `docker-compose: command not found`, run:

```
bash ⌂ Copy ⌂ Edit
sudo yum install docker -y
sudo service docker start
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

### Step 5: Access Prometheus and Grafana

- \* Prometheus: [http://<EC2\\_PUBLIC\\_IP>:9090](http://<EC2_PUBLIC_IP>:9090)
- \* Grafana: [http://<EC2\\_PUBLIC\\_IP>:3000](http://<EC2_PUBLIC_IP>:3000)
  - \* Default Login:
    - \* Username: admin
    - \* Password: admin

### Next Steps (after success)

1. Modify `prometheus.yml` to scrape your server app (`:8000/metrics`)
2. Install `prometheus_client` in your app's Dockerfile
3. Add metrics endpoint in `server.py`

```
Creating grafana ... done
[ec2-user@ip-172-31-42-115 monitoring]$ cd
[ec2-user@ip-172-31-42-115 ~]$ cd server
[ec2-user@ip-172-31-42-115 server]$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
 NAMES
77db51d802b7 grafana/grafana "/run.sh" 4 minutes ago Up 4 minutes 0.0.0.0:3000->3000/tcp, :::3000
->3000/tcp grafana
06c7e3d66aed prom/prometheus "/bin/prometheus --c..." 4 minutes ago Up 4 minutes 0.0.0.0:9090->9090/tcp, :::9090
->9090/tcp prometheus
[ec2-user@ip-172-31-42-115 server]$
```

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
 NAMES
77db51d802b7 grafana/grafana "/run.sh" 4 minutes ago Up 4 minutes 0.0.0.0:3000->3000/tcp, :::3000
->3000/tcp grafana
06c7e3d66aed prom/prometheus "/bin/prometheus --c..." 4 minutes ago Up 4 minutes 0.0.0.0:9090->9090/tcp, :::9090
->9090/tcp prometheus
[ec2-user@ip-172-31-42-115 server]$ curl localhost:8000/metrics
curl: (7) Failed to connect to localhost port 8000 after 0 ms: Couldn't connect to server
[ec2-user@ip-172-31-42-115 server]$
```

`curl localhost:8000/metrics`

## ✓ Step-by-Step Fix

### 1. Security Group Rules – Allow Inbound Traffic

You're using AWS EC2, so you must allow ports 3000, 9090, and 8000 in the security group.

Steps:

1. Go to AWS Console → EC2 → Instances.
2. Select your EC2 instance ( 18.168.221.107 ).
3. Click Security → Security Groups → select the group.
4. Click Inbound rules → Edit inbound rules.
5. Add the following rules:

Type	Protocol	Port Range	Source
Custom TCP	TCP	3000	0.0.0.0/0
Custom TCP	TCP	9090	0.0.0.0/0
Custom TCP	TCP	8000	0.0.0.0/0

Save the rules.

The screenshot shows the AWS Security Groups console for a security group named "sg-0387d1ba5298a81b6 - launch-wizard-1". The "Inbound rules" tab is selected. A success message at the top states: "Inbound security group rules successfully modified on security group sg-0387d1ba5298a81b6 - launch-wizard-1". The "Details" section shows the security group name, ID, owner, and various counts for rules and permissions. The "Inbound rules" table lists eight rules, each allowing traffic from 0.0.0.0/0 on specific ports (3000, 9090, 8000) using TCP. The table includes columns for Name, Security group rule ID, IP version, Type, Protocol, Port range, Source, and Description.

