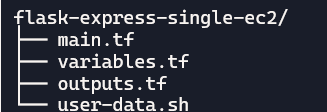
#### **Part 1: Deploy Both Flask and Express on a Single EC2 Instance**

1. **Objective**: Deploy both the Flask backend and the Express frontend on a **single EC2 instance** using Terraform.
   * Provision the EC2 instance with Terraform.
   * Configure the instance with:
     + A user data script or a configuration management tool (e.g., Ansible or Cloud-Init) to install dependencies (Python, Node.js).
     + Scripts to start the Flask backend and the Express frontend.
   * Ensure both applications are running on different ports (e.g., Flask on port 5000 and Express on port 3000).
2. **Expected Deliverables**:
   * Terraform configuration files (main.tf, variables.tf, etc.).
   * A working EC2 instance with Flask and Express running and accessible via the instance's public IP.

**Result :-**

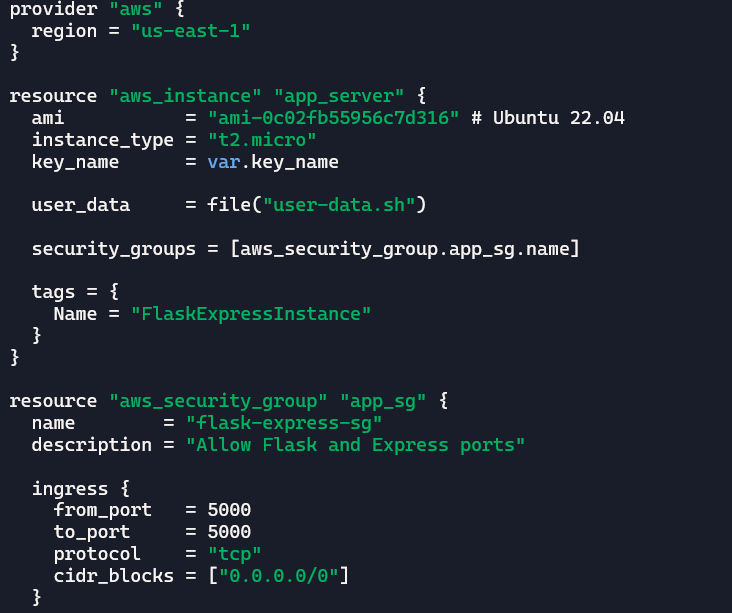
1️. Setup Project Structure

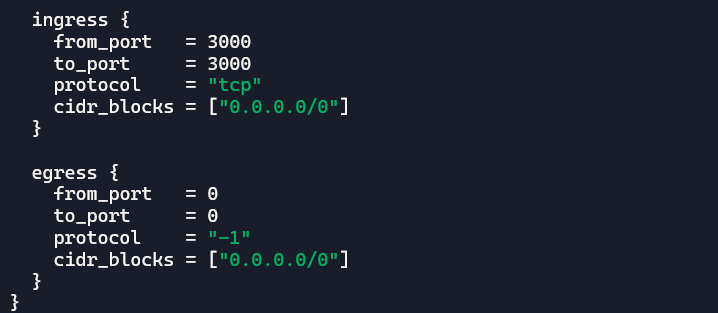
Create a folder



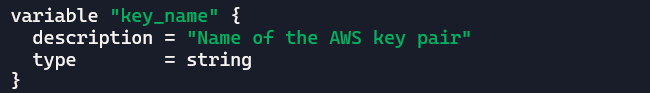
2️. Write main.tf

Provision EC2 with security group and user data





3️. Define [variables.tf](http://variables.tf)

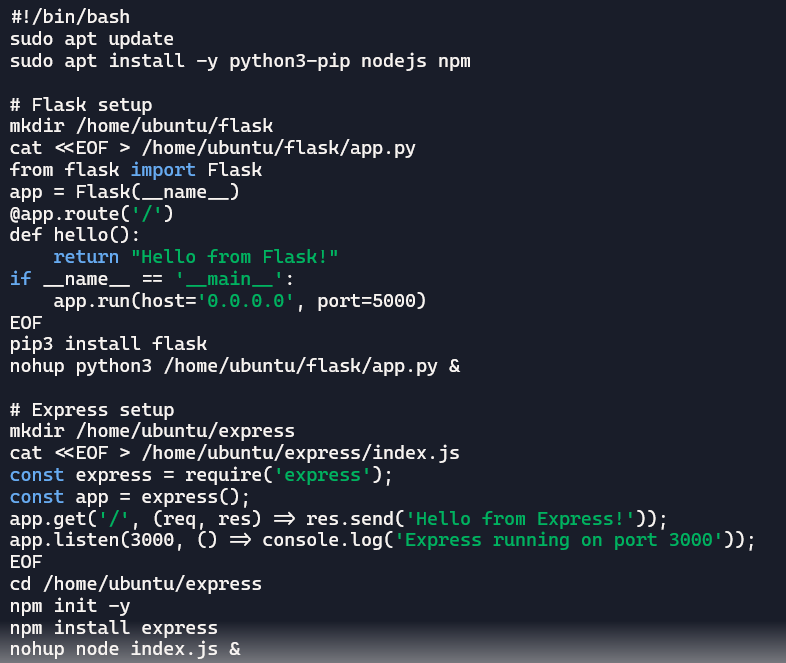


4️. Define [outputs.tf](http://outputs.tf)

