# 🧾 Hackathon DevOps Project Documentation

## 📁 Project Structure (Monorepo)

**mkdir -p /hackathon-devops/{appointment-service,patient-service,terraform/environments/{dev,prod,staging},terraform/modules/{alb,ecs,iam,network,securitygroup},.github/workflows}**

**# Create empty files**

**touch {Dockerfile,index.js,package.json}**

**touch {Dockerfile,index.js,package.json}**

**touch {backend.tf,main.tf}**

**touch root/hackathon-devops/.github/workflows/{appointment.yml,patient.yml}**

root/hackathon-devops/

├── appointment-service/

│ ├── Dockerfile

│ ├── index.js

│ └── package.json

├── patient-service/

│ ├── Dockerfile

│ ├── index.js

│ └── package.json

├── terraform/

│ ├── environments/

│ │ ├── dev/

│ │ │ ├── backend.tf

│ │ │ └── main.tf

│ │ ├── prod/

│ │ └── staging/

│ └── modules/

│ ├── alb/

│ ├── ecs/

│ ├── iam/

│ ├── network/

│ └── securitygroup/

└── .github/

└── workflows/

├── appointment.yml

└── patient.yml

---

## 📦 appointment-service

### Dockerfile

```dockerfile

FROM node:18-alpine

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 80

CMD ["npm", "start"]

**index.js**

**JavaScript**

Copy

const express = require('express');

const app = express();

const port = process.env.PORT || 80;

const host = '0.0.0.0';

app.get('/appointment', (req, res) => {

res.send('✅ Appointment Service running on port 80!');

});

app.listen(port, host, () => {

console.log(`✅ Appointment service running on http://${host}:${port}`);

});

**package.json**

**JSON**

Copy

{

"name": "appointment-service",

"version": "1.0.0",

"main": "index.js",

"scripts": {

"start": "node index.js"

},

"dependencies": {

"express": "^4.18.2"

}

}

**📦 patient-service**

**Dockerfile**

**dockerfile**

Copy

FROM node:18-alpine

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 80

CMD ["npm", "start"]

**index.js**

**JavaScript**

Copy

const express = require('express');

const app = express();

const port = process.env.PORT || 80;

const host = '0.0.0.0';

app.get('/patient', (req, res) => {

res.send('✅ Patient Service running on port 80!');

});

app.listen(port, host, () => {

console.log(`✅ Patient service running on http://${host}:${port}`);

});

**package.json**

**JSON**

Copy

{

"name": "patient-service",

"version": "1.0.0",

"main": "index.js",

"scripts": {

"start": "node index.js"

},

"dependencies": {

"express": "^4.18.2"

}

}

**🧱 Terraform Structure**

**environments/dev/main.tf**

**hcl**

Copy

provider "aws" {

region = "ap-south-1"

}

module "network" {

source = "../../modules/network"

env = "dev"

vpc\_cidr = "10.0.0.0/16"

public\_subnet\_cidrs = ["10.0.1.0/24", "10.0.2.0/24"]

private\_subnet\_cidrs = ["10.0.3.0/24", "10.0.4.0/24"]

azs = ["ap-south-1a", "ap-south-1b"]

}

module "iam" {

source = "../../modules/iam"

env = "dev"

aws\_account\_id = "373649774472"

}

module "sg" {

source = "../../modules/securitygroup"

env = "dev"

vpc\_id = module.network.vpc\_id

}

module "alb" {

source = "../../modules/alb"

env = "dev"

vpc\_id = module.network.vpc\_id

public\_subnet\_ids = module.network.public\_subnet\_ids

alb\_sg\_id = module.sg.alb\_sg\_id

}

module "ecs" {

source = "../../modules/ecs"

env = "dev"

region = "ap-south-1"

execution\_role\_arn = module.iam.ecs\_task\_execution\_role\_arn

task\_role\_arn = module.iam.ecs\_task\_role\_arn

patient\_image = "373649774472.dkr.ecr.ap-south-1.amazonaws.com/patient-service:latest"

