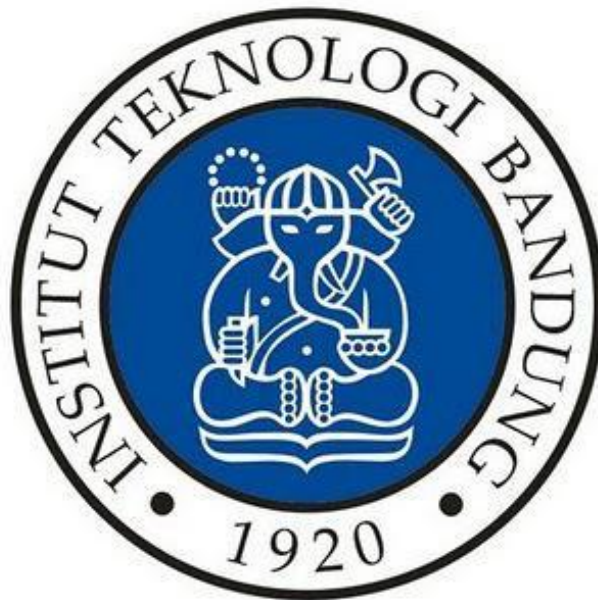


IF4031 Pengembangan Aplikasi Terdistribusi

Terraform Auto Scaling ECS NGINX Cluster

Muhammad Garebaldhie ER Rahman
13520029



PROGRAM STUDI SARJANA TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

Daftar Isi

Daftar Isi	2
How to implement	3
Definitions	3
1. Create cluster	3
2. Task definition	3
3. Security Group	4
4. Target Group	5
5. Load balancer	6
6. Service	8
a. Service	8
b. Network	9
c. Auto Scaling	10
Implementation	11
Terraform tfvars	11
Provider	12
Variables	12
Security Group	14
Networks	15
Load Balancer	16
ECS	17
Auto Scaling - [Bonus]	19
Outputs	21
How to Test	23
Main Feature	23
Auto scale	23
Lesson learned	26
References	27
Lampiran	27

How to implement

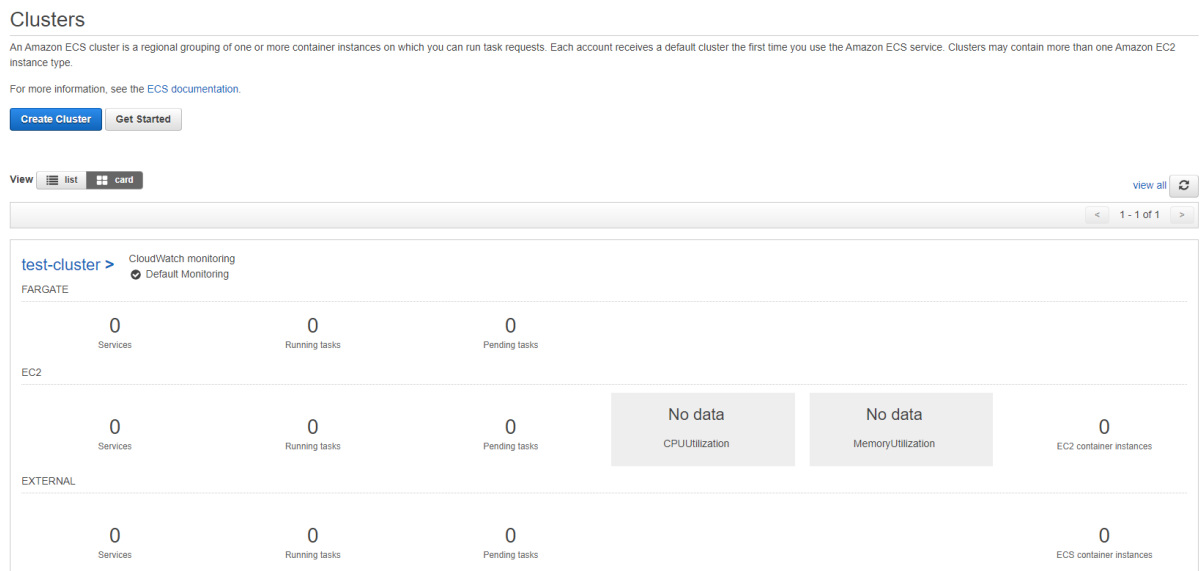
Untuk implementasi dengan terraform kita harus mengerti bagaimana untuk membuat load balancer secara manual. Terdapat beberapa poin utama yang diperlukan untuk membuat ECS Nginx Cluster. Berikut penjelasan singkatnya.

1. **Elastic Container Registry (ECR).** Mirip seperti docker hub yang berisi nginx image
2. **Elastic Container Service (ECS).** Cluster yang akan orchestrasi container serta dapat running service nya
3. **Task Definition.** Task definition merupakan konfigurasi mounting, env, setup per instance serta, container yang bisa di konfigurasi.
4. **Application Load Balancer (ALB).** Load balancer yang akan mengatur dan distribusi traffic yang masuk ke dalam aplikasi
5. **Security Group.** Security group dapat disimplifikasi seperti firewall, dapat di atur inbound serta outbound rule yang diterima pada aplikasi
6. **Target Group.** Target group merupakan target instance yang akan di eksekusi oleh load balancer
7. **Service.** Service dapat mendefinisikan task apa saja yang akan dijalankan, jumlahnya, serta menyediakan recovery mechanism yang akan membuat availability dari aplikasi semakin tinggi

Pertama mencoba untuk implmentasi secara manual yaitu membuat langsung dari aws console nya.

Definitions

1. Create cluster



Dibuat sebuah test-cluster yang masih kosong yang belum memiliki service apa apa.

2. Task definition

- a. Ini buat ngasi kaya mounting, env, intinya setup per instance
- b. Task definition juga memiliki container yang bisa di konfigurasi

- i. Network mode set default yaitu awsvpc
- ii. Task memory yaitu memory yang digunakan per task pada test ini digunakan memory 2048
- iii. Task vCPU yaitu virtual cpu yang digunakan, 1024 berarti 1vCPU
- iv. Operating system family pilih linux

c.