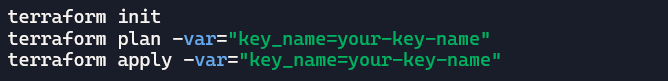


5️. Create [user-data.sh](http://user-data.sh)

This script installs dependencies and runs both apps:



6️. Deploy with Terraform



7️. Test the Deployment

* Get public IP from Terraform output.
* Visit:
* http://:5000 → Flask
* http://:3000 → Express

8. Deliverables

* main.tf, variables.tf, outputs.tf, user-data.sh

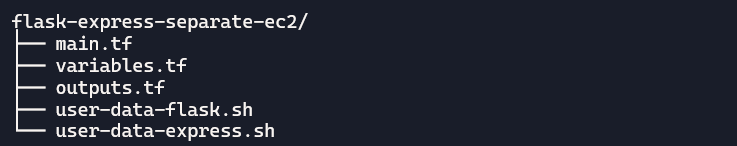
#### **Part 2: Deploy Flask and Express on Separate EC2 Instances**

1. **Objective**: Deploy the Flask backend and the Express frontend on **two separate EC2 instances** using Terraform.
   * Provision two EC2 instances using Terraform:
     + One for the Flask backend.
     + One for the Express frontend.
   * Configure security groups to:
     + Allow communication between the two instances.
     + Expose both applications to the internet on their respective ports.
2. **Configuration**:
   * Use Terraform to define networking resources, such as VPC, subnets, and security groups.
   * Use user data scripts to automate the installation and startup of both applications.
3. **Expected Deliverables**:
   * Terraform configuration files.
   * Two working EC2 instances: one running Flask and one running Express.
   * Security groups configured to allow proper communication and public access.

**Results :-**

**1️. Project Structure**

**Create folders**

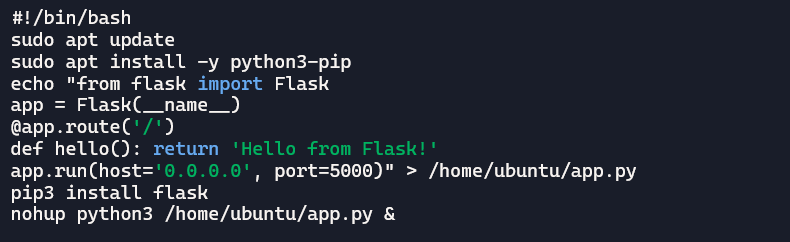
****

**2️. Terraform Setup**

* Create a VPC, subnets, and security groups.
* Provision two EC2 instances:
  + One for Flask (port 5000)
  + One for Express (port 3000)
* Security groups:
* Allow public access to ports 5000 and 3000
* Allow internal communication between instances

**3️. User Data Scripts**

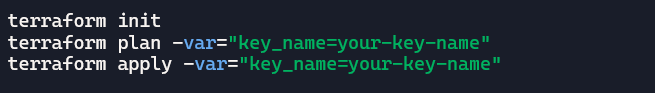
**user-data-flask.sh**

****

[**user-data-express.sh**](http://user-data-express.sh)

****

**4️. Terraform Commands**

****

**5️. Access Apps**

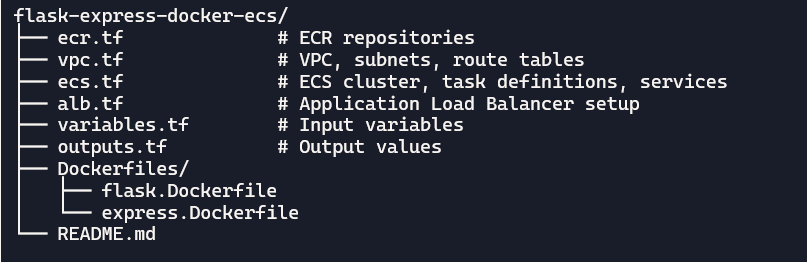
* **Get public IPs from Terraform output.**
* **Visit:**
* **http://:5000**
* [**http://:3000**](about:blank)

#### **Part 3: Deploy Flask and Express Using Docker and AWS Services**

1. **Objective**: Deploy Flask and Express as **Docker containers** using **AWS ECR**, **ECS**, and **VPC** with Terraform.
2. **Steps**:
   * **ECR**:
     + Use Terraform to create two Elastic Container Registry (ECR) repositories: one for the Flask backend and one for the Express frontend.
     + Build Docker images for both applications and push them to their respective ECR repositories.
   * **VPC**:
     + Use Terraform to create a VPC with subnets, route tables, and security groups.
   * **ECS**:
     + Use Terraform to set up an ECS cluster.
     + Create two ECS services:
       - One for the Flask backend.
       - One for the Express frontend.
     + Use ECS Fargate or EC2 launch type to deploy the containers.
   * **Load Balancer**:
     + Use Terraform to provision an Application Load Balancer (ALB).
     + Configure ALB listeners to route requests to the appropriate ECS service.
3. **Expected Deliverables**:
   * Terraform configuration files for ECR, ECS, VPC, and ALB setup.
   * Docker images pushed to ECR.
   * ECS services running and accessible via the ALB.

