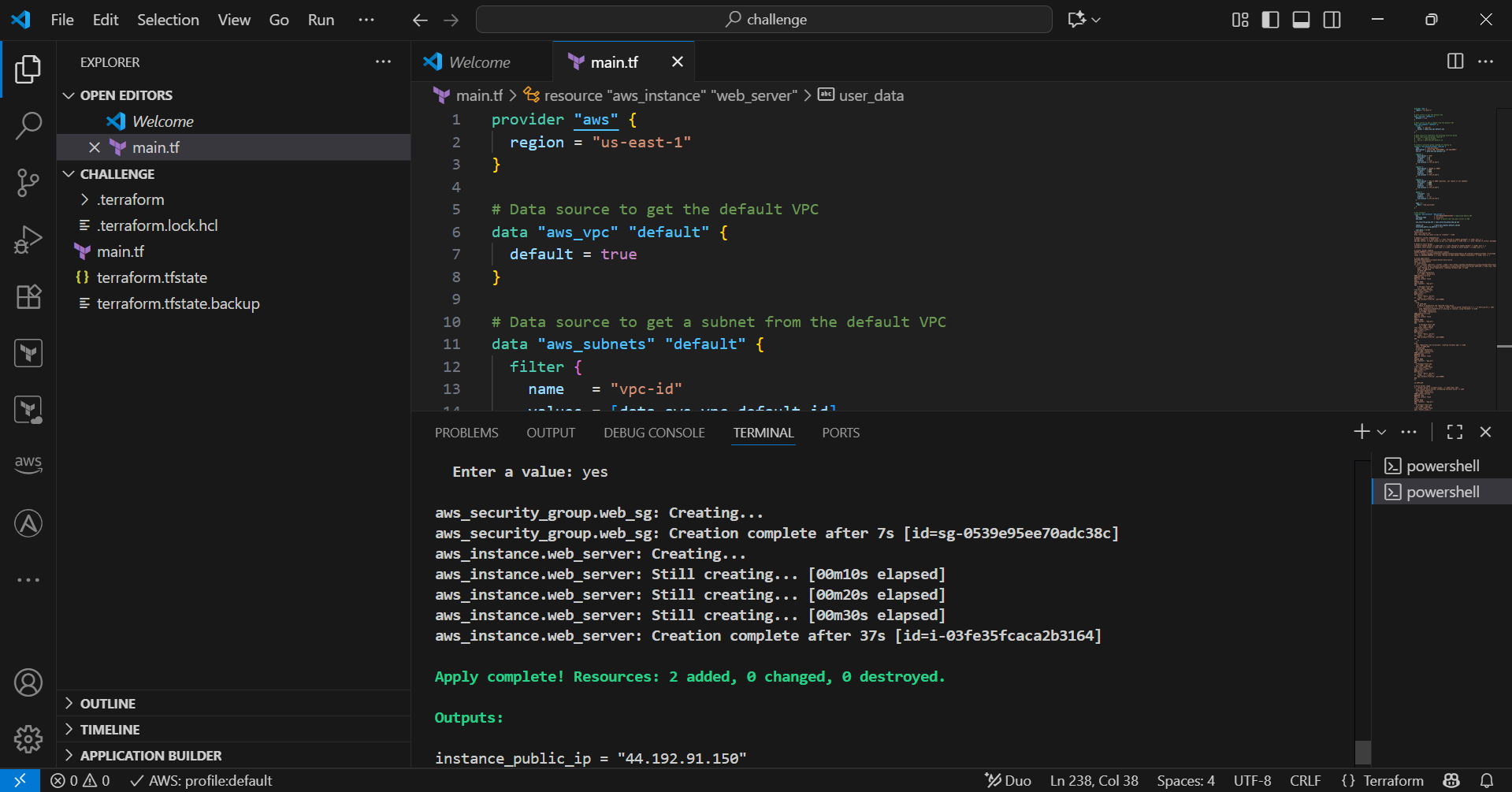