The screenshot shows the AWS ECS console configuration for a new task definition. The configuration is as follows:

- Task definition name:** test-task-definition
- Task role:** None
- Network mode:** awsvpc
- Operating system family:** Linux
- Compatibilities:** EC2, FARGATE
- Requires compatibilities:** FARGATE
- Task execution IAM role:** ecsTaskExecutionRole
- Task size:** 2048 MB memory and 1024 CPU units

3. Security Group

The screenshot shows the AWS Management Console details for a security group named 'sg-03e5cf5e0b1561f1f - testapp-load-balancer-security-group'. The details are as follows:

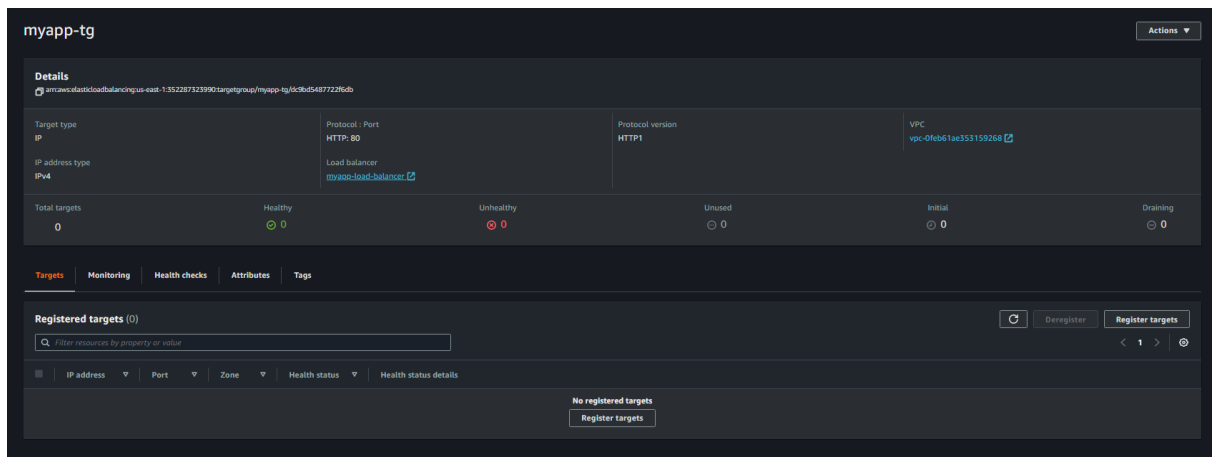
- Security group name:** testapp-load-balancer-security-group
- Security group ID:** sg-03e5cf5e0b1561f1f
- Description:** controls access to the ALB
- VPC ID:** vpc-0f6eb61ae353159268
- Owner:** 352287323990
- Inbound rules count:** 1 Permission entry
- Outbound rules count:** 1 Permission entry

The inbound rules section shows a single rule for HTTP traffic on port 80.

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
-	sg-03e5cf5e0b1561f1f	IPv4	HTTP	TCP	80	0.0.0.0/0	-

Security group dapat dikatakan sebagai firewall serta access control dari aplikasi kita. Dapat dispesifikasikan inbound serta outbound rule dan pada security group ini juga dapat mengallow suatu security group untuk mengakses satu sama lain

4. Target Group



Target group merupakan hal yang akan di proses oleh load balancer, dapat dipilih bermacam macam mulai dari instance, ip address, lambda function ataupun application load balancer. Pada kasus ini pilih **IP ADDRESS** sebagai target agar load balancer dapat distribusi request yang masuk ke berbagai private subnet

Specify group details

Your load balancer routes requests to the targets in a target group and performs health checks on the targets.

Basic configuration

Settings in this section cannot be changed after the target group is created.

Choose a target type

- ☒ **Instances**
 - Supports load balancing to instances within a specific VPC.
 - Facilitates the use of [Amazon EC2 Auto Scaling](#) to manage and scale your EC2 capacity.
- ☐ **IP addresses**
 - Supports load balancing to VPC and on-premises resources.
 - Facilitates routing to multiple IP addresses and network interfaces on the same instance.
 - Offers flexibility with microservice based architectures, simplifying inter-application communication.
 - Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.
- ☐ **Lambda function**
 - Facilitates routing to a single Lambda function.
 - Accessible to Application Load Balancers only.
- ☐ **Application Load Balancer**
 - Offers the flexibility for a Network Load Balancer to accept and route TCP requests within a specific VPC.
 - Facilitates using static IP addresses and PrivateLink with an Application Load Balancer.

Target group name

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

Protocol : Port

HTTP : 80

VPC

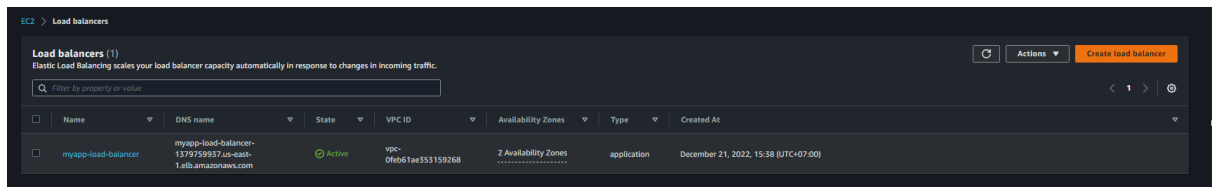
Select the VPC with the instances that you want to include in the target group.