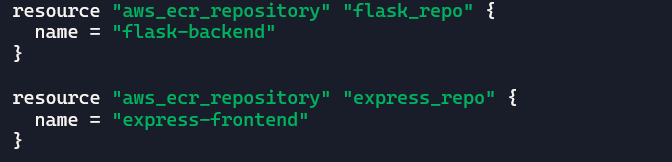
**Results:-**

1. **Create folders structure be like**

****

**2️. Create ECR Repositories**

**Use ecr.tf to define two repositories**

****

**Build and push Docker images:**

# Flask

docker build -t flask-backend -f Dockerfiles/flask.Dockerfile .

docker tag flask-backend <aws\_account\_id>.dkr.ecr.<region>.amazonaws.com/flask-backend

aws ecr get-login-password | docker login --username AWS --password-stdin <ecr-url>

docker push <ecr-url>/flask-backend

# Express

docker build -t express-frontend -f Dockerfiles/express.Dockerfile .

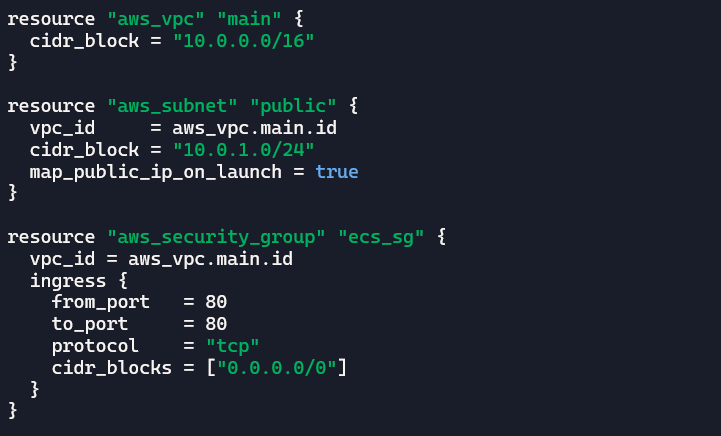
docker tag express-frontend <ecr-url>/express-frontend

docker push <ecr-url>/express-frontend

**3️.Set Up VPC and Networking**

Use vpc.tf to create:

* VPC
* Public subnets
* Internet gateway
* Route tables
* Security groups for ECS and ALB

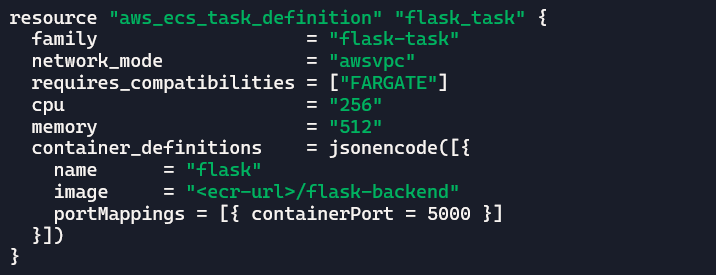


4️. **Create ECS Cluster and Services**

Use ecs.tf to define:

* ECS cluster
* Task definitions for Flask and Express (using ECR images)
* ECS services with Fargate launch type

Example task definition:



Repeat similarly for Express.

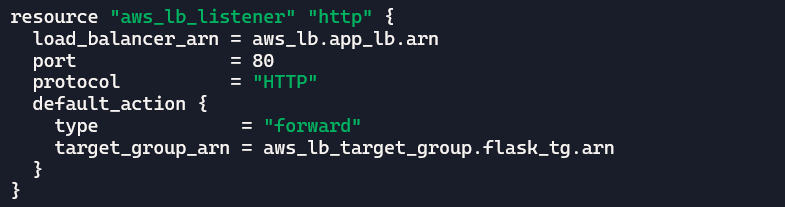
5️. **Configure Application Load Balancer**

Use alb.tf to:

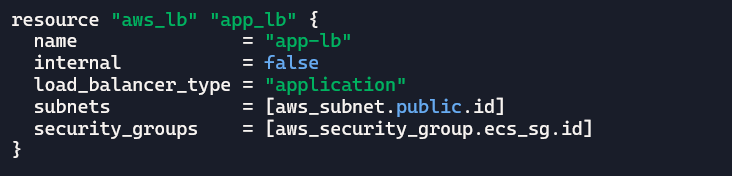
* Create ALB
* Define target groups for Flask and Express
* Set up listeners to route traffic based on path or port

Example listener rule:

**Provision ALB**:



**Configure Listeners and Target Groups**



6️. **Deploy with Terraform**

terraform init terraform plan terraform apply

7. Deliverables

* **Terraform Files**: For ECR, VPC, ECS, and ALB setup. **Docker Images**: Built and pushed to ECR.
* **ECS Services**: Running Flask and Express containers.
* **ALB**: Routing traffic to both services.