appointment\_image = "373649774472.dkr.ecr.ap-south-1.amazonaws.com/appointment-service:latest"

private\_subnet\_ids = module.network.private\_subnet\_ids

sg\_id = module.sg.ecs\_sg\_id

patient\_tg\_arn = module.alb.patient\_tg\_arn

appointment\_tg\_arn = module.alb.appointment\_tg\_arn

}

**environments/dev/backend.tf**

**hcl**

Copy

terraform {

backend "s3" {

bucket = "balu-terraform-backend"

key = "dev/terraform.tfstate"

region = "ap-south-1"

dynamodb\_table = "terraform-locks"

encrypt = true

}

}

**📁 Terraform Modules**

**modules/alb**

**main.tf**

resource "aws\_lb" "this" {

name = "${var.env}-alb"

internal = false

load\_balancer\_type = "application"

security\_groups = [var.alb\_sg\_id]

subnets = var.public\_subnet\_ids

enable\_deletion\_protection = false

tags = {

Environment = var.env

}

}

resource "aws\_lb\_target\_group" "patient" {

name = "${var.env}-tg-patient"

port = 80

protocol = "HTTP"

target\_type = "ip"

vpc\_id = var.vpc\_id

health\_check {

path = "/patient"

protocol = "HTTP"

interval = 30

timeout = 5

healthy\_threshold = 2

unhealthy\_threshold = 2

matcher = "200-399"

}

}

resource "aws\_lb\_target\_group" "appointment" {

name = "${var.env}-tg-appointment"

port = 80

protocol = "HTTP"

target\_type = "ip"

vpc\_id = var.vpc\_id

health\_check {

path = "/appointment"

protocol = "HTTP"

interval = 30

timeout = 5

healthy\_threshold = 2

unhealthy\_threshold = 2

matcher = "200-399"

}

}

resource "aws\_lb\_listener" "http" {

load\_balancer\_arn = aws\_lb.this.arn

port = 80

protocol = "HTTP"

default\_action {

type = "fixed-response"

fixed\_response {

content\_type = "text/plain"

message\_body = "Default ALB response"

status\_code = "200"

}

}

}

resource "aws\_lb\_listener\_rule" "patient\_rule" {

listener\_arn = aws\_lb\_listener.http.arn

priority = 100

action {

type = "forward"

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

}

condition {

path\_pattern {

values = ["/patient\*"]

}

}

}

resource "aws\_lb\_listener\_rule" "appointment\_rule" {

listener\_arn = aws\_lb\_listener.http.arn

priority = 200

action {

type = "forward"

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

}

condition {

path\_pattern {

values = ["/appointment\*"]

}

}

}

**outputs.tf**

output "alb\_dns\_name" {

value = aws\_lb.this.dns\_name

}

output "patient\_target\_group\_arn" {

value = aws\_lb\_target\_group.patient.arn

}

output "appointment\_target\_group\_arn" {

value = aws\_lb\_target\_group.appointment.arn

}

output "patient\_tg\_arn" {

value = aws\_lb\_target\_group.patient.arn

}

output "appointment\_tg\_arn" {

value = aws\_lb\_target\_group.appointment.arn

}

**variables.tf**

variable "env" {}

variable "vpc\_id" {}

variable "public\_subnet\_ids" {

type = list(string)

}

variable "alb\_sg\_id" {}

**modules/ecs**

**main.tf**

resource "aws\_ecs\_cluster" "this" {

name = "${var.env}-ecs-cluster"

}

resource "aws\_cloudwatch\_log\_group" "patient" {

name = "/ecs/patient"

retention\_in\_days = 7

tags = {

Name = "patient-log-group"

}

}

resource "aws\_cloudwatch\_log\_group" "appointment" {

name = "/ecs/appointment"

retention\_in\_days = 7

tags = {

Name = "appointment-log-group"

}

}

resource "aws\_ecs\_task\_definition" "patient" {

family = "patient-task"

network\_mode = "awsvpc"

requires\_compatibilities = ["FARGATE"]

cpu = "256"

memory = "512"

execution\_role\_arn = var.execution\_role\_arn