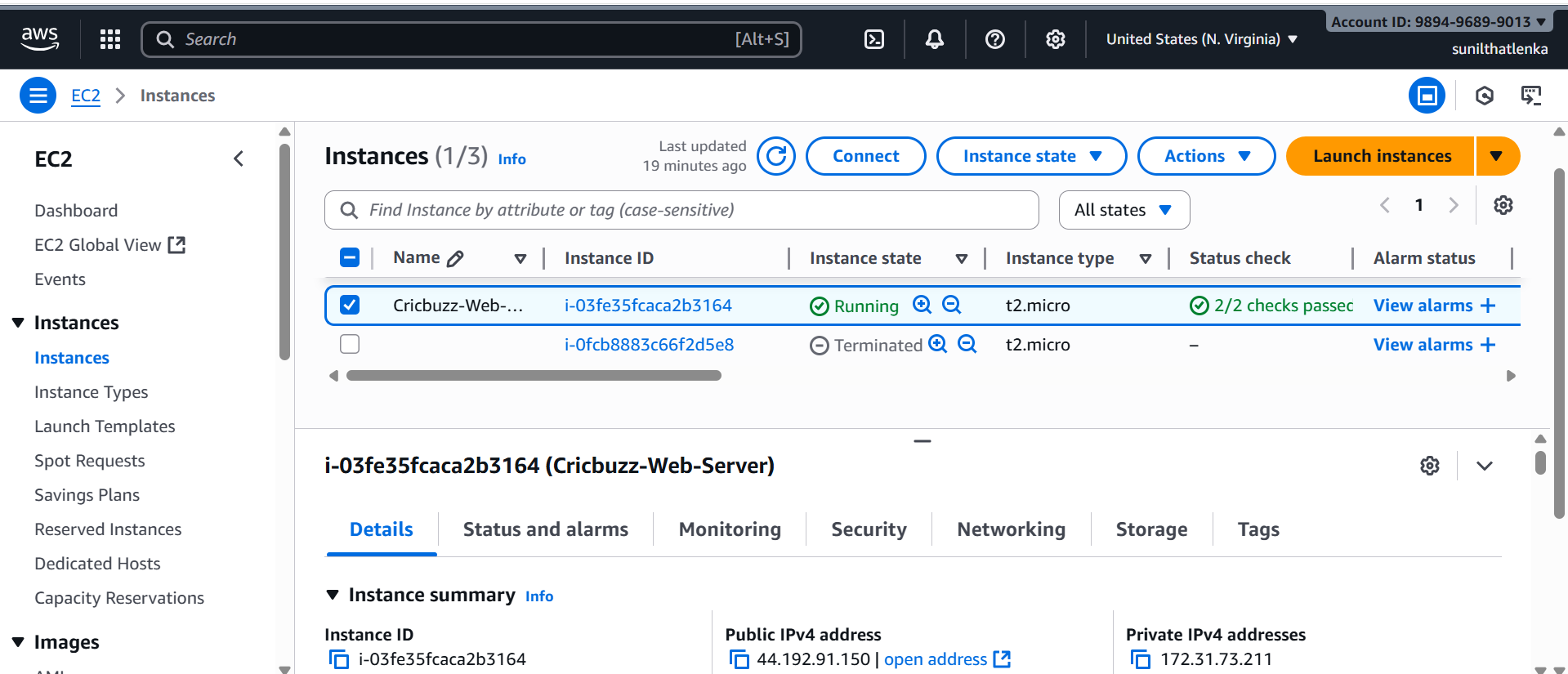
**Terraform Challenge**

