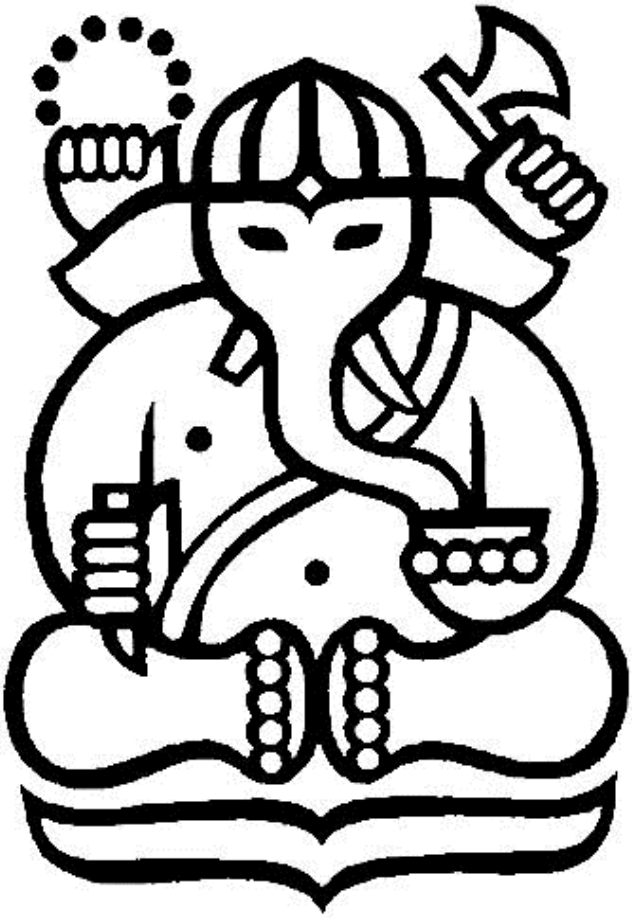
# Tugas Tambahan

# *Provisioning* ECS Fargate dengan Terraform

## IF4031 Pengembangan Aplikasi Terdistribusi




Amar Fadil - 13520103

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

# Table of Contents

# Implementasi

## *Setup Credential* dengan AWS CLI

Untuk menyimpan *credential*, saya menggunakan AWS CLI yang nantinya bisa digunakan oleh terraform. Konfigurasi dilakukan dengan perintah "aws configure". Referensi lebih lanjut dapat dilihat pada link berikut: [Configuration and credential file settings - AWS Command Line Interface (amazon.com)]

```
C:\Users\marfgold1\Documents\IFTugas\Sem 5\PAT\terraform>aws configure
AWS Access Key ID [****************3CC4]:
AWS Secret Access Key [****************rr1f]:
Default region name [ap-southeast-1]:
Default output format [json]:
```

## Versioning dan Inisialisasi

Sebelum melakukan pembuatan resource, terlebih dahulu akan diatur versi provider aws beserta konfigurasi lain untuk inisialisasi terraform. Terdapat file "versions.tf" yang berisikan *required providers* yaitu aws (hashicorp/aws:4.48.0 saat dokumen ini dibuat), kemudian *block provider* aws untuk set konfigurasi spesifik aws (disini diatur region beserta nama *default tags* yang dipakai). Referensi lebih lanjut dapat dilihat pada link berikut: [Docs overview | hashicorp/aws | Terraform Registry]

```
terraform {
    required_providers {
        aws = {
            source  = "hashicorp/aws"
            version = "~> 4.0"
        }
    }
}

provider "aws" {
    region = "ap-southeast-1"

    default_tags {
        tags = {
            Name = "terraform-pat-demo"
        }
    }
}
```

Setelah melakukan konfigurasi, akan dilakukan inisialisasi terraform dengan perintah "terraform init". Terraform kemudian akan melakukan inisialisasi dan meng-install dependency yang dibutuhkan yakni hashicorp/aws. Setelah inisialisasi, akan ada folder .terraform berisi *binary dependency* yang dipilih beserta file .terraform.lock.hcl berisikan lock file provider yang digunakan. Masukkan file tersebut dalam *version control* (tidak perlu di-ignore) sehingga terraform dapat melakukan pemilihan versi default yang sama setiap kali inisialisasi.

```
C:\Users\marfgold1\Documents\IFTugas\Sem 5\PAT\terraform>terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 4.0"...
- Installing hashicorp/aws v4.48.0...
- Installed hashicorp/aws v4.48.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

## *Variables*

Supaya *fine-tuning* lebih mudah dilakukan, akan digunakan *variable* app yang menyimpan semua argument yang bisa di-*tuning* nantinya. Penyimpanan *variable* ini diletakkan pada file “variables.tf” sebagai berikut:

```
variable "app" {
    type = object({
        port = number
        internal_port = number
        container_port = number
        name = string
        cpu = number
        memory = number
        request_per_target = number
        minCount = number
        maxCount = number
    })
    default = {
        port = 80
        internal_port = 80
        container_port = 80
        name = "app"
        cpu = 256
        memory = 1024
        request_per_target = 10
        minCount = 1
        maxCount = 3
    }
```