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
-	sg-01f8cd1f304ed337	IPv4	Custom TCP	TCP	3000	0.0.0.0/0	-
-	sg-0a486029a7f5e9e	IPv4	Custom TCP	TCP	9090	0.0.0.0/0	-
-	sg-017423aee11ef900	IPv4	Custom TCP	TCP	8000	0.0.0.0/0	-
-	sg-099449c17f13a529	IPv4	Custom TCP	TCP	3000	0.0.0.0/0	-
-	sg-012300a0a156ab28	IPv4	HTTPS	TCP	443	0.0.0.0/0	-
-	sg-039242ec07726a0e	IPv4	HTTP	TCP	80	0.0.0.0/0	-
-	sg-09593d4496a6c594	IPv4	SSH	TCP	22	0.0.0.0/0	-

### 3. Restart Docker Compose

After adjusting the files, **restart** containers:

```
bash

cd ~/monitoring
docker-compose -f docker-compose-monitoring.yml down
docker-compose -f docker-compose-monitoring.yml up -d

cd ~/server
docker-compose -f docker-compose-server.yml down
docker-compose -f docker-compose-server.yml up -d
```

```
[ec2-user@ip-172-31-42-115:~/server]
[ec2-user@ip-172-31-42-115 ~]$ ls
monitoring server
[ec2-user@ip-172-31-42-115 ~]$ cd ~/monitoring
[ec2-user@ip-172-31-42-115 monitoring]$ docker-compose -f docker-compose-monitoring.yml down
g.yml up -d

cd ~/server
docker-compose -f docker-compose-server.yml down
docker-compose -f docker-compose-server.yml up -d
Stopping grafana ... done
Stopping prometheus ... done
Removing grafana ... done
Removing prometheus ... done
Removing network monitoring_default
[ec2-user@ip-172-31-42-115 monitoring]$ docker-compose -f docker-compose-monitoring.yml up -d
Creating network "monitoring_default" with the default driver
Creating prometheus ... done
Creating grafana ... done
[ec2-user@ip-172-31-42-115 monitoring]$
[ec2-user@ip-172-31-42-115 monitoring]$ cd ~/server
[ec2-user@ip-172-31-42-115 server]$ docker-compose -f docker-compose-server.yml down
Removing server_devops_server_1 ... done
Removing network server_default
[ec2-user@ip-172-31-42-115 server]$ docker-compose -f docker-compose-server.yml up -d
Creating network "server_default" with the default driver
Creating server_devops_server_1 ... done
[ec2-user@ip-172-31-42-115 server]
```

### 5. Test Locally and Remotely

Run:

```
bash

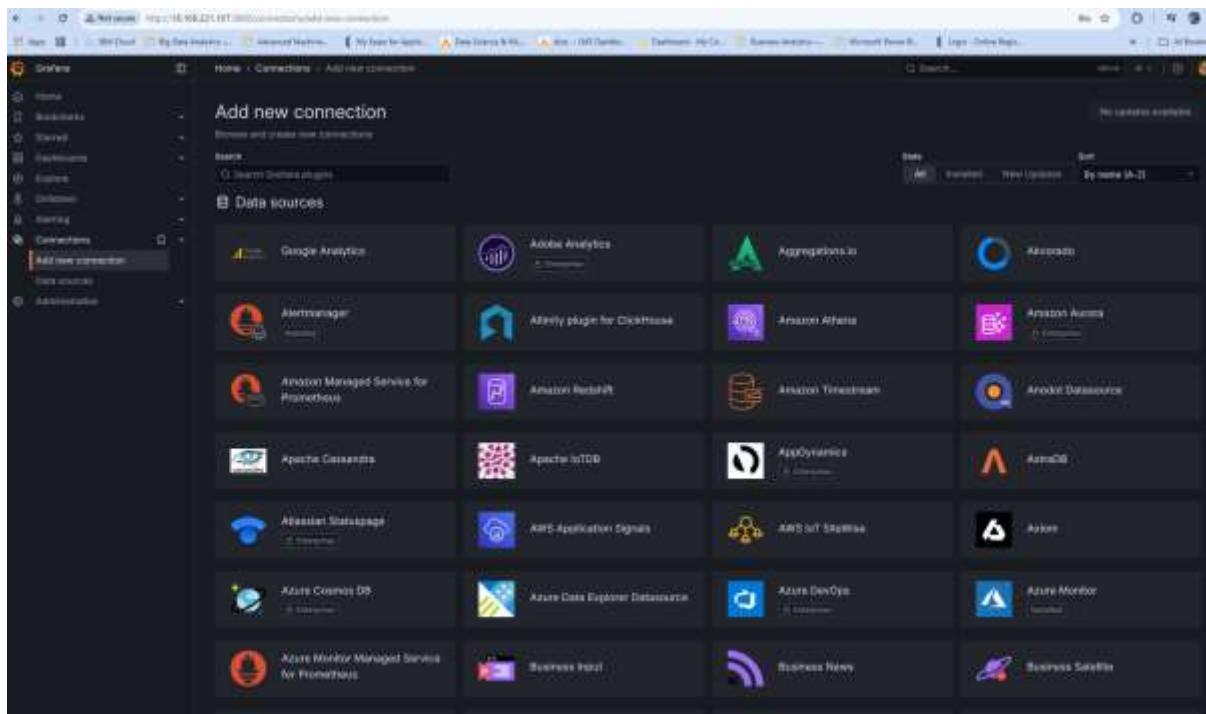
curl localhost:8000/metrics
```