task\_role\_arn = var.task\_role\_arn

container\_definitions = jsonencode([{

name = "patient"

image = var.patient\_image

portMappings = [{

containerPort = 80

protocol = "tcp"

}]

essential = true

logConfiguration = {

logDriver = "awslogs"

options = {

awslogs-group = "/ecs/patient"

awslogs-region = var.region

awslogs-stream-prefix = "ecs"

}

}

}])

}

resource "aws\_ecs\_service" "patient" {

name = "patient-service"

cluster = aws\_ecs\_cluster.this.id

task\_definition = aws\_ecs\_task\_definition.patient.arn

launch\_type = "FARGATE"

desired\_count = 1

load\_balancer {

target\_group\_arn = var.patient\_tg\_arn

container\_name = "patient"

container\_port = 80

}

network\_configuration {

subnets = var.private\_subnet\_ids

security\_groups = [var.sg\_id]

assign\_public\_ip = false

}

depends\_on = [

aws\_ecs\_task\_definition.patient,

aws\_cloudwatch\_log\_group.patient

]

}

resource "aws\_ecs\_task\_definition" "appointment" {

family = "appointment-task"

network\_mode = "awsvpc"

requires\_compatibilities = ["FARGATE"]

cpu = "256"

memory = "512"

execution\_role\_arn = var.execution\_role\_arn

task\_role\_arn = var.task\_role\_arn

container\_definitions = jsonencode([{

name = "appointment"

image = var.appointment\_image

portMappings = [{

containerPort = 80

protocol = "tcp"

}]

essential = true

logConfiguration = {

logDriver = "awslogs"

options = {

awslogs-group = "/ecs/appointment"

awslogs-region = var.region

awslogs-stream-prefix = "ecs"

}

}

}])

}

resource "aws\_ecs\_service" "appointment" {

name = "appointment-service"

cluster = aws\_ecs\_cluster.this.id

task\_definition = aws\_ecs\_task\_definition.appointment.arn

launch\_type = "FARGATE"

desired\_count = 1

load\_balancer {

target\_group\_arn = var.appointment\_tg\_arn

container\_name = "appointment"

container\_port = 80

}

network\_configuration {

subnets = var.private\_subnet\_ids

security\_groups = [var.sg\_id]

assign\_public\_ip = true

}

depends\_on = [

aws\_ecs\_task\_definition.appointment,

aws\_cloudwatch\_log\_group.appointment

]

}

**outputs.tf**

output "ecs\_cluster\_id" {

value = aws\_ecs\_cluster.this.id

}

**variables.tf**

variable "env" {}

variable "region" {}

variable "execution\_role\_arn" {}

variable "task\_role\_arn" {}

variable "patient\_image" {}

variable "appointment\_image" {}

variable "private\_subnet\_ids" {

type = list(string)

}

variable "sg\_id" {}

variable "patient\_tg\_arn" {

description = "Target group ARN for patient service"

type = string

}

variable "appointment\_tg\_arn" {

description = "Target group ARN for appointment service"

type = string

}

**modules/iam**

**main.tf**

resource "aws\_iam\_role" "ecs\_task\_execution\_role" {

name = "${var.env}-ecs-task-execution-role"

assume\_role\_policy = jsonencode({

Version = "2012-10-17",

Statement = [

{

Action = "sts:AssumeRole",

Principal = {

Service = "ecs-tasks.amazonaws.com"

},

Effect = "Allow",

Sid = ""

}

]

})

tags = {

Name = "${var.env}-ecs-task-execution-role"

}

}

resource "aws\_iam\_role\_policy\_attachment" "ecs\_task\_execution\_policy" {

role = aws\_iam\_role.ecs\_task\_execution\_role.name

policy\_arn = "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"

}

resource "aws\_iam\_policy" "ecs\_logs\_scoped\_policy" {

name = "${var.env}-ecs-logs-scoped"

policy = jsonencode({

Version = "2012-10-17",

Statement = [

{

Effect = "Allow",