vpc-019b388d107248f1f
IPv4: 172.31.0.0/16

Protocol version

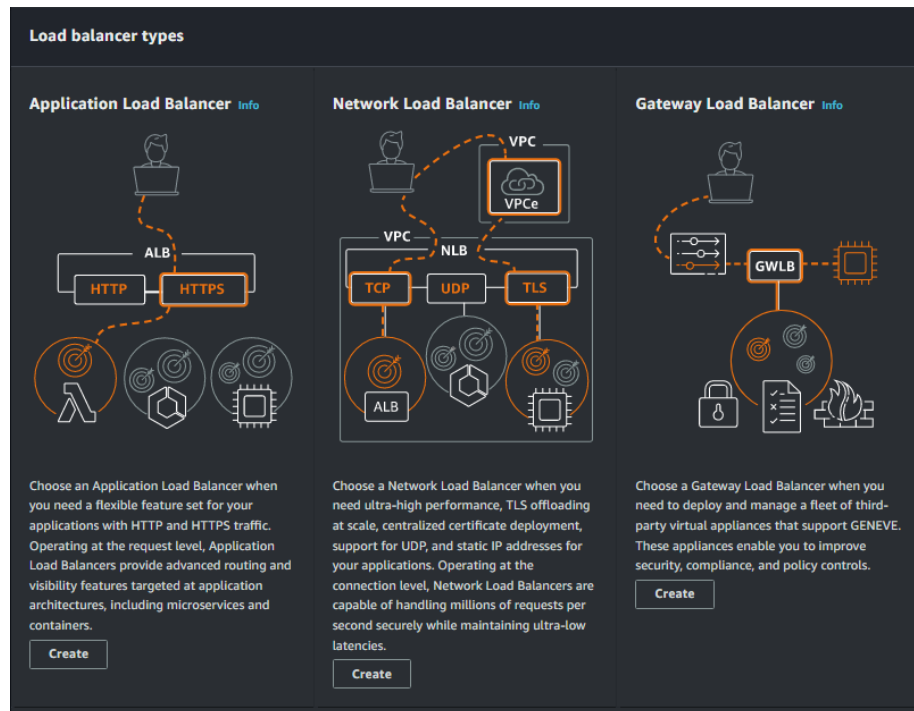
- ☒ **HTTP1**
Send requests to targets using HTTP/1.1. Supported when the request protocol is HTTP/1.1 or HTTP/2.
- ☐ **HTTP2**
Send requests to targets using HTTP/2. Supported when the request protocol is HTTP/2 or gRPC, but gRPC-specific features are not available.
- ☐ **gRPC**
Send requests to targets using gRPC. Supported when the request protocol is gRPC.

5. Load balancer



	Name	DNS name	State	VPC ID	Availability Zones	Type	Created At
<input type="checkbox"/>	myapp-load-balancer	myapp-load-balancer-137975937.as-east-1.elb.amazonaws.com	Active	vpc-0feb61ae353159268	2 Availability Zones	application	December 21, 2022, 15:38 (UTC+07:00)

Terdapat 3 load balancer pada AWS



Namun pada nginx ecs cluster ini akan digunakan application load balancer yang akan routing dan distribusi traffic aplikasi nginx. Pada load balancer juga dapat dipilih availability zones yang tersedia serta listener yang akan listen dan akan memforward ke container yang akan dibuat. Pilih security group dan target group yang telah dibuat pada proses sebelumnya

Network mapping

The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.

VPC

Select the virtual private cloud (VPC) for your targets. Only VPCs with an Internet gateway are enabled for selection. The selected VPC cannot be changed after the load balancer is created. To confirm the VPC for your targets, view your [target group](#).

1

vp-019b3861072248f1f

IPv4: 172.31.0.0/16

Mappings

Select at least two Availability Zones and one subnet per zone. The load balancer routes traffic to targets in these Availability Zones only. Availability Zones that are not supported by the load balancer or the VPC are not available for selection.

☐ us-east-1a (use1-east)

☐ us-east-1b (use1-east)

☐ us-east-1c (use1-east)

☐ us-east-1d (use1-east)

☐ us-east-1e (use1-east)

☐ us-east-1f (use1-east)

Security groups

A security group is a set of firewall rules that control the traffic to your load balancer.

Security groups

Select up to 5 security groups

default sg-02bucc00912a62f

VPC: vp-019b3861072248f1f

Listeners and routing

A listener is a process that checks for connection requests using the port and protocol you configure. The rules that you define for a listener determine how the load balancer routes requests to its registered targets.

Listener HTTP-80

Remove

Protocol

Port

Default action

Info

HTTP

80

Forward to

Select a target group

Create target group

Listener tags - optional

Consider adding tags to your listener. Tags enable you to categorize your AWS resources so you can more easily manage them.

Add listener tag

You can add up to 50 more tags.

Add listener

Jangan lupa untuk configure load balancernya dengan menambahkan inbound rule yaitu adding security group di fargate agar dapat mengallow traffic security group dari load balancernya

Inbound rules

Security group rule ID

sg-057906cabf3330052

Type

Protocol

Port range

Source

Description - optional

HTTP

TCP

80

Custom

0.0.0.0/0

Delete

-

All TCP

TCP

0 - 65535

Custom

sg-0b6159637f5428f1d

Delete

Add rule

Cancel

Preview changes

Save rules

6. Service

a. Service

Configure service

A service lets you specify how many copies of your task definition to run and maintain in a cluster. You can optionally use an Elastic Load Balancing load balancer to distribute incoming traffic to containers in your service. Amazon ECS maintains that number of tasks and coordinates task scheduling with the load balancer. You can also optionally use Service Auto Scaling to adjust the number of tasks in your service.

Launch type

☒ FARGATE

AWS Fargate is migrating service quotas from the current Amazon ECS task count-based quotas to vCPU-based quotas. To learn more, refer to the AWS Fargate FAQs.

☐ EC2

☐ EXTERNAL

[Switch to capacity provider strategy](#)

Operating system family

Linux

Task Definition

Family

test-task-definition

Revision

1 (latest)

Platform version

LATEST

Cluster

test-cluster

Service name

test-service

Service type*

REPLICA

Number of tasks

2

Minimum healthy percent

100

Maximum percent

200

Deployment circuit breaker

Disabled

Enter a value

Deployments

Service yang akan dibuat akan memilih task yang akan dijalankan serta berapa jumlah instance minimal yang akan di buat. Service juga akan menjaga agar availability dari aplikasi tinggi dengan cara langsung spawn instance baru jika terdapat masalah pada salah satu instance

b. Network

Load balancing

An Elastic Load Balancing load balancer distributes incoming traffic across the tasks running in your service. Choose an existing load balancer, or create a new one in the [Amazon EC2 console](#).

Load balancer type*

☐ None

Your service will not use a load balancer.

☒ Application Load Balancer

Allows containers to use dynamic host port mapping (multiple tasks allowed per container instance). Multiple services can use the same listener port on a single load balancer with rule-based routing and paths.

☐ Network Load Balancer

A Network Load Balancer functions at the fourth layer of the Open Systems Interconnection (OSI) model. After the load balancer receives a request, it selects a target from the target group for the default rule using a flow hash routing algorithm.

☐ Classic Load Balancer

Requires static host port mappings (only one task allowed per container instance); rule-based routing and paths are not supported.

Service IAM role

Task definitions that use the awsvpc network mode use the AWSServiceRoleForECS service-linked role, which is created for you automatically. [Learn more](#).

