====Use case of AWS ECS service===

**Purpose:**

ECS cluster created with EC2 + Networking template to deploy NGINX container image and Scale out the instance in case target/CPU capacity increase up to 85% and Application load balancer is attached to the service, so the traffic can be distributed between tasks.

# Provisioning VPC, ECS, ELB, S3 Bucket, DynamoDB using Terraform.

Create a simple infrastructure using Terraform and AWS cloud provider. It consists of:

- Virtual Private Cloud (VPC) with 3 public subnets in 3 availability zones.

- Created 3 EC2 instances for ECS cluster.

- created Auto scalling group to scale-in and scale-out actions for EC2.

- Elastic Container Service (ECS).

- ELB for Application Load Balancer (ALB).

- S3 Bucket to upload terraform.tfstate file remotely.

- DynamoDB to maintain the terraform state file in encrypted format.

**AWS resource description for Terraform files.**

In file `**iam.tf`** have create roles, which will help us to associate EC2 instances to clusters, and other tasks.

In file `**alb.tf**` have create Application Load Balancer with target groups, security group and listener.

In file `**asg.tf**`will create:

- launch configuration

- key pair

- security groups for EC2 instances

- auto-scaling group.

Resource output value have define in **output.tf** .

Resource variable defined in **variable.tf** file

In **main.tf** have defined provider, S3 bucket, DynamoDB and Terraform backend as S3.

Terraform Commands to build the AWS infra.

1. terraform init

2. terraform plan

3. terraform apply

Terraform Commands to destroy the AWS infra.

1. Terminate instances

2. Run `terraform destroy`

## Brief Description

Cluster is created using container instances (EC2 launch type).

In this example, verified module `vpc` is imported from Terraform Registry, other resources are created in relevant files.

In file `**ecs.tf`** we create:

- cluster of container instances \_web-cluster\_

- capacity provider, which is basically AWS Autoscaling group for EC2 instances. In this example managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group used when creating the capacity provider. I set target capacity to 85%, which will result in the Amazon EC2 instances in Auto Scaling group being utilized for 85% and any instances not running any tasks will be scaled in.

- task definition with family \_web-family\_, volume and container definition is defined in the file container-def.json

- service \_web-service\_, desired count is set to 10, which means there are 10 tasks will be running simultaneously on your cluster. There are two service scheduler strategies available: REPLICA and DAEMON, in this example REPLICA is used. Application load balancer is attached to this service, so the traffic can be distributed between those tasks.

**Note:** The \_binpack\_ task placement strategy is used, which places tasks on available instances that have the least available amount of the cpu (specified with the field parameter).