**Write a code using Terraform that deploys and achieves the following:-**

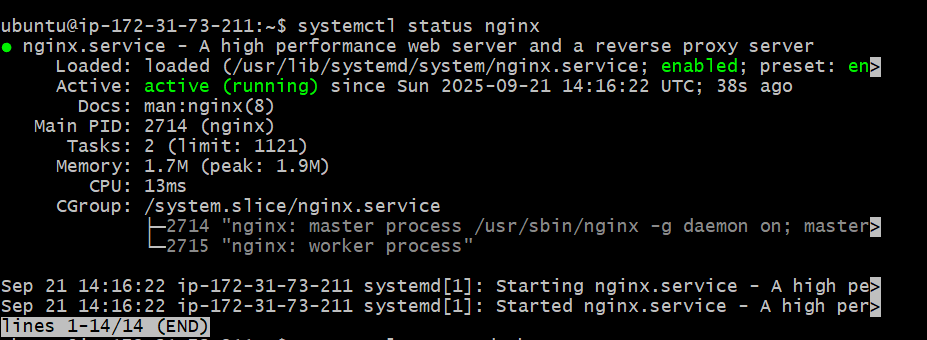
****

**1.Create an EC2 Instance on AWS Cloud using the below-mentioned ubuntu AMI and instance\_type. Or, (You can use of your choice as well) ami = "ami-0461b1436c9b3c1a3" instance\_type = "t2.micro"**

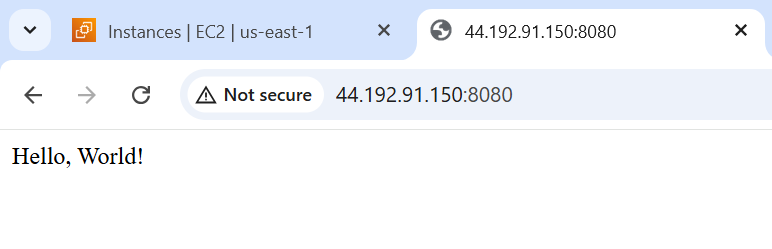
EC2 created (Ubuntu, t2.micro).

****

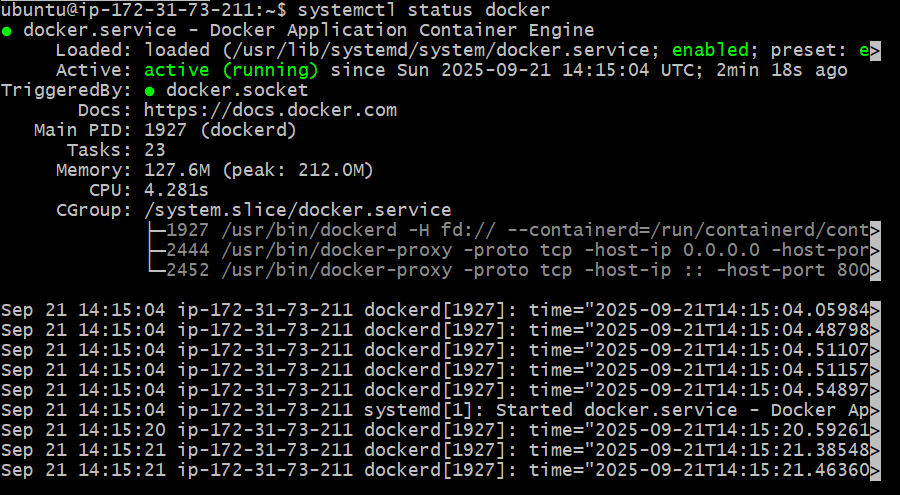
**2.Install a web server on the same EC2 Instance. The web server should listen on port 8080.**

****

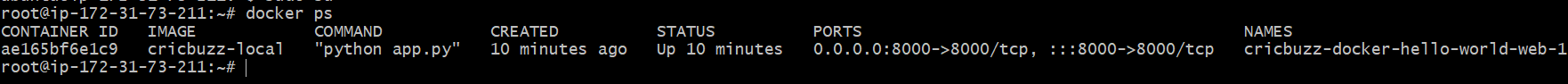
Nginx installed & listening on 8080.