Then check in browser:

- <http://18.168.221.107:3000> (Grafana)
- <http://18.168.221.107:9090> (Prometheus)
- <http://18.168.221.107:8000/metrics> (Python app)

The screenshot shows the Prometheus web interface at <http://10.105.223.102:9090>. The title bar says "Not Home". The main area displays a query editor with the placeholder "Enter expression (Press Enter/Enter for search)" and tabs for "Table", "Graph", and "Report". A message "No data queried yet" is shown. At the bottom left is a "Add query" button.

The screenshot shows the Grafana login screen at <http://10.104.32.1.107.3305/login>. The title bar says "Not logged in". The main area features a dark purple gradient background with a central login form. The form includes a yellow sunburst logo, the text "Welcome to Grafana", a "Email or username" input field, a "Password" input field, a "Log in" button, and a "Forgot your password?" link.



## Fix All Issues — Step by Step

### Step 1: Update `Dockerfile` in `server/`

Make sure your `Dockerfile` in the `server/` folder looks like this:

```
dockerfile
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY server.py .
EXPOSE 8000
CMD ["python", "server.py"]
```

### Step 3: Update `docker-compose-server.yml`

Ensure you are mapping port `8000` to host and giving container a name:

```
yaml  
  
version: '3'  
services:  
  server:  
    build: ./server  
    ports:  
      - "8000:8000"  
    container_name: server_devops_server
```

### Fix: Docker `COPY` Error in Dockerfile

From the error:

```
sql  
  
COPY requires at least two arguments, but only one was provided
```

#### Problem:

You're missing the destination directory in the `COPY` command.

#### Solution – Update `Dockerfile`:

Update line 12 like this:

```
dockerfile  
  
COPY server.py /app/
```

If you're copying multiple files (e.g., `requirements.txt`, `server.py`), do:

```
dockerfile  
  
WORKDIR /app  
COPY requirements.txt server.py ./
```

Then rebuild:

```
bash  
  
docker-compose -f docker-compose-server.yml up --build -d
```

 Solution: Update `docker-compose-monitoring.yml`

Ensure it contains both `prometheus` and `server` in the same network. Here's an example:

```
yaml
version: "3"

services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"
    networks:
      - monitoring

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    networks:
      - monitoring

  server:
    build:
      context: ../server
    container_name: devops_server
    ports:
      - "8000:8000"
    expose:
      - "8000"
    networks:
      - monitoring

networks:
  monitoring:
```

 Copy  Edit

```
[ec2-user@ip-172-31-42-115 ~]$ cd server
Creating server_devops_server_1 ... done
[ec2-user@ip-172-31-42-115 server]$ ls
ls
docker-compose-server.yml Dockerfile requirements.txt server.py
[ec2-user@ip-172-31-42-115 server]$ cd monitoring
[ec2-user@ip-172-31-42-115 monitoring]$ ls
ls
monitoring_prometheus.yml
[ec2-user@ip-172-31-42-115 monitoring]$ cat docker-compose-monitoring.yml
version: "3"
services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"
    networks:
      - monitoring

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    networks:
      - monitoring

  server:
    build:
      context: ../server
    container_name: devops_server
    ports:
      - "8000:8000"
    expose:
      - "8000"
    networks:
      - monitoring

networks:
  monitoring:
[ec2-user@ip-172-31-42-115 monitoring]$
```