```
}
```

## Data *Availability Zone*

Untuk definisi *resource* nantinya membutuhkan *availability zone* yang tersedia pada region yang dipilih. Hal ini untuk memastikan Fargate menyediakan *high availability* dengan menyebar task yang sama ke beberapa region prorata. File "data.tf" dibuat untuk menyediakan data ini. Data ini kemudian dapat diakses dengan prefix "data.aws_availability_zones.available". Referensi: aws_availability_zones | Data Sources | hashicorp/aws | Terraform Registry

```
data "aws_availability_zones" "available" {
    state = "available"
}
```

## VPC dan Subnet

Sebelum melakukan konfigurasi jaringan, terlebih dahulu dibuat resource untuk VPC beserta subnet sehingga *container* berada pada jaringan yang sama. Jaringan VPC akan berada pada CIDR block 10.32.0.0/16. Referensi VPC: aws_vpc | Resources | hashicorp/aws | Terraform Registry. File "vpc.tf" menyediakan resource vpc ini.

```
resource "aws_vpc" "default" {
    cidr_block = "10.32.0.0/16"
}
```

Kemudian, VPC tersebut dibagi menjadi 4 subnet, 2 subnet untuk public dan 2 subnet untuk private. Masing-masing subnet memiliki jumlah 2, dengan cidr blocknya diletakkan pada subnet vpc yang telah dibuat dengan alokasi panjang /24 (254 usable host) secara terurut, dimana private akan mendapatkan CIDR block 10.32.0.0/24 dan 10.32.1.0/24 serta public akan mendapatkan CIDR block 10.32.2.0/24 dan 10.32.3.0/24. Alokasi tersebut menggunakan fungsi cidrsubnet yang disediakan oleh terraform. Selain itu, masing-masing subnet akan menggunakan *availability zone* yang tersedia secara *round-robin* dan menggunakan vpc yang telah dibuat sebelumnya. Untuk subnet public, ketika launch resource, public IP akan langsung di-*attach* ke resource tersebut dengan "map_public_ip_in_launch = true". Referensi lebih lanjut dapat dilihat pada link berikut: aws_subnet | Resources | hashicorp/aws | Terraform Registry. Akan ditambahkan *resource* subnet tersebut pada file "vpc.tf" dengan definisi sebagai berikut:

```
resource "aws_subnet" "public" {
    count = 2
    cidr_block = cidrsubnet(aws_vpc.default.cidr_block, 8, 2 + count.index)
    availability_zone =
data.aws_availability_zones.available.names[count.index]
    vpc_id = aws_vpc.default.id
    map_public_ip_on_launch = true
}

resource "aws_subnet" "private" {
    count = 2
    cidr_block = cidrsubnet(aws_vpc.default.cidr_block, 8, count.index)
    availability_zone =
data.aws_availability_zones.available.names[count.index]
    vpc_id = aws_vpc.default.id
}
```

## *Networking*

Untuk menjalankan jaringan, tentunya akan dibuat *resources* yang dibutuhkan untuk menjalankan fungsi *networking*. Pertama, akan ditambahkan VPC internet gateway dari VPC yang telah dibuat sebelumnya. Referensi: aws_internet_gateway | Resources | hashicorp/aws | Terraform Registry. File "network.tf" akan memuat semua definisi resource ini. Definisi VPC internet gateway sebagai berikut:

```
resource "aws_internet_gateway" "gateway" {
    vpc_id = aws_vpc.default.id
}
```

Kedua, akan dibuat rute ke internet (destinasi 0.0.0.0/0) yang menyambungkan *internet gateway* dengan routing table dari VPC yang telah dibuat sebelumnya sebagai berikut: [Referensi: aws_route | Resources | hashicorp/aws | Terraform Registry]

```
resource "aws_route" "internet" {
    route_table_id = aws_vpc.default.main_route_table_id
    destination_cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.gateway.id
}
```

Ketiga, akan dibuat dua buah *resource* Elastic IP yang digunakan untuk allocation id dari NAT gateway pada tahap keempat sebagai berikut: [Referensi: aws_eip | Resources | hashicorp/aws | Terraform Registry]

```
resource "aws_eip" "gateway" {
    count = 2
    vpc = true
    depends_on = [aws_internet_gateway.gateway]
}
```

Keempat, dibuat dua buah public NAT gateway yang menghubungkan dua buah *resource* subnet public yang telah dibuat sebelumnya ke EIP yang telah dibuat sehingga dapat diakses oleh internet. Referensi: aws_nat_gateway | Resources | hashicorp/aws | Terraform Registry