Action = ["logs:CreateLogGroup"],

Resource = [

"arn:aws:logs:ap-south-1:${var.aws\_account\_id}:log-group:/ecs/patient-service:\*",

"arn:aws:logs:ap-south-1:${var.aws\_account\_id}:log-group:/ecs/appointment-service:\*"

]

},

{

Effect = "Allow",

Action = [

"logs:CreateLogStream",

"logs:PutLogEvents"

],

Resource = [

"arn:aws:logs:ap-south-1:${var.aws\_account\_id}:log-group:/ecs/patient-service:\*",

"arn:aws:logs:ap-south-1:${var.aws\_account\_id}:log-group:/ecs/appointment-service:\*"

]

}

]

})

}

resource "aws\_iam\_role\_policy\_attachment" "attach\_logs\_scoped" {

role = aws\_iam\_role.ecs\_task\_execution\_role.name

policy\_arn = aws\_iam\_policy.ecs\_logs\_scoped\_policy.arn

}

resource "aws\_iam\_role" "ecs\_task\_role" {

name = "${var.env}-ecs-task-role"

assume\_role\_policy = jsonencode({

Version = "2012-10-17",

Statement = [

{

Action = "sts:AssumeRole",

Principal = {

Service = "ecs-tasks.amazonaws.com"

},

Effect = "Allow",

Sid = ""

}

]

})

tags = {

Name = "${var.env}-ecs-task-role"

}

}

**outputs.tf**

output "ecs\_task\_execution\_role\_arn" {

value = aws\_iam\_role.ecs\_task\_execution\_role.arn

}

output "ecs\_task\_role\_arn" {

value = aws\_iam\_role.ecs\_task\_role.arn

}

**variables.tf**

variable "env" {

type = string

description = "Environment name"

}

variable "aws\_account\_id" {

description = "Your AWS account ID"

type = string

}

**modules/network**

**main.tf**

resource "aws\_vpc" "main" {

cidr\_block = var.vpc\_cidr

enable\_dns\_hostnames = true

enable\_dns\_support = true

tags = {

Name = "${var.env}-vpc"

}

}

resource "aws\_internet\_gateway" "igw" {

vpc\_id = aws\_vpc.main.id

tags = {

Name = "${var.env}-igw"

}

}

resource "aws\_subnet" "public" {

count = length(var.public\_subnet\_cidrs)

vpc\_id = aws\_vpc.main.id

cidr\_block = var.public\_subnet\_cidrs[count.index]

availability\_zone = element(var.azs, count.index)

map\_public\_ip\_on\_launch = true

tags = {

Name = "${var.env}-public-subnet-${count.index + 1}"

}

}

resource "aws\_subnet" "private" {

count = length(var.private\_subnet\_cidrs)

vpc\_id = aws\_vpc.main.id

cidr\_block = var.private\_subnet\_cidrs[count.index]

availability\_zone = element(var.azs, count.index)

tags = {

Name = "${var.env}-private-subnet-${count.index + 1}"

}

}

resource "aws\_eip" "nat" {}

resource "aws\_nat\_gateway" "nat" {

allocation\_id = aws\_eip.nat.id

subnet\_id = aws\_subnet.public[0].id

tags = {

Name = "${var.env}-nat"

}

}

resource "aws\_route\_table" "public" {

vpc\_id = aws\_vpc.main.id

tags = {

Name = "${var.env}-public-rt"

}

}

resource "aws\_route" "public\_internet\_access" {

route\_table\_id = aws\_route\_table.public.id

destination\_cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.igw.id

}

resource "aws\_route\_table\_association" "public" {

count = length(aws\_subnet.public)

subnet\_id = aws\_subnet.public[count.index].id

route\_table\_id = aws\_route\_table.public.id

}

resource "aws\_route\_table" "private" {

vpc\_id = aws\_vpc.main.id

tags = {

Name = "${var.env}-private-rt"

}

}

resource "aws\_route" "private\_nat\_gateway" {

route\_table\_id = aws\_route\_table.private.id

destination\_cidr\_block = "0.0.0.0/0"

nat\_gateway\_id = aws\_nat\_gateway.nat.id

}

resource "aws\_route\_table\_association" "private" {

count = length(aws\_subnet.private)

subnet\_id = aws\_subnet.private[count.index].id

route\_table\_id = aws\_route\_table.private.id