## 📝 Test After Fixes

1. Rebuild with updated paths:

```
bash docker-compose -f docker-compose-monitoring.yml up --build -d
```

2. Check status:

```
bash docker ps
```

3. Test Flask metrics endpoint from host:

```
bash curl http://localhost:8000/metrics
```

4. Test from inside Prometheus container:

```
bash docker exec -it prometheus curl http://server:8000/metrics
```

5. Open in browser:

- Prometheus: <http://<EC2-IP>:9090>
  - Grafana: <http://<EC2-IP>:3000>
  - Add Prometheus as data source in Grafana.
- 

## ✅ Project Coverage So Far

Feature	Status
GitHub Actions CI/CD	✅ Done
SCP fallback (manual)	✅ Done
Dockerized Client & Server	✅ Done
Prometheus + Grafana Setup	🟡 In progress
Flask Metrics Endpoint	✅ Done
Prometheus Scraping	✍️ Fixing
Grafana Dashboard View	🔜 Next Step

---

```
ec2-user@ip-172-31-42-115:~/monitoring
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 32768.0
# HELP http_requests_total Total HTTP Requests
# TYPE http_requests_total counter
http_requests_total 0.0
# HELP http_requests_created Total HTTP Requests
# TYPE http_requests_created gauge
http_requests_created 1.754262257881273e+09
[ec2-user@ip-172-31-42-115 monitoring]$ sudo lsof -i :8000
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
docker-pr 13167 root 4u IPv4 120930 0t0 TCP *:irdmi (LISTEN)
docker-pr 13172 root 4u IPv6 120936 0t0 TCP *:irdmi (LISTEN)
[ec2-user@ip-172-31-42-115 monitoring]$ cd
[ec2-user@ip-172-31-42-115 ~]$ sudo lsof -i :8000
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
docker-pr 13167 root 4u IPv4 120930 0t0 TCP *:irdmi (LISTEN)
docker-pr 13172 root 4u IPv6 120936 0t0 TCP *:irdmi (LISTEN)
[ec2-user@ip-172-31-42-115 ~]$ cd server
[ec2-user@ip-172-31-42-115 server]$ sudo lsof -i :8000
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
docker-pr 13167 root 4u IPv4 120930 0t0 TCP *:irdmi (LISTEN)
docker-pr 13172 root 4u IPv6 120936 0t0 TCP *:irdmi (LISTEN)
[ec2-user@ip-172-31-42-115 server]$ sudo kill -9 <PID>
-bash: syntax error near unexpected token `newline'
[ec2-user@ip-172-31-42-115 server]$ sudo kill -9 13167
[ec2-user@ip-172-31-42-115 server]$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
97f59b18151e grafana/grafana "/run.sh" 7 minutes ago Up 6 minutes 0.0.0.0:3000->3000/tcp,
:::3000->3000/tcp grafana
bd026fd0c885 prom/prometheus "/bin/prometheus --c..." 7 minutes ago Up 6 minutes 0.0.0.0:9090->9090/tcp,
:::9090->9090/tcp prometheus
c39f7d048104 server_devops_server "python server.py" 11 minutes ago Up 11 minutes 0.0.0.0:8000->8000/tcp,
:::8000->8000/tcp server_devops_server_1
[ec2-user@ip-172-31-42-115 server]$ docker stop c39f7d048104
c39f7d048104
[ec2-user@ip-172-31-42-115 server]$ cd
[ec2-user@ip-172-31-42-115 ~]$ ls
monitoring server
[ec2-user@ip-172-31-42-115 ~]$ cd monitoring/
[ec2-user@ip-172-31-42-115 monitoring]$ docker-compose -f docker-compose-monitoring.yml up --build -d
Building server
[+] Building 0.7s (10/10) FINISHED
   > [internal] load build definition from Dockerfile docker:default
   >> transferring dockerfile: 377B 0.0s
   > [internal] load metadata for docker.io/library/python:3.9-slim 0.0s
   > [internal] load .dockerrcignore 0.0s
   >> transferring context: 2B 0.0s
   >> [1/5] FROM docker.io/library/python:3.9-slim@sha256:4bb826e2ca2191c24a03bdbf8c342a73c6ee4ce929eac5ebc47feff78f6 0.0s
   >> [internal] load build context 0.0s
   >> transferring context: 668 0.0s
   >> CACHED [2/5] WORKDIR /app 0.0s
   >> CACHED [3/5] COPY requirements.txt . 0.0s
   >> CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt 0.0s
   >> CACHED [5/5] COPY server.py /app/ 0.0s
   >> exporting to image 0.0s
   >> >> exporting layers 0.0s
   >> >> writing image sha256:b23d68eea884d08cb0c551cfabdbd6012d2c02434c38627f0b0861fd58c8d1d7 0.0s
   >> >> naming to docker.io/library/monitoring_server 0.0s
prometheus is up-to-date
grafana is up-to-date
Starting devops_server ... done
[ec2-user@ip-172-31-42-115 monitoring]$
```

