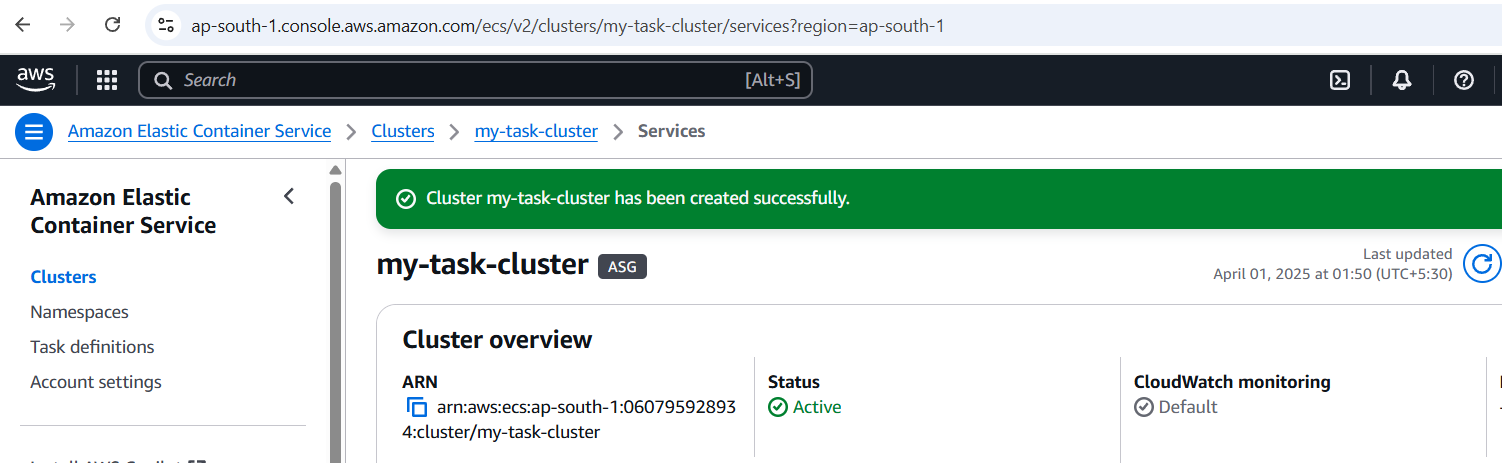
**Create Terraform module structure**



**Step-1. VPC, ECR, and ECS with the required variables.tf,**

**outputs.tf,**

**main.tf files**

**for each module.**

**Step -2 .Paste code of respective module in their .tf file**

**Step -3 Run below command**

**terraform init**

**terraform plan**

**Directory Structure**

**terraform-project/**

**│── main.tf**

**│── variables.tf**

**│── outputs.tf**

**│── modules/**

**│ ├── vpc/**

**│ │ ├── main.tf**

**│ │ ├── variables.tf**

**│ │ ├── outputs.tf**

**│ ├── ecr/**

**│ │ ├── main.tf**

**│ │ ├── variables.tf**

**│ │ ├── outputs.tf**

**│ ├── ecs/**

**│ │ ├── main.tf**

**│ │ ├── variables.tf**

**│ │ ├── outputs.tf**

**ECS Cluster Documentation**

**1. ECS Cluster**

**Resource: aws\_ecs\_cluster**

**Description: Creates an ECS cluster to run containerized applications.**

**Code Explanation:**

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

name = var.cluster\_name

}

**name: Defines the name of the ECS cluster.**

**2. IAM Roles and Policies**

**ECS Instance Role**

**Resource: aws\_iam\_role**

**Description: Creates an IAM role for ECS EC2 instances with necessary permissions to interact with ECS services.**

**Code Explanation:**

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

name = "ecs-instance-role"

assume\_role\_policy = jsonencode({

Version = "2012-10-17"

Statement = [

{

Action = "sts:AssumeRole"

Effect = "Allow"

Principal = {

Service = "ec2.amazonaws.com"

}

}

]

})

}

**name: The IAM role name.**

**assume\_role\_policy: Allows EC2 instances to assume this role.**

**2.Attach Policy to ECS Instance Role**

**Resource: aws\_iam\_role\_policy\_attachment**

**Description: Attaches a policy to the ECS instance role to allow access to ECS and related services.**

**Code Explanation:**

resource "aws\_iam\_role\_policy\_attachment" "ecs\_instance\_role\_policy" {

role = aws\_iam\_role.ecs\_instance\_role.name

policy\_arn = "arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role"