Load balancer name

test-nginx-lb



Container to load balance

test-nginx-container : 80

[Remove](#) ✕

Production listener port*

80:HTTP



Production listener protocol*

HTTP

Target group name

test-nginx-target-group



Target group protocol

HTTP



Target type

ip



Path pattern

/

Evaluation order

default

Health check path

/



Additional health check options can be configured in the ELB console after you create your service.

Network akan menspesifikasikan jenis load balancer yang digunakan serta target group dari load balancer tersebut

c. Auto Scaling

Set Auto Scaling (optional)

Automatically adjust your service's desired count up and down within a specified range in response to CloudWatch alarms. You can modify your Service Auto Scaling configuration at any time to meet the needs of your application.

Service Auto Scaling ☐ Do not adjust the service's desired count
☒ Configure Service Auto Scaling to adjust your service's desired count

Minimum number of tasks ⓘ

Automatic task scaling policies you set cannot reduce the number of tasks below this number.

Desired number of tasks ⓘ

Maximum number of tasks ⓘ

Automatic task scaling policies you set cannot increase the number of tasks above this number.

IAM role for Service Auto Scaling ⓘ

Automatic task scaling policies

Scaling policy type ☒ Target tracking ⓘ
☐ Step scaling

Policy name* ⓘ

EC2 service metric* ⓘ

Target value* ⓘ

Scale-out cooldown period seconds between scaling actions ⓘ

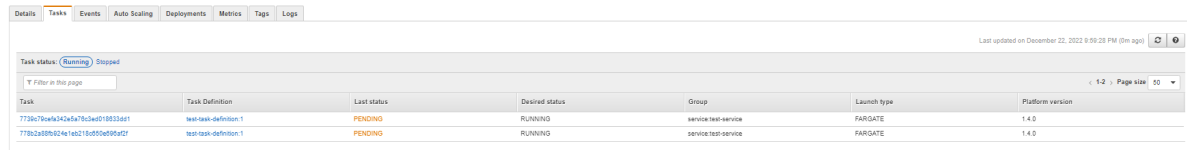
Scale-in cooldown period seconds between scaling actions ⓘ

Disable scale-in ☐ ⓘ

Scaling pada kasus ini merupakan horizontal scaling, yaitu akan membuat instance baru yang sama dengan instance sebelumnya untuk mengurangi load pada instance tersebut. Terdapat tiga jenis scaling. Simple, target dan step. Namun pada kasus ini hanya diberi dua pilhan yaitu target dan step.

Perbedaan utamanya ialah target akan mengkeep angka yang telah ditentukan (metric) dan akan berusaha agar instance memiliki metric yang telah ditentukan. Step scaling berbeda, step scaling akan melakukan scaling ketika metric melewati threshold tertentu. Misal jika cpu utilization melebihi 50%, add satu instance jika 80% add empat instance dan sebagainya. Target scaling akan berusaha membuat instance memiliki average cpu utilization di angka 50 dan jika tidak bisa maka akan di buat instance baru agar metricnya tetap.

Jika sudah selesai maka nanti servicenya akan mendeploy target (container)



Task	Task Definition	Last status	Desired status	Group	Launch type	Platform version
7799c70e9f4c4a78c5ed18833d41	test-task-definition:1	PENDING	RUNNING	service-test-service	FARGATE	1.4.0
7799c70e9f4c4a78c5ed18833d41	test-task-definition:1	PENDING	RUNNING	service-test-service	FARGATE	1.4.0

Implementation

Setelah memahami definisi dan cara pembuatan secara manual maka akan dicoba untuk diimplementasikan dengan terraform. Implementasi dapat diakses di sini <https://github.com/IloveNoodles/Terraform-ECS-Nginx-Cluster>

Berikut merupakan terraform “data type” serta command console yang digunakan

Data type and command

Data type

Input Variables — Serve as parameters for a Terraform module, so users can customize behavior without editing the source

Modules — Acts as a container for multiple resources that are used together. It is a way to package and reuse resource configurations.

Resources — Documents the syntax for declaring resources

Data sources — Allow data to be fetched or computed for use elsewhere in Terraform configuration

Output values — Return values for a Terraform module

Local values — A convenience feature for assigning a short name to an expression

Command

terraform init — Initializes the working directory which consists of all the configuration files

terraform validate — Validates the configuration files in a directory

terraform plan — Creates an execution plan to reach a desired state of the infrastructure

terraform apply — Makes the changes in the infrastructure as defined in the plan

terraform destroy — Deletes all the old infrastructure resources

Terraform tfvars

terraform.tfvars

```
region = "prefered-region"
aws_access_key_id = "YOUR_AWS_ACCESS_KEY"
aws_secret_access_key = "YOUR_AWS_SECRET_KEY"
```

Buat sebuah terraform.tfvars dengan cara copy contoh yang ada pada .example lalu isi dengan access dan secret key yang dimiliki.

Provider

provider.tf

```
# ===== Creating AWS Provider
provider "aws" {
  region = var.region
  access_key = var.aws_access_key_id
  secret_key = var.aws_secret_access_key
}
```

Provider merupakan “third party” platform yang sudah terintegrasi dengan terraform. Pada kasus ini akan dipilih aws sebagai provder. Untuk access key dan secret key dapat dimasukan pada terraform.tfvars

Variables

Variables yang telah dispesifikasikan pada .tfvars perlu di definisikan datanya.

variables.tf

```
variable "aws_access_key_id" {
  description = "Your aws access key id"
  type      = string
}

variable "aws_secret_access_key" {
  description = "Your aws secret access key"
  type      = string
}

variable "region" {
  description = "AWS region"
  default    = "us-east-1"
  type      = string
}

# What image you want to build
# How many container should be spawned
# port
variable "app_type" {
  description = "Application and configuration"
  type = object({
    image = string
    count = number
    port  = number
  })
  default = {
    image = "nginx:latest"
    count = 2
    port  = 80
  }
}
```

```

}

variable "aws_launch_type" {
  description = "ECS Launch type. (1vCPU = 1024, memory in MiB)"
  type = object({
    type  = string
    cpu   = number
    memory = number
  })
  default = {
    type  = "FARGATE"
    cpu   = 256
    memory = 512
  }
}

variable "availability_zones_count" {
  description = "Many instance that will be created"
  type        = number
  default     = 2
}

variable "vpc_cidr_block" {
  description = "CIDR block for your vpc"
  type        = string
  default     = "172.16.0.0/16" # 16 bit hosts, 2^16 which is maximal
}

variable "autoscale_config" {
  description = "Configuration for app autoscaling target"
  type = object({
    min = number
    max = number
  })
  default = {
    min = 1
    max = 4
  }
}

variable "autoscale_metric" {
  description = "Configuration for cloud metric alarm"
  type = object({
    period          = string
    cooldown        = number
    evaluation_periods = string
    max_threshold   = string
    min_threshold   = string
    up              = number
    down            = number
  })
  default = {
    period          = "120"
    cooldown        = 120
  }
}