## 2. Grafana Login Issue

You're using:

- **Username:** admin
- **Password:** likely incorrect.

### Solution:

If you forgot the password, reset it like this:

#### Option A: Change Grafana Admin Password using Docker CLI

bash

 Copy  Edit

```
docker exec -it <grafana_container_id_or_name> grafana-cli admin reset-admin-password <newpassword>
```

Example:

bash

 Copy  Edit

```
docker exec -it grafana grafana-cli admin reset-admin-password admin123
```

Then try again with:

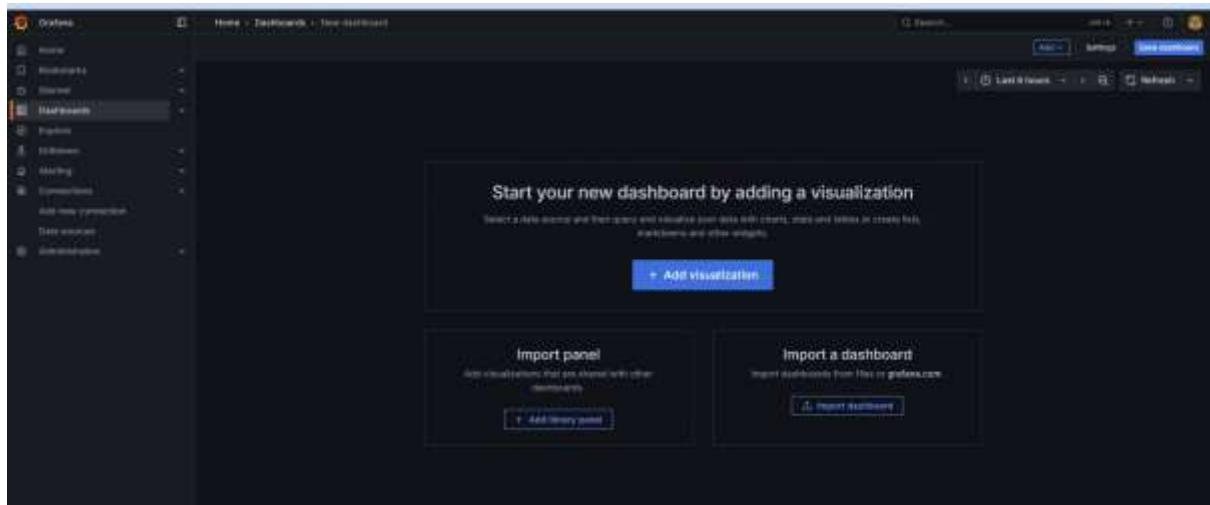
- **Username:** admin
  - **Password:** admin123
-

```
prometheus is up-to-date
grafana is up-to-date
Starting devops_server ... done
[ec2-user@ip-172-31-42-115 monitoring]$ curl localhost:8000/metrics
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 533.0
python_gc_objects_collected_total{generation="1"} 33.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 78.0
python_gc_collections_total{generation="1"} 7.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="9",patchlevel="23",version="3.9.23"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.13369088e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 3.0957568e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.75426302635e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.35
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 6.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 32768.0
# HELP http_requests_total Total HTTP Requests
# TYPE http_requests_total counter
http_requests_total 0.0
# HELP http_requests_created Total HTTP Requests
# TYPE http_requests_created gauge
http_requests_created 1.7542630272071826e+09
[ec2-user@ip-172-31-42-115 monitoring]$
```