****

**3.Create a Docker-compose file for the project: https://github.com/betawins/cricbuzz-docker-hello-world.git and point it to port 8000.**

****

Docker Compose runs Cricbuzz app on 8000.

****

 Clones **your GitHub project**.

 Builds a Docker image cricbuzz-local.

 Runs it with **Docker Compose** on port 8000.

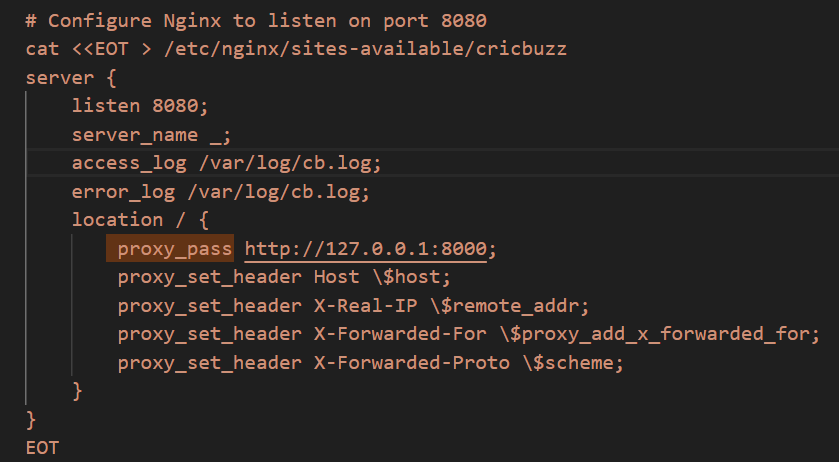
**4. Make the web server point to the above service so that 8080 returns the response from the Docker service.**

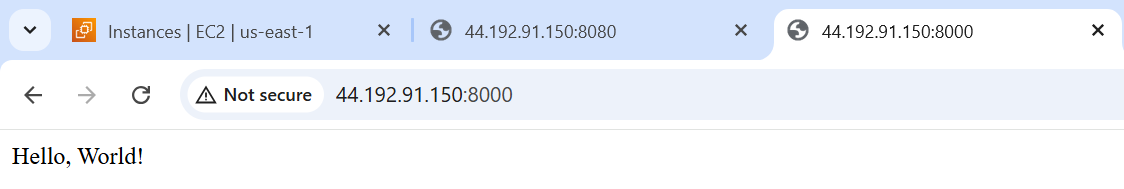
Nginx reverse proxy connects 8080 → 8000.

 Nginx listens on **8080**.

 Proxies requests to Docker container running on **8000**.

 So when you hit http://<EC2-IP>:8080, you actually get the Docker app response.