```

```
    evaluation_periods = "3"
    max_threshold      = "80"
    min_threshold      = "10"
    up                 = 1
    down               = -1
  }
}
```

Security Group

Security.tf

```
# ===== Creating security group for load balancer
resource "aws_security_group" "lb" {
  name = "nginx-load-balancer-security-group"
  description = "Allow http to be accepted and forwarded to 80"
  vpc_id = aws_vpc.main.id

  # inbound rule accept 80:80
  ingress {
    protocol = "tcp"
    description = "Accepting 80 and forward to 80"
    from_port = var.app_type.port
    to_port = var.app_type.port
    cidr_blocks = ["0.0.0.0/0"]
  }

  # outbound rule
  egress {
    protocol = "-1" # set to all
    from_port = 0
    to_port = 0
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# ===== Creating security group for ecs
resource "aws_security_group" "ecs" {
  name = "nginx-ecs-tasks-security-group"
  description = "Limiting access of ecs to get from alb only"
  vpc_id = aws_vpc.main.id

  # inbound rule accept 80:80 for lb only
  ingress {
    protocol = "tcp"
    description = "Accepting 80 and forward to 80"
    from_port = var.app_type.port
    to_port = var.app_type.port
    security_groups = [aws_security_group.lb.id]
  }

  # outbound rule
```

```
egress {  
  protocol = "-1" # set to all  
  from_port = 0  
  to_port = 0  
  cidr_blocks = ["0.0.0.0/0"]  
}  
}
```

Buat dua security groups, yang pertama security group untuk load balancer lalu yang kedua untuk ECS nya. Pastikan ECS mengallow load balancer untuk mengakses dengan cara menambahkan inbound rule pada security groups ECS

Networks

Networks.tf

```
# ===== Get all available zones in current region  
data "aws_availability_zones" "az" {  
  state = "available"  
}  
  
# ===== Creating vpc  
resource "aws_vpc" "main" {  
  cidr_block = var.vpc_cidr_block  
}  
  
# ===== Creating 2 /24 private subnet  
resource "aws_subnet" "private" {  
  vpc_id      = aws_vpc.main.id  
  count       = var.availability_zones_count  
  availability_zone = data.aws_availability_zones.az.names[count.index]  
  cidr_block   = cidrsubnet(aws_vpc.main.cidr_block, 8, count.index)  
}  
  
# ===== Creating 2 /24 public subnet  
# Adding availability_zones_count to differ the cidr numnetwork  
resource "aws_subnet" "public" {  
  vpc_id      = aws_vpc.main.id  
  count       = var.availability_zones_count  
  availability_zone = data.aws_availability_zones.az.names[count.index]  
  cidr_block   = cidrsubnet(aws_vpc.main.cidr_block, 8, var.availability_zones_count +  
count.index)  
  map_public_ip_on_launch = true  
}  
  
# Create gateway to make public accessible  
resource "aws_internet_gateway" "gateway" {  
  vpc_id = aws_vpc.main.id  
}
```

```

# ===== Create gateway for nat

# Elastic IP
resource "aws_eip" "nat" {
  count      = var.availability_zones_count
  depends_on = [aws_internet_gateway.gateway]
  vpc       = true
}

# NAT
# Element needed to map to each private ip
resource "aws_nat_gateway" "gateway" {
  count      = var.availability_zones_count
  subnet_id  = element(aws_subnet.public.*.id, count.index)
  allocation_id = element(aws_eip.nat.*.id, count.index)
}

# ===== Routing table

# Public
resource "aws_route" "internet" {
  route_table_id      = aws_vpc.main.main_route_table_id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id          = aws_internet_gateway.gateway.id
}

# Private
resource "aws_route_table" "private" {
  count = var.availability_zones_count
  vpc_id = aws_vpc.main.id

  route {
    cidr_block      = "0.0.0.0/0"
    nat_gateway_id = element(aws_nat_gateway.gateway.*.id, count.index)
  }
}

# ===== Attaching routing table to the private subnet
resource "aws_route_table_association" "private" {
  count      = var.availability_zones_count
  subnet_id  = element(aws_subnet.private.*.id, count.index)
  route_table_id = element(aws_route_table.private.*.id, count.index)
}

```

Network akan membuat public subnet yang dapat diakses oleh public serta private subnet yang hanya dapat diakses melalui NAT yang telah didefinisikan diatas. Pastikan routing table telah di attached ke private subnet agar dapat diakses. Menurut saya konfigurasi network ini merupakan yang paling sulit



Load Balancer

load-balancer.tf

```
# ===== Creating a load balancer
resource "aws_alb" "lb" {
  name      = "nginx-load-balancer"
  subnets  = aws_subnet.public.*.id
  security_groups = [aws_security_group.lb.id]
}

# ===== Createing a target group
resource "aws_alb_target_group" "target" {
  vpc_id    = aws_vpc.main.id
  name      = "nginx-target-group"
  port      = var.app_type.port
  protocol  = "HTTP"
  target_type = "ip"

  health_check {
    path    = "/"
    matcher = "200"
  }
}

# ===== Redirect incoming traffic to target from lb
resource "aws_alb_listener" "listen" {
  load_balancer_arn = aws_alb.lb.arn
  port              = var.app_type.port

  default_action {
    type = "forward"
    target_group_arn = aws_alb_target_group.target.arn
  }
}
```

Buat load balancer serta target group untuk load balancernya. Jangan lupa untuk membuat listener yang akan memforward traffic ke dalam target group dari public