← → ⌂ Not secure http://18.168.221.107:8000/metrics

Apps IBM Cloud Big Data Analytics... Advanced Machine... My Essex for App... Data Science & ML dice -- LMS Des... Des

```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 533.0
python_gc_objects_collected_total{generation="1"} 33.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 78.0
python_gc_collections_total{generation="1"} 7.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="9",patchlevel="23",version="3.9.23"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.13369088e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 3.1002824e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.75426302635e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.4800000000000004
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 6.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 32768.0
# HELP http_requests_total Total HTTP Requests
# TYPE http_requests_total counter
http_requests_total 0.0
# HELP http_requests_created Total HTTP Requests
# TYPE http_requests_created gauge
http_requests_created 1.7542630272071826e+09
```



---

## Final Summary of Your DevOps Capstone Project

### Project Goal

Create a monitoring stack using **Docker**, **Grafana**, **Prometheus**, and a custom **Flask server** exposing Prometheus metrics.

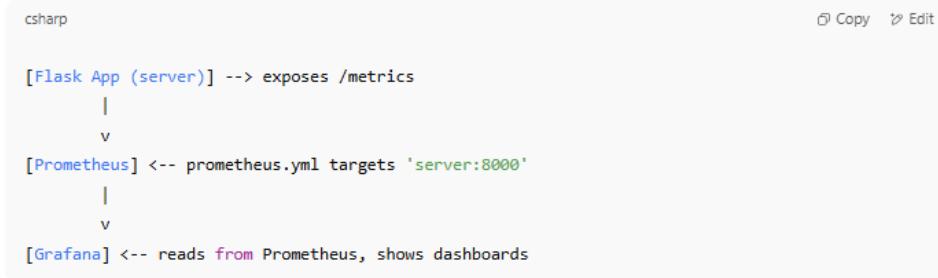
---

### Technologies Used

Layer	Tool	Purpose
Containerization	Docker, Docker Compose	Containerize services (Grafana, Prometheus, Flask server)
Monitoring	Prometheus	Scrapes metrics from Flask service
Visualization	Grafana	Displays dashboards and graphs
Backend	Python Flask	Exposes <code>/metrics</code> endpoint with Prometheus client
Infra	AWS EC2 (Amazon Linux)	Hosted all services

---

### Architecture Overview



### What You Built

- Flask Python app with Prometheus metrics
  - Dockerfile + `requirements.txt` for the app
  - Docker Compose setup for:
    - Flask server
    - Prometheus (scraping `server:8000`)
    - Grafana (access via `:3000`)
  - Custom `prometheus.yml` config for scraping
  - Successfully visualized metrics on Grafana
-

### Remaining Future Work (Optional Ideas)

Feature	Description
<input checked="" type="checkbox"/> Alerts in Prometheus	Add alerting rules and integrate email/Slack
<input checked="" type="checkbox"/> Auto Docker Image Push	CI/CD with GitHub Actions and Docker Hub
<input checked="" type="checkbox"/> Nginx Reverse Proxy	Use Nginx for SSL and domain routing
<input checked="" type="checkbox"/> Grafana Dashboards	Create custom dashboards using scraped metrics
<input type="checkbox"/> Service Discovery	Use Docker labels or DNS-based discovery in Prometheus
<input type="checkbox"/> Monitoring Client	Add Node Exporter or Blackbox exporter
<input type="checkbox"/> Persistent Volumes	Mount volumes to preserve Grafana/Prometheus data
<input type="checkbox"/> Terraform Setup	Provision EC2 instance and network using Terraform