****

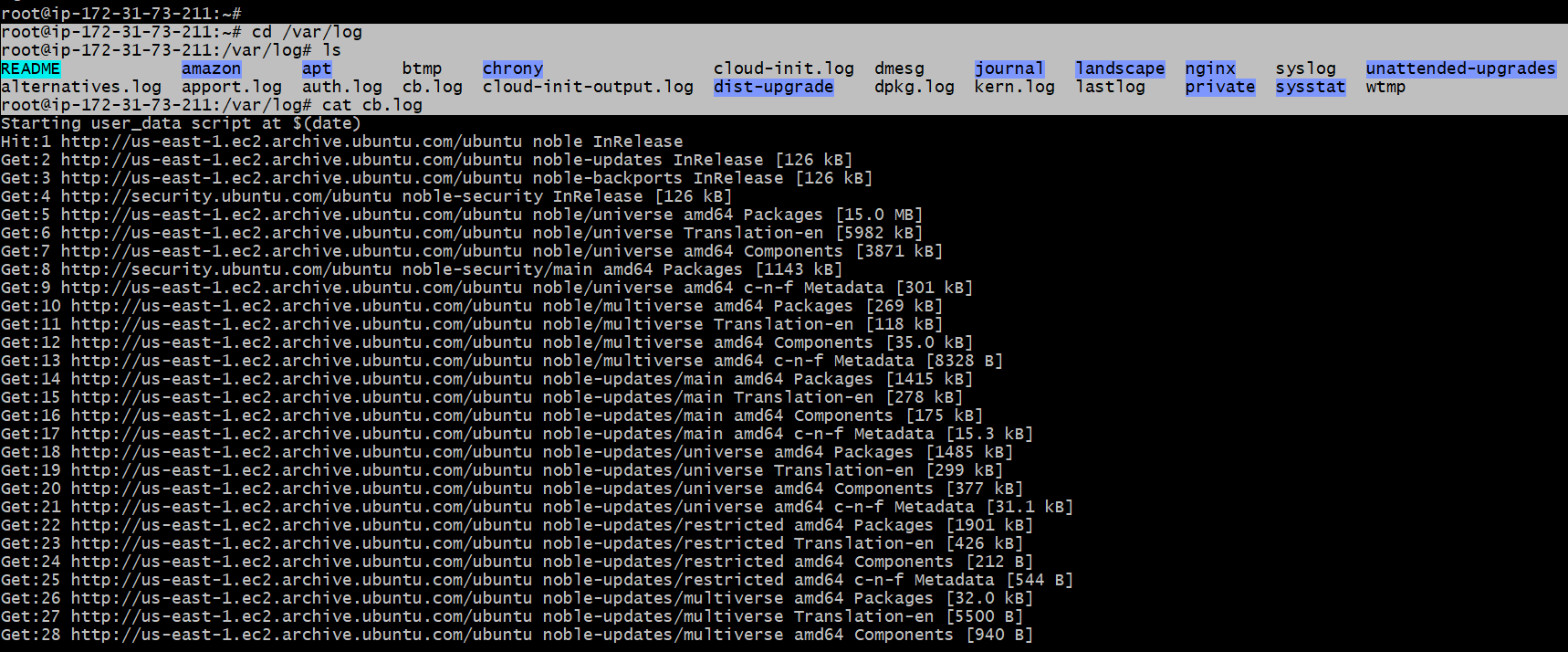
**5. Configure to Save the Logs in the location /var/log/cb.log**

 All setup steps write logs to /var/log/cb.log.

 Nginx access\_log and error\_log also go to /var/log/cb.log.

Logs saved at /var/log/cb.log.

**From root location-> cd /var/log/ -> cat cb.log**

****

**6. Write the Log rotate configuration for the above file**

Logrotate configured for cb.log.

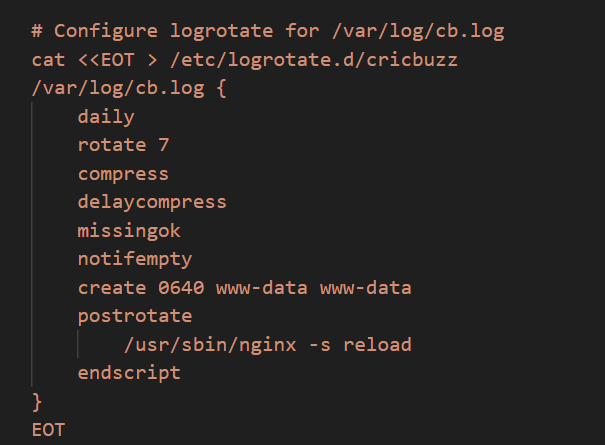
 Rotates cb.log **daily**.