ECS

ecs.tf

```
# ===== Creating aws ecs cluster
resource "aws_ecs_cluster" "ecs" {
  name = "nginx-cluster"
}

# ===== Creating task definition
data "template_file" "nginx" {
  template = file("../templates/task-definition.json")
}
```

```

vars = {
  app_image = var.app_type.image
  app_cpu   = var.aws_launch_type.cpu
  app_memory = var.aws_launch_type.memory
}

resource "aws_ecs_task_definition" "td" {
  family           = "nginx-task"
  network_mode     = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  cpu              = var.aws_launch_type.cpu
  memory          = var.aws_launch_type.memory
  container_definitions = data.template_file.nginx.rendered
}

# ===== Creating service for the cluster
resource "aws_ecs_service" "service" {
  name           = "nginx-service"
  cluster        = aws_ecs_cluster.ecs.id
  task_definition = aws_ecs_task_definition.td.arn
  desired_count  = var.app_type.count
  launch_type    = var.aws_launch_type.type

  network_configuration {
    security_groups = [aws_security_group.ecs.id]
    subnets        = aws_subnet.private.*.id
    assign_public_ip = true
  }

  load_balancer {
    target_group_arn = aws_alb_target_group.target.id
    container_name   = "nginx"
    container_port    = var.app_type.port
  }

  depends_on = [
    aws_alb_listener.listen,
  ]
}

```

Membuat ECS Cluster dengan load balancer yang telah dibuat. Type dari aplikasi dapat didefinisikan pada `launch_type` pada service pada kasus ini dipilih FARGATE serta dipilih 256MB untuk vCPU serta 512MB untuk memory.

Berikut task definition yang telah dibuat. Task definition akan menspesifikasikan task yang akan di buat mulai dari nama, image yang digunakan, cpu, memory serta port mappingnya.

```
templates/task-definition.json
```

```
[
  {
    "name": "nginx",
    "image": "${app_image}",
    "cpu": ${app_cpu},
    "memory": ${app_memory},
    "networkMode": "awsvpc",
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ]
  }
]
```

Auto Scaling - [Bonus]

Saya membuat dua jenis scaling, scaling dengan menggunakan target dan menggunakan step.

auto-scale.tf ; Target scaling

```
resource "aws_appautoscaling_target" "ecs_target" {
  max_capacity      = var.autoscale_config.max
  min_capacity      = var.autoscale_config.min
  resource_id       = "service/${aws_ecs_cluster.ecs.name}/${aws_ecs_service.service.name}"
  scalable_dimension = "ecs:service:DesiredCount"
  service_namespace = "ecs"
}

# ===== Target Scaling
# This is an example:
app/EC2Co-EcsEl-1TKLTMITMM0EO/f37c06a68c1748aa/targetgroup/EC2Co-Defau-LDNM7Q3
ZH1ZN/6d4ea56ca2d6a18d.
resource "aws_appautoscaling_policy" "target_scaling" {
  name                = "nginx-target-policy"
  policy_type         = "TargetTrackingScaling"
  resource_id         = aws_appautoscaling_target.ecs_target.resource_id
  scalable_dimension  = aws_appautoscaling_target.ecs_target.scalable_dimension
  service_namespace   = aws_appautoscaling_target.ecs_target.service_namespace
  target_tracking_scaling_policy_configuration {
    predefined_metric_specification {
      predefined_metric_type = "ALBRequestCountPerTarget"
      resource_label        =
"app/${aws_alb.lb.name}/${basename("${aws_alb.lb.id}")}/targetgroup/${aws_alb_target_group.t
arget.name}/${basename("${aws_alb_target_group.target.id}")}"
    }
    target_value = 10
  }
}
```

Pada target scaling dipilih target value 10 per request, the scaling will maintain the number of request in about 10 request

auto-scale.tf ; Step Scaling

```
# ===== Step Scaling
# Scale up
resource "aws_appautoscaling_policy" "up_policy" {
  name           = "nginx_scale_up_policy"
  resource_id     = aws_appautoscaling_target.ecs_target.resource_id
  scalable_dimension = aws_appautoscaling_target.ecs_target.scalable_dimension
  service_namespace = aws_appautoscaling_target.ecs_target.service_namespace

  step_scaling_policy_configuration {
    adjustment_type = "ChangeInCapacity"
    cooldown        = var.autoscale_metric.cooldown
    metric_aggregation_type = "Maximum"
    step_adjustment {
      metric_interval_lower_bound = 0
      scaling_adjustment          = var.autoscale_metric.up
    }
  }
}

# Scale down
resource "aws_appautoscaling_policy" "down_policy" {
  name           = "nginx-scale-down-policy"
  resource_id     = aws_appautoscaling_target.ecs_target.resource_id
  scalable_dimension = aws_appautoscaling_target.ecs_target.scalable_dimension
  service_namespace = aws_appautoscaling_target.ecs_target.service_namespace

  step_scaling_policy_configuration {
    adjustment_type = "ChangeInCapacity"
    cooldown        = var.autoscale_metric.cooldown
    metric_aggregation_type = "Maximum"

    step_adjustment {
      metric_interval_upper_bound = 0
      scaling_adjustment          = var.autoscale_metric.down
    }
  }
}

# ===== Define metrics for the cpu
# High
resource "aws_cloudwatch_metric_alarm" "high_cpu_service" {
  alarm_name           = "nginx-cpu-high"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  period              = var.autoscale_metric.period
  evaluation_periods   = var.autoscale_metric.evaluation_periods
  metric_name          = "CPUUtilization"
  namespace            = "AWS/ECS"
}
```

```

statistic      = "Average"
threshold      = var.autoscale_metric.max_threshold

dimensions = {
  ClusterName = aws_ecs_cluster.ecs.name
  ServiceName = aws_ecs_service.service.name
}

alarm_actions = [aws_appautoscaling_policy.up_policy.arn]
}

# Low
resource "aws_cloudwatch_metric_alarm" "low_cpu_service" {
  alarm_name      = "nginx-cpu-down"
  comparison_operator = "LessThanOrEqualToThreshold"
  period          = var.autoscale_metric.period
  evaluation_periods = var.autoscale_metric.evaluation_periods
  metric_name     = "CPUUtilization"
  namespace      = "AWS/ECS"
  statistic       = "Average"
  threshold       = var.autoscale_metric.min_threshold

  dimensions = {
    ClusterName = aws_ecs_cluster.ecs.name
    ServiceName = aws_ecs_service.service.name
  }

  alarm_actions = [aws_appautoscaling_policy.down_policy.arn]
}

