

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.
 - o Provision the EC2 instance with Terraform.
 - o 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.
 - o Ensure both applications are running on different ports (e.g., Flask on port 5000 and Express on port 3000).
2. **Expected Deliverables:**
 - o Terraform configuration files (main.tf, variables.tf, etc.).
 - o A working EC2 instance with Flask and Express running and accessible via the instance's public IP.



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.

- o Provision two EC2 instances using Terraform:
 - One for the Flask backend.
 - One for the Express frontend.
- o Configure security groups to:
 - Allow communication between the two instances.
 - Expose both applications to the internet on their respective ports.

2. Configuration:

- o Use Terraform to define networking resources, such as VPC, subnets, and security groups.
- o Use user data scripts to automate the installation and startup of both applications.

3. Expected Deliverables:

- o Terraform configuration files.
- o Two working EC2 instances: one running Flask and one running Express.
- o Security groups configured to allow proper communication and public access.

```
    },
  ],
+ name                  = "express-frontend-sg-final-001"
+ name_prefix           = "(known after apply)"
+ owner_id              = "(known after apply)"
+ revoke_rules_on_delete = false
+ tags_all              = "(known after apply)"
+ vpc_id                = "(known after apply)"
}

Plan: 4 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ express_public_ip = (known after apply)
+ flask_public_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_security_group.frontend_sg: Creating...
aws_security_group.frontend_sg: Creation complete after 6s [id=sg-0000b996aebd4c4c02]
aws_security_group.backend_sg: Creating...
aws_instance.express_server: Creating...
aws_security_group.backend_sg: Creation complete after 7s [id=sg-0eeaf8d00008dea28]
aws_instance.flask_server: Creating...
aws_instance.express_server: Still creating... [00m10s elapsed]
aws_instance.flask_server: Still creating... [00m10s elapsed]
aws_instance.express_server: Creation complete after 17s [id=i-0528254b91156b10c]
aws_instance.flask_server: Creation complete after 16s [id=i-0af04c5c141f86da5]

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

express_public_ip = "3.98.189.172"
flask_public_ip  = "100.53.33.242"
C:\Users\HARSHINI B\separate-ec2>
```

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:

- o **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.

```

}

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

Changes to Outputs:
+ express_ecr_url = (known after apply)
+ flask_ecr_url   = (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_ecr_repository.express_repo: Creating...
aws_ecr_repository.flask_repo: Creating...
aws_ecr_repository.express_repo: Creation complete after 3s [id=express-frontend-repo]
aws_ecr_repository.flask_repo: Creation complete after 3s [id=flask-backend-repo]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

express_ecr_url = "904233091541.dkr.ecr.us-east-1.amazonaws.com/express-frontend-repo"
flask_ecr_url   = "904233091541.dkr.ecr.us-east-1.amazonaws.com/flask-backend-repo"

C:\Users\HARSHINI B\part3-ecs>

```

```
Command Prompt X + - 
=> transferring dockerfile: 171B 0.1s
=> [internal] load metadata for docker.io/library/node:18 0.1s
=> [internal] load .dockerignore 0.1s
=> transferring context: 28 0.0s
=> [1/5] FROM docker.io/library/node:18@sha256:c6ae79e38498325db67193d391e6ec1d224d96c693a8a4d943498556716d3783 0.1s
=> resolve docker.io/library/node:18@sha256:c6ae79e38498325db67193d391e6ec1d224d96c693a8a4d943498556716d3783 0.1s
=> [internal] load build context 0.1s
=> transferring context: 450B 0.3s
=> CACHED [2/5] WORKDIR /app 0.2s
=> [3/5] COPY package.json . 0.0s
=> [4/5] RUN npm install 2.9s
=> [5/5] COPY server.js . 0.2s
=> exporting to image 1.2s
=> exporting layers 0.0s
=> exporting manifest sha256:db7f7e10934c30c45d85572214aaa0198a393d71565d49c423d5697bb0d22a1c 0.0s
=> exporting config sha256:1c990hca4d1eaelb554d3be03f8c44e6aced96dff84e2a627459d5502b102e8b 0.0s
=> exporting attestation manifest sha256:f5d27ff17c2ea387e889c3fb2ae66fa128c2e4165c4ffd58d781b5329c0b67781 0.1s
=> exporting manifest list sha256:a8fd8fc7cc396463863694c2eca36615419ccf83b8d048aa7f0f7fabc0945597 0.0s
=> naming to docker.io/library/express-frontend:latest 0.0s
=> unpacking to docker.io/library/express-frontend:latest 1.2s

C:\Users\HARSHINI\B\part3-ecs\express-app>docker tag express-frontend:latest 904233091541.dkr.ecr.us-east-1.amazonaws.com/express-frontend-repo:latest
C:\Users\HARSHINI\B\part3-ecs\express-app>docker push 904233091541.dkr.ecr.us-east-1.amazonaws.com/express-frontend-repo:latest
The push refers to repository [904233091541.dkr.ecr.us-east-1.amazonaws.com/express-frontend-repo]
2b44cb4439ec: Pushed
187839213657: Pushed
c6ede8f5e28a: Pushed
37927ed901b1: Pushed
3697be50c98b: Pushed
e23f099911d6: Pushed
834e4e5087c7: Pushed
ef6ed9a1c23: Pushed
cda7f44f2bbd: Pushed
461077a72fb7: Pushed
79b2f47ad444: Pushed
c6b30c3f1696: Pushed
3e6b9d1a9511: Pushed
latest: digest: sha256:a8fd8fc7cc396463863694c2eca36615419ccf83b8d048aa7f0f7fabc0945597 size: 856

C:\Users\HARSHINI\B\part3-ecs\express-app>
```