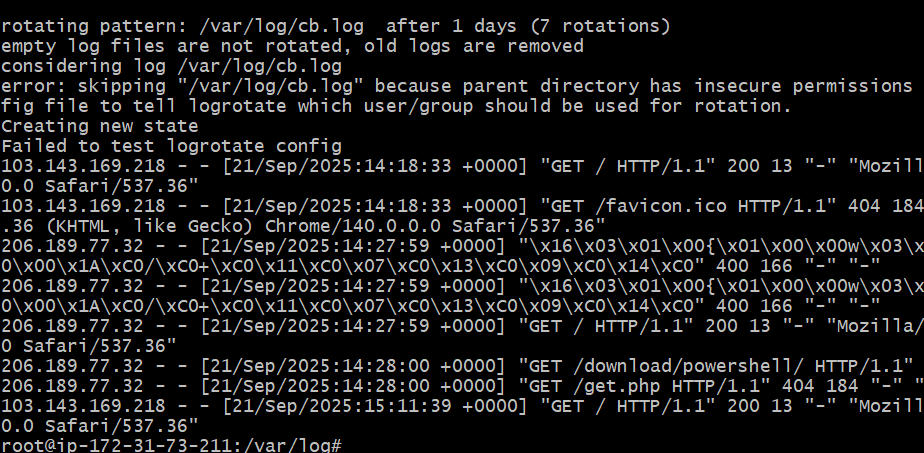
 Keeps **7 days** of old logs.

 Compresses old logs.

 Ensures file recreated with correct permissions (www-data).

 Restarts nginx after rotation so it writes to the new file.





**Code:**

**provider "aws" {**

**region = "us-east-1"**

**}**

**# Data source to get the default VPC**

**data "aws\_vpc" "default" {**

**default = true**

**}**

**# Data source to get a subnet from the default VPC**

**data "aws\_subnets" "default" {**

**filter {**

**name = "vpc-id"**

**values = [data.aws\_vpc.default.id]**

**}**

**}**

**# Data source to reference the existing security group**

**# data "aws\_security\_group" "web\_sg" {**

**# name = "web\_sg\_unique"**

**# vpc\_id = data.aws\_vpc.default.id**

**# }**

**# Create a security group instead of looking up**

**resource "aws\_security\_group" "web\_sg" {**

**name = "web\_sg\_unique"**

**description = "Allow SSH, HTTP(8080), and App(8000)"**

**vpc\_id = data.aws\_vpc.default.id**

**ingress {**

**description = "SSH"**

**from\_port = 22**

**to\_port = 22**

**protocol = "tcp"**

**cidr\_blocks = ["0.0.0.0/0"]**

**}**

**ingress {**

**description = "NGINX on 8080"**

**from\_port = 8080**

**to\_port = 8080**

**protocol = "tcp"**

**cidr\_blocks = ["0.0.0.0/0"]**

**}**

**ingress {**

**description = "App on 8000 (optional, can remove if not needed)"**

**from\_port = 8000**

**to\_port = 8000**

**protocol = "tcp"**

**cidr\_blocks = ["0.0.0.0/0"]**

**}**

**egress {**

**from\_port = 0**

**to\_port = 0**

**protocol = "-1"**

**cidr\_blocks = ["0.0.0.0/0"]**

**}**

**tags = {**

**Name = "web\_sg\_unique"**

**}**

**}**

**# EC2 Instance**

**resource "aws\_instance" "web\_server" {**

**ami = "ami-0360c520857e3138f" # Specified Ubuntu AMI**

**instance\_type = "t2.micro"**

**key\_name = "rock" # Ensure this key pair exists in AWS**

**vpc\_security\_group\_ids = [aws\_security\_group.web\_sg.id]**

**subnet\_id = data.aws\_subnets.default.ids[0]**

**associate\_public\_ip\_address = true**

**user\_data = <<-EOF**

**#!/bin/bash**

**LOG="/var/log/cb.log"**

**echo "Starting user\_data script at \$(date)" > $LOG**

**# Update & install dependencies**

**apt-get update -y >> $LOG 2>&1 || { echo "Failed to update packages" >> $LOG; exit 1; }**

**apt-get install -y nginx docker.io git curl logrotate >> $LOG 2>&1 || { echo "Failed to install packages" >> $LOG; exit 1; }**