```

Step scaling memerlukan cloud watch alarm sebagai pengukur metric, Jika metric yang ditentukan telah melwati threshold maka cloudwatch alarm akan melakukan step scaling berdasarkan step adjusment yang telah di definisikan

Outputs

```

outputs.tf

# ===== Outputing the created dns server
output "alb_hostname" {
  value = aws_alb.lb.dns_name
}

```

Output block merupakan block yang special karena jika telah melakukan terraform apply maka hasil dari hostname akan dikeluarkan pada console.

```
aws_ecs_service.service: Creation complete after 3s
```

```
[id=arn:aws:ecs:us-east-1:352287323990:service/nginx-cluster/nginx-service]
aws_appautoscaling_target.ecs_target: Creating...
aws_appautoscaling_target.ecs_target: Creation complete after 1s
[id=service/nginx-cluster/nginx-service]
aws_appautoscaling_policy.target_scaling: Creating...
aws_appautoscaling_policy.target_scaling: Creation complete after 2s [id=nginx-target-policy]
```

Apply complete! Resources: 25 added, 0 changed, 0 destroyed.

Outputs:

```
alb_hostname = "nginx-load-balancer-1045795057.us-east-1.elb.amazonaws.com"
```

Semua materi saya ambil dari dokumentasi aws ^[10] serta terraform ^[11]. Namun terdapat beberapa masalah pada saat development.

1. IAM User & Root

Awalnya STEI belum memberikan akses pada IAM user sehingga tidak dapat membuat access key dan secret key. Jadi alternatif yang digunakan ialah membuat root user sendiri lalu assign iam user beserta pollicynya.

2. Variables in terraform

Telah membuat file dengan .tfvars namun konfigurasi tetap saja tidak ke load. Setelah membaca dokumentasi ternyata .tfvars yang akan diload pertama ialah `terraform.tfvars` untuk .tfvars yang lain perlu dispesifikasikan dengan menggunakan argument -var-file="tfvars file".

3. Udah ke deploy tapi ga muncul ternyata salah server

Setelah melakukan deployment dan tertulis apply complete, namun instance tak kunjung muncul. Setelah ditelusuri lebih lanjut, server yang saya gunakan merupakan ap-southeast-1 (tokyo) dan saya melakukan deployment pada us-east-1 sehingga hanya perlu mengganti region dan instance pun muncul

4. Aws_subnet

Pada saat pendefinisian network pada network.tf. Routing table , private subnet , public subnet, NAT banyak istilah istilah aneh yang belum familiar dan perlu konfigurasi agar public dapat mengakses private melalui NAT dan internet dapat mengaskes public IP. Masalah ini diselesaikan dengan menonton tutorial di youtube serta membaca dokumentasi yang disediakan oleh aws serta terraform.

5. Builtin terraform function

Masalah ini juga berkaitan dengan aws subnet sebelumnya, perlu mendefinisikan CIDR untuk setiap instance dan ternyata masalah ini dapat diselesaikan dengan builtin function pada terraform yaitu cidrsubnet

cidrsubnet Function

v1.3.x (latest) ▾

cidrsubnet calculates a subnet address within given IP network address prefix.

```
cidrsubnet(prefix, newbits, netnum)
```

Copy

prefix must be given in CIDR notation, as defined in [RFC 4632 section 3.1](#).

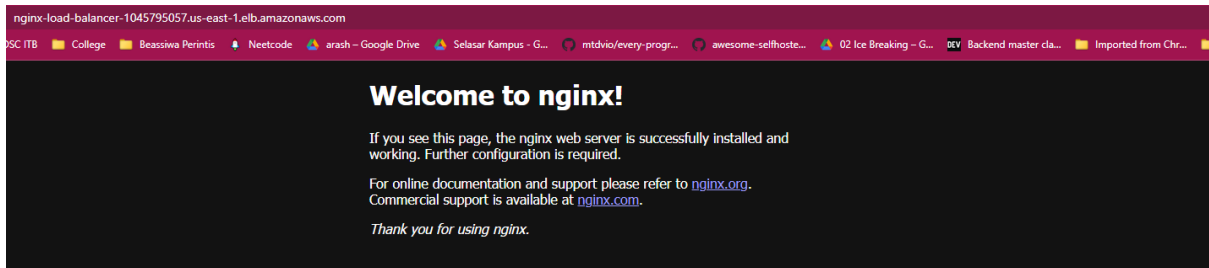
newbits is the number of additional bits with which to extend the prefix. For example, if given a prefix ending in /16 and a newbits value of 4, the resulting subnet address will have length /20.

netnum is a whole number that can be represented as a binary integer with no more than newbits binary digits, which will be used to populate the additional bits added to the prefix.