}

**role: The ECS instance role to attach the policy to.**

**policy\_arn: The policy ARN that grants ECS permissions.**

**3.ECS Instance Profile**

**Resource: aws\_iam\_instance\_profile**

**Description: Creates an instance profile associated with the ECS instance role.**

**Code Explanation:**

resource "aws\_iam\_instance\_profile" "ecs\_instance\_profile" {

name = "ecs-instance-profile"

role = aws\_iam\_role.ecs\_instance\_role.name

}

**role: The IAM role associated with the instance profile.**

**4. Launch Template**

**Resource: aws\_launch\_template**

**Description: Creates a launch template to define the configuration for EC2 instances running ECS tasks.**

**Code Explanation:**

resource "aws\_launch\_template" "ecs\_lt" {

name\_prefix = "ecs-template"

image\_id = "ami-01bd9d8f06d29d6a0" # Amazon ECS-optimized AMI

instance\_type = "t2.micro"

user\_data = base64encode(<<-EOF

#!/bin/bash

echo ECS\_CLUSTER=${aws\_ecs\_cluster.this.name} >> /etc/ecs/ecs.config

EOF

)

iam\_instance\_profile {

name = aws\_iam\_instance\_profile.ecs\_instance\_profile.name

}

vpc\_security\_group\_ids = [var.security\_group\_id]

}

**image\_id: Specifies the ECS-optimized Amazon Machine Image (AMI) for EC2 instances.**

**user\_data: Configures EC2 instances to join the ECS cluster.**

**5. Auto Scaling Group**

**Resource: aws\_autoscaling\_group**

**Description: Creates an Auto Scaling group to manage EC2 instances for the ECS cluster.**

**Code Explanation:**

resource "aws\_autoscaling\_group" "ecs\_asg" {

name = "ecs-asg"

vpc\_zone\_identifier = var.subnet\_ids

desired\_capacity = 1

max\_size = 1

min\_size = 1

launch\_template {

id = aws\_launch\_template.ecs\_lt.id

version = "$Latest"

}

tag {

key = "AmazonECSManaged"

value = true

propagate\_at\_launch = true

}

}

**desired\_capacity: Defines the number of EC2 instances in the Auto Scaling group.**

**6. ECS Task Definition**

**Resource: aws\_ecs\_task\_definition**

**Description: Defines the task definition for running the Nginx container in ECS.**

**Code Explanation:**

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

family = "nginx"

requires\_compatibilities = ["EC2"]

network\_mode = "bridge"

container\_definitions = jsonencode([{

name = "nginx"

image = "${var.ecr\_repository\_url}:latest"

memory = 512

cpu = 256

essential = true

portMappings = [{

containerPort = 80

hostPort = 80

protocol = "tcp"

}]

}])

}

**container\_definitions: Specifies the container (Nginx) and its configuration (e.g., memory, CPU, ports).**

**7. ECS Service**

**Resource: aws\_ecs\_service**

**Description: Creates an ECS service to manage the Nginx task.**

**Code Explanation:**

resource "aws\_ecs\_service" "nginx\_service" {

name = "nginx-service"

cluster = aws\_ecs\_cluster.this.id

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

desired\_count = "1"

launch\_type = "EC2"

}

**desired\_count: Specifies the number of tasks (containers) to run.**

**launch\_type: Specifies the EC2 launch type.**

**8. CloudWatch Metric Alarms**

**CPU Utilization Alarm**

**Resource: aws\_cloudwatch\_metric\_alarm**

**Description: Creates an alarm for CPU utilization exceeding 80%.**

**Code Explanation:**

resource "aws\_cloudwatch\_metric\_alarm" "cpu\_alarm" {

alarm\_name = "cpu\_utilization\_high"

comparison\_operator = "GreaterThanThreshold"

evaluation\_periods = "1"

metric\_name = "CPUUtilization"

namespace = "AWS/ECS"

period = "60"

statistic = "Average"

threshold = "80"

alarm\_description = "This alarm triggers when CPU utilization exceeds 80%."

dimensions = {

ClusterName = aws\_ecs\_cluster.this.name

ServiceName = aws\_ecs\_service.nginx\_service.name

}

}

**threshold: Defines the CPU utilization threshold to trigger the alarm.**

**9 .Memory Utilization Alarm**

**Resource: aws\_cloudwatch\_metric\_alarm**

**Description: Creates an alarm for memory utilization exceeding 80%.**

**Code Explanation:**

resource "aws\_cloudwatch\_metric\_alarm" "memory\_alarm" {

alarm\_name = "memory\_utilization\_high"

comparison\_operator = "GreaterThanThreshold"

evaluation\_periods = "1"

metric\_name = "MemoryUtilization"

namespace = "AWS/ECS"

period = "60"

statistic = "Average"

threshold = "80"

alarm\_description = "This alarm triggers when memory utilization exceeds 80%."

dimensions = {

ClusterName = aws\_ecs\_cluster.this.name

ServiceName = aws\_ecs\_service.nginx\_service.name

}

}

**threshold: Defines the memory utilization threshold to trigger the alarm.**

**This output will return the URL which can be used to check if the container is accessible via the load balancer.**

**Then, open a browser or use curl:**

**curl http://<public-ip>**

**This will show the Nginx default page, confirming the container is running.**

**Screenshot**