**# Enable & start Docker**

**systemctl enable docker >> $LOG 2>&1 || { echo "Failed to enable Docker" >> $LOG; exit 1; }**

**systemctl start docker >> $LOG 2>&1 || { echo "Failed to start Docker" >> $LOG; exit 1; }**

**# Install Docker Compose**

**DOCKER\_COMPOSE=/usr/local/bin/docker-compose**

**curl -L "https://github.com/docker/compose/releases/download/v2.20.2/docker-compose-$(uname -s)-$(uname -m)" -o $DOCKER\_COMPOSE >> $LOG 2>&1 || { echo "Failed to download Docker Compose" >> $LOG; exit 1; }**

**chmod +x $DOCKER\_COMPOSE || { echo "Failed to make Docker Compose executable" >> $LOG; exit 1; }**

**# Clone application**

**APP\_DIR=/home/ubuntu/cricbuzz-docker-hello-world**

**mkdir -p /home/ubuntu**

**cd /home/ubuntu**

**if curl --output /dev/null --silent --head --fail https://github.com/betawins/cricbuzz-docker-hello-world.git; then**

**if ! git clone https://github.com/betawins/cricbuzz-docker-hello-world.git $APP\_DIR >> $LOG 2>&1; then**

**echo "Failed to clone repository, creating fallback app" >> $LOG**

**mkdir -p $APP\_DIR**

**cd $APP\_DIR**

**# Fallback Dockerfile**

**cat <<EOD > Dockerfile**

**FROM python:3.9-slim**

**WORKDIR /app**

**RUN pip install flask**

**COPY . .**

**EXPOSE 8000**

**CMD ["python", "app.py"]**

**EOD**

**# Fallback Flask app**

**cat <<EOD > app.py**

**from flask import Flask**

**app = Flask(\_\_name\_\_)**

**@app.route('/')**

**def hello():**

**return 'Hello, World!'**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(host='0.0.0.0', port=8000)**

**EOD**

**else**

**cd $APP\_DIR**

**# Check if Dockerfile and required files exist**

**if [ ! -f Dockerfile ] || ! grep -q "COPY ./hello\_world" Dockerfile || [ ! -f hello\_world ]; then**

**echo "Repository Dockerfile is missing or invalid, using fallback" >> $LOG**

**# Fallback Dockerfile**

**cat <<EOD > Dockerfile**

**FROM python:3.9-slim**

**WORKDIR /app**

**RUN pip install flask**

**COPY . .**

**EXPOSE 8000**

**CMD ["python", "app.py"]**

**EOD**

**# Fallback Flask app**

**cat <<EOD > app.py**

**from flask import Flask**

**app = Flask(\_\_name\_\_)**

**@app.route('/')**

**def hello():**

**return 'Hello, World!'**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(host='0.0.0.0', port=8000)**

**EOD**

**fi**

**fi**

**else**

**echo "Repository not accessible, creating fallback app" >> $LOG**

**mkdir -p $APP\_DIR**

**cd $APP\_DIR**

**# Fallback Dockerfile**

**cat <<EOD > Dockerfile**

**FROM python:3.9-slim**

**WORKDIR /app**

**RUN pip install flask**

**COPY . .**

**EXPOSE 8000**

**CMD ["python", "app.py"]**

**EOD**

**# Fallback Flask app**

**cat <<EOD > app.py**

**from flask import Flask**

**app = Flask(\_\_name\_\_)**

**@app.route('/')**

**def hello():**

**return 'Hello, World!'**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(host='0.0.0.0', port=8000)**

**EOD**

**fi**

**cd $APP\_DIR**

**# Build Docker image**

**if ! docker build -t cricbuzz-local . >> $LOG 2>&1; then**

**echo "Docker build failed, attempting fallback build" >> $LOG**

**# Fallback Dockerfile**

**cat <<EOD > Dockerfile**

**FROM python:3.9-slim**

**WORKDIR /app**

**RUN pip install flask**

**COPY . .