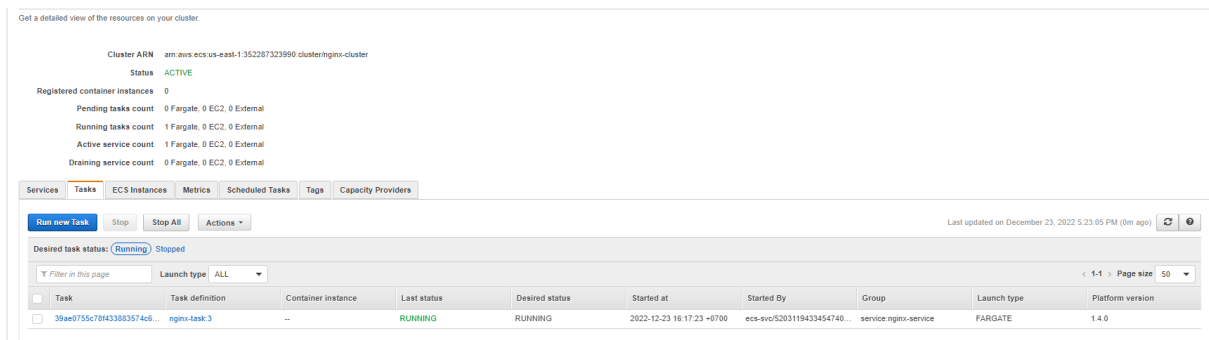
How to Test

Main Feature

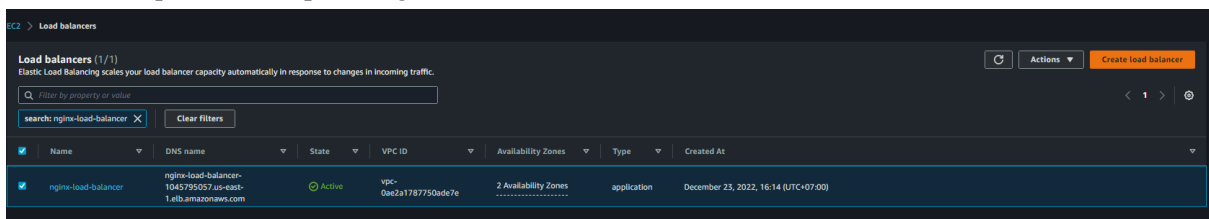
Untuk melakukan testing apakah service sudah berjalan apa belum, hal yang perlu dilakukan ialah membuka dns hostname yang di output oleh terraform. Jika muncul NGINX maka sudah berjalan.



Selain itu dapat di check pada console apakah pada tasks sudah menunjukkan status running apa belum, jika sudah running maka service tersebut sudah berjalan

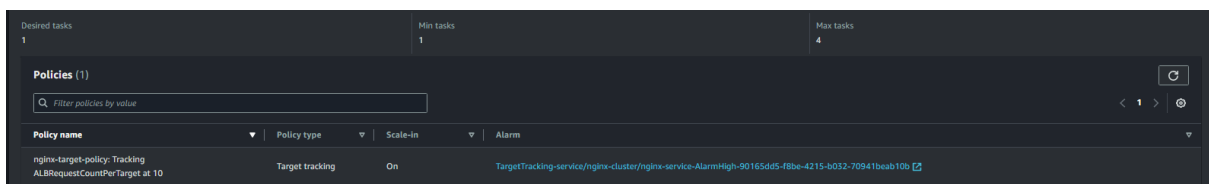


Hostname dapat di check pada bagian EC2 > Load balancer



Auto scale

Auto scale dapat di cek pada cluster configuration. Jika statusnya sudah aktif maka auto scale policy sudah aktif



Selanjutnya dicoba untuk mengakses website tersebut dengan menggunakan artillery sebagai bantuan load testing

```
Terraform-ECS-Cluster on main [!] via default took 25s  
0% > artillery quick --count 20 --num 500 http://nginx-load-balancer-1  
045795057.us-east-1.elb.amazonaws.com/
```

Berikut policy yang menentukan apakah perlu dilakukan auto scale atau tidak dapat dilihat bahwa minimum instance adalah 1 dan maximum instance ialah 4, jadi instance yang di spawn tidak akan melebihi 4

Auto Scaling

Desired tasks

4

Min tasks

1

Max tasks

4

Policies (1)

Filter policies by value

< 1 >

Policy name	Policy type	Scale-in	Alarm
nginx-target-policy: Tracking ALBRequestCountPerTarget at 10	Target tracking	On	TargetTracking-service/nginx-cluster/nginx-service-AlarmHigh-90165dd5-f8be-4215-b032-70941beab10b

Policy name	Status	Last time updated	Configuration	Actions
<input type="checkbox"/> TargetTracking-service/nginx-cluster/nginx-service-AlarmHigh-90165dd5-f8be-4215-b032-70941beab10b	OK	2022-12-23 17:45:18	RequestCountPerTarget > 10 for 3 datapoints within 3 minutes	Actions enabled
<input type="checkbox"/> TargetTracking-service/nginx-cluster/nginx-service-AlarmLow-425fd12b-7fd6-4578-964b-b0f96a1231d4	OK	2022-12-23 17:37:23	RequestCountPerTarget < 9 for 15 datapoints within 15 minutes	Actions enabled

Dapat dilihat bahwa container mulai scale berdasarkan policy yang telah dibuat

Tasks (1/4)								
Filter tasks by property or value								
Task	Last status	Desired st...	Tas...	Revisi...	Health sta...	Started at	Container instan...	
39ae0...	Running	Running	nginx...	3	Unknown	1 hour ago	-	
3c1d8...	Running	Running	nginx...	3	Unknown	1 minute ago	-	
e2ac3c...	Running	Running	nginx...	3	Unknown	57 seconds ago	-	
f385ce...	Running	Running	nginx...	3	Unknown	1 minute ago	-	

Setelah 15 menit dan ternyata traffiknya sudah turun maka policy low akan aktif dan mulai mendeaktivasi ketiga instance yang telah dibuat



Lesson learned

Terdapat beberapa hal yang saya pelajari setelah mengerjakan tugas ini diantaranya

1. Mengerti apa itu ECR
2. Mengerti apa itu ECS
3. Mengerti cara menggunakan terraform
4. Mengerti cara membuat aws_subnet
5. Mengerti provider terraform
6. Mengerti cara load balancer bekerja
7. Mengerti bagaimana cara autoscale bekerja dengan metric yang ditentukan

References

- [1] [Install Terraform](#)
- [2] [Amazon Machine Images \(AMI\) - Amazon Elastic Compute Cloud](#)
- [3] [Terraform 101 | by Shanika Perera | Towards Data Science](#)
- [4] [Step 1: Set Up an AWS Account and Create a User - Amazon Polly](#)
- [5] [Terraform .tfvars files: Variables Management with Examples](#)
- [6] [ECS Fargate Cluster With Terraform | Automate ECS Fargate with Terraform | Tutorial | DevOps](#)
- [7] [How to Setup AWS ECS Fargate with a Load Balancer | Step by Step](#)
- [8] [How to setup terraform module for ECS Fargate](#)
- [9] [AWS ECS Creation with Terraform | JayDemy](#)
- [10] [Docs overview | hashicorp/aws | Terraform Registry](#)
- [11] [Overview - Configuration Language | Terraform | HashiCorp Developer](#)
- [12] [cidrsubnet - Functions - Configuration Language | Terraform | HashiCorp Developer](#)
- [13] [Step and simple scaling policies for Amazon EC2 Auto Scaling](#)
- [14] [Using Amazon CloudWatch alarms](#)
- [15] [How to Choose the Best Way to Scale EC2 Instances When Faced with Changing Demand](#)

Lampiran

Link github: <https://github.com/IloveNoodles/Terraform-ECS-Nginx-Cluster>