**Project Structure Overview**

Here's a suggested folder structure for your GitHub repo:

bash

flask-express-aws-terraform-deployment/

│

├── part1-single-ec2/

│ ├── main.tf

│ ├── variables.tf

│ ├── outputs.tf

│ ├── userdata.sh

│ └── README.md

│

├── part2-separate-ec2/

│ ├── main.tf

│ ├── variables.tf

│ ├── outputs.tf

│ ├── flask\_user\_data.sh

│ ├── express\_user\_data.sh

│ └── README.md

│

├── part3-ecs-docker/

│ ├── ecr.tf

│ ├── ecs.tf

│ ├── alb.tf

│ ├── vpc.tf

│ ├── variables.tf

│ ├── outputs.tf

│ ├── flask/

│ │ ├── Dockerfile

│ │ └── app.py

│ ├── express/

│ │ ├── Dockerfile

│ │ └── app.js

│ └── README.md

│

├── terraform-backend/

│ └── backend.tf

│

├── screenshots/

│ └── (Your Terraform CLI command outputs and EC2 instance test results)

│

└── README.md

**Part 1: Single EC2 Instance Deployment**

**Key Points:**

* Use user\_data.sh to install:
  + Python, pip, Flask
  + Node.js, npm, Express
* Run Flask on port 5000, Express on 3000.

**Terraform Components:**

* main.tf: EC2 + Security Group
* variables.tf: Instance type, AMI, region
* outputs.tf: Public IP

**Sample Snippets:**

**main.tf (snippet):**

hcl

CopyEdit

resource "aws\_instance" "app\_server" {

ami = var.ami\_id

instance\_type = var.instance\_type

key\_name = var.key\_name

user\_data = file("userdata.sh")

security\_groups = [aws\_security\_group.web\_sg.name]

}

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

name\_prefix = "web-sg"

ingress {

from\_port = 3000

to\_port = 3000

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 5000

to\_port = 5000

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"]

}

}

**userdata.sh (snippet):**

bash

CopyEdit

#!/bin/bash

# Install Python & Flask

sudo apt update

sudo apt install -y python3-pip

pip3 install flask

# Install Node.js & Express

curl -sL https://deb.nodesource.com/setup\_16.x | sudo -E bash -

sudo apt install -y nodejs

npm install express

# Start Flask app

nohup python3 /home/ubuntu/app.py &

# Start Express app

nohup node /home/ubuntu/app.js &

**Part 2: Separate EC2 Instances**

**Key Points:**

* Two EC2 instances:
  + Flask: Port 5000
  + Express: Port 3000
* Security group allows intra-communication

**Components:**

* main.tf: 2 EC2 instances, 2 SGs
* flask\_user\_data.sh, express\_user\_data.sh

**Terraform Networking Configuration:**

Use a custom VPC and subnet or default VPC.

**Security Groups:**

* Flask SG allows port 5000 public access
* Express SG allows port 3000 public access and allows internal calls from Flask instance

**Part 3: Docker, ECS, ECR, VPC**

**Key Points:**

* Create 2 ECR Repos: Flask and Express
* Dockerize both apps, push images to ECR
* Use ECS Fargate launch type for deployment
* Setup ALB with listener rules:
  + /api/\* → Flask
  + /app/\* → Express

**Terraform Files:**

* ecr.tf: Creates ECR repos
* ecs.tf: ECS Cluster, Task Definitions, Services
* alb.tf: Load balancer and listeners
* vpc.tf: Custom VPC, subnets, route tables, NAT, IGW

**Dockerfile (Flask):**

Dockerfile

CopyEdit

FROM python:3.9

WORKDIR /app

COPY app.py .

RUN pip install flask

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

**Dockerfile (Express):**

Dockerfile

CopyEdit

FROM node:16

WORKDIR /app

COPY app.js .

RUN npm install express

CMD ["node", "app.js"]

**ALB Listener Rule Example:**

hcl

listener {

port = "80"

protocol = "HTTP"

default\_action {

type = "forward"

target\_group\_arn = aws\_lb\_target\_group.flask.arn

}

}

**Backend Configuration for Terraform**

Use remote backend for managing state in S3:

**backend.tf**

hcl

CopyEdit

terraform {

backend "s3" {

bucket = "your-terraform-state-bucket"

key = "state/app.tfstate"

region = "us-east-1"

dynamodb\_table = "terraform-locks"

}

}