}

**outputs.tf**

output "vpc\_id" {

value = aws\_vpc.main.id

}

output "public\_subnet\_ids" {

value = aws\_subnet.public[\*].id

}

output "private\_subnet\_ids" {

value = aws\_subnet.private[\*].id

}

**variables.tf**

variable "vpc\_cidr" {

type = string

description = "VPC CIDR block"

}

variable "public\_subnet\_cidrs" {

type = list(string)

description = "List of public subnet CIDRs"

}

variable "private\_subnet\_cidrs" {

type = list(string)

description = "List of private subnet CIDRs"

}

variable "azs" {

type = list(string)

description = "Availability zones"

}

variable "env" {

type = string

description = "Environment name (dev/staging/prod)"

}

**modules/securitygroup**

**main.tf**

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

name = "${var.env}-alb-sg"

description = "Allow HTTP traffic from internet"

vpc\_id = var.vpc\_id

ingress {

from\_port = 80

to\_port = 80

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 = "${var.env}-alb-sg"

}

}

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

name = "${var.env}-ecs-sg"

description = "Allow traffic from ALB to ECS containers"

vpc\_id = var.vpc\_id

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "${var.env}-ecs-sg"

}

}

resource "aws\_security\_group\_rule" "allow\_alb\_to\_ecs" {

type = "ingress"

from\_port = 80

to\_port = 80

protocol = "tcp"

security\_group\_id = aws\_security\_group.ecs\_sg.id

source\_security\_group\_id = aws\_security\_group.alb\_sg.id

description = "Allow HTTP from ALB to ECS"

}

**outputs.tf**

output "ecs\_sg\_id" {

value = aws\_security\_group.ecs\_sg.id

}

output "alb\_sg\_id" {

value = aws\_security\_group.alb\_sg.id

}

**variables.tf**

variable "env" {

description = "Environment"

type = string

}

variable "vpc\_id" {

description = "VPC ID"

type = string

}

**🚀 GitHub Actions Workflows**

**.github/workflows/appointment.yml**

**yaml**

Copy

name: Build and Deploy Appointment Service

on:

push:

branches: [main]

jobs:

deploy:

runs-on: ubuntu-latest

env:

IMAGE\_URI: ${{ secrets.ECR\_REPO\_APPOINTMENT }}:latest

steps:

- uses: actions/checkout@v3

- uses: aws-actions/configure-aws-credentials@v2

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: ${{ secrets.AWS\_REGION }}

- run: |

aws ecr get-login-password --region $AWS\_REGION | \

docker login --username AWS --password-stdin ${{ secrets.ECR\_REPO\_APPOINTMENT }}

- run: |

cd appointment-service

docker build -t $IMAGE\_URI .

docker push $IMAGE\_URI

- run: |

aws ecs update-service \

--cluster ${{ secrets.CLUSTER\_NAME }} \

--service ${{ secrets.APPOINTMENT\_SERVICE\_NAME }} \

--force-new-deployment

**.github/workflows/patient.yml**

**yaml**

Copy

name: Build and Deploy Patient Service

on:

push:

branches: [main]

jobs:

deploy:

runs-on: ubuntu-latest

env:

IMAGE\_URI: ${{ secrets.ECR\_REPO\_PATIENT }}:latest

steps:

- uses: actions/checkout@v3

- uses: aws-actions/configure-aws-credentials@v2

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: ${{ secrets.AWS\_REGION }}

- run: |

aws ecr get-login-password --region $AWS\_REGION | \

docker login --username AWS --password-stdin ${{ secrets.ECR\_REPO\_PATIENT }}

- run: |

cd patient-service

docker build -t $IMAGE\_URI .

docker push $IMAGE\_URI

- run: |

aws ecs update-service \

--cluster ${{ secrets.CLUSTER\_NAME }} \

--service ${{ secrets.PATIENT\_SERVICE\_NAME }} \

--force-new-deployment