**

**EXPOSE 8000**

**CMD ["python", "app.py"]**

**EOD**

**# Fallback Flask app**

**cat <<EOD > app.py**

**from flask import Flask**

**app = Flask(\_\_name\_\_)**

**@app.route('/')**

**def hello():**

**return 'Hello, World!'**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(host='0.0.0.0', port=8000)**

**EOD**

**docker build -t cricbuzz-local . >> $LOG 2>&1 || { echo "Fallback Docker build failed" >> $LOG; exit 1; }**

**fi**

**# Docker Compose file**

**cat <<EOT > $APP\_DIR/docker-compose.yml**

**version: '3'**

**services:**

**web:**

**image: cricbuzz-local**

**ports:**

**- "8000:8000"**

**restart: always**

**EOT**

**# Start container**

**$DOCKER\_COMPOSE -f $APP\_DIR/docker-compose.yml up -d**

**>> $LOG 2>&1 || { echo "Failed to start Docker container" >> $LOG; exit 1; }**

**# Wait for app to be ready**

**sleep 60**

**# Verify Flask app is running**

**if ! curl --fail http://127.0.0.1:8000 >> $LOG 2>&1; then**

**echo "Flask app failed to start" >> $LOG**

**exit 1**

**fi**

**# Remove default Nginx site to avoid conflicts**

**rm -f /etc/nginx/sites-enabled/default >> $LOG 2>&1**

**# Configure Nginx to listen on port 8080**

**cat <<EOT > /etc/nginx/sites-available/cricbuzz**

**server {**

**listen 8080;**

**server\_name \_;**

**access\_log /var/log/cb.log;**

**error\_log /var/log/cb.log;**

**location / {**

**proxy\_pass http://127.0.0.1:8000;**

**proxy\_set\_header Host \$host;**

**proxy\_set\_header X-Real-IP \$remote\_addr;**

**proxy\_set\_header X-Forwarded-For \$proxy\_add\_x\_forwarded\_for;**

**proxy\_set\_header X-Forwarded-Proto \$scheme;**

**}**

**}**

**EOT**

**# Link Nginx configuration**

**ln -sf /etc/nginx/sites-available/cricbuzz /etc/nginx/sites-enabled/cricbuzz >> $LOG 2>&1 || { echo "Failed to link Nginx config" >> $LOG; exit 1; }**

**# Configure logrotate for /var/log/cb.log**

**cat <<EOT > /etc/logrotate.d/cricbuzz**

**/var/log/cb.log {**

**daily**

**rotate 7**

**compress**

**delaycompress**

**missingok**

**notifempty**

**create 0640 www-data www-data**

**postrotate**

**/usr/sbin/nginx -s reload**

**endscript**

**}**

**EOT**

**# Test and restart Nginx**

**if ! nginx -t >> $LOG 2>&1; then**

**echo "Nginx configuration test failed" >> $LOG**

**exit 1**

**fi**

**systemctl restart nginx >> $LOG 2>&1 || { echo "Failed to restart Nginx" >> $LOG; exit 1; }**

**systemctl enable nginx >> $LOG 2>&1 || { echo "Failed to enable Nginx" >> $LOG; exit 1; }**

**# Test logrotate configuration**

**logrotate -d /etc/logrotate.d/cricbuzz >> $LOG 2>&1 || { echo "Failed to test logrotate config" >> $LOG; exit 1; }**

**echo "user\_data script completed successfully at \$(date)" >> $LOG**

**EOF**

**tags = {**

**Name = "Cricbuzz-Web-Server"**

**}**

**}**

**# Output the public IP of the EC2 instance**

**output "instance\_public\_ip" {**

**value = aws\_instance.web\_server.public\_ip**

**}**