```
resource "aws_nat_gateway" "gateway" {
    count = 2
    subnet_id = element(aws_subnet.public.*.id, count.index)
    allocation_id = element(aws_eip.gateway.*.id, count.index)
}
```

Kelima, akan dibuat dua buah *routing table* yang digunakan untuk *subnet private* pada step selanjutnya sehingga dapat diakses oleh NAT. Referensi: aws_route_table | Resources | hashicorp/aws | Terraform Registry

```
resource "aws_route_table" "private" {
    count = 2
    vpc_id = aws_vpc.default.id

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

```
    }
}
```

Keenam dan terakhir, untuk menghubungkan *routing table* yang telah dibuat ke *subnet private*, dibuatlah dua buah *resource* aws_route_table_association sebagai berikut: [Referensi: aws_route_table_association | Resources | hashicorp/aws | Terraform Registry]

```
resource "aws_route_table_association" "private" {
    count = 2
    subnet_id = element(aws_subnet.private.*.id, count.index)
    route_table_id = element(aws_route_table.private.*.id, count.index)
}
```

## Firewall

*Network* tersebut perlu diberikan *firewall* yang sangat restriktif, sehingga perlu *fine-tuning* dengan menambahkan *security group*. Referensi definisi *security group* dapat dilihat pada link berikut: aws_security_group | Resources | hashicorp/aws | Terraform Registry. Semua definisi firewall berada pada file "firewalls.tf". Pertama, akan ditambahkan *resource firewall* untuk *load balancer* yang memberikan izin *inbound connection* dengan protokol tcp dari port app (80) ke port app (80) untuk *address* apapun (CIDR block 0.0.0.0/0), kemudian *outbound connection* dengan protokol, port dan *address* apapun. *Resource* tersebut akan mengekspos port 80 untuk protokol TCP ke internet, sehingga internet dapat mengakses *load balancer* dan *load balancer* dapat mengakses apapun (allow all), sebagai berikut:

```
resource "aws_security_group" "lb" {
    name = "pat-lb-fw"
    vpc_id = aws_vpc.default.id

    ingress {
        protocol = "tcp"
        from_port = var.app.port
        to_port = var.app.port
        cidr_blocks = ["0.0.0.0/0"]
    }

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

Kedua dan terakhir, akan ditambahkan *resource firewall* untuk aplikasinya itu sendiri yang memberikan izin *inbound connection* dengan protokol tcp dari port internal (3000) dengan IP *load balancer* ke port internal (3000) aplikasi, kemudian *outbound connection* dengan protokol, port dan *address* apapun. *Resource* tersebut akan mengekspos port internal untuk protokol TCP ke *load balancer*, sehingga internet tidak dapat mengakses aplikasi secara langsung, hanya dapat mengakses *load balancer* dan *load balancer* yang kemudian mengakses internal aplikasi, kemudian aplikasi dapat mengakses apapun (allow all), sebagai berikut:

```
resource “aws_security_group” “app” {
    name = “pat-app-fw”
    vpc_id = aws_vpc.default.id

    ingress {
        protocol = “tcp”
        from_port = var.app.internal_port
        to_port = var.app.internal_port
        security_groups = [ aws_security_group.lb.id ]
    }

    egress {
        protocol = “-1”
        from_port = 0
        to_port = 0
        cidr_blocks = [ “0.0.0.0/0” ]
    }
}
```

## Load Balancer

Untuk melakukan *load balancing* ke beberapa *container* yang akan dibuat, tentunya akan dibuat *resource* AWS Elastic Load Balancer. Semua definisi diletakkan pada file “load_balancer.tf”. Pertama, akan dibuat *resource load balancer* itu sendiri terlebih dahulu dengan nama “pat-lb” yang menggunakan *subnet-subnet public* dan *firewall* untuk *load balancer* yang telah dibuat sebelumnya dengan definisi sebagai berikut: [Referensi: aws_lb | Resources | hashicorp/aws | Terraform Registry]

```
resource “aws_lb” “default” {
    name = “pat-lb”
    subnets = aws_subnet.public.*.id
    security_groups = [ aws_security_group.lb.id ]
}
```

Kedua, akan dibuat *resource load balancer target group* dengan nama “pat-lb-target-group” yang membuat grup yang dapat di-*serve* oleh *load balancer* dengan tipe *ip* pada port app (80) dengan protokol HTTP di jaringan VPC default yang telah dibuat sebelumnya sebagai berikut: [Referensi: aws_lb_target_group | Resources | hashicorp/aws | Terraform Registry]

```
resource “aws_lb_target_group” “app” {
    name = “pat-lb-target-group”
    port = var.app.port
    protocol = “HTTP”
    vpc_id = aws_vpc.default.id
    target_type = “ip”
}
```

Ketiga dan terakhir, akan dibuat *resource load balancer listener* yang akan mendengarkan *request* HTTP pada port app (80) dengan *load balancer* yang telah dibuat sebelumnya, kemudian melakukan aksi berupa *forward request* ke *target group* app.

```
Resource “aws_lb_listener” “app” {
    load_balancer_arn = aws_lb.default.id
```

```
    port = var.app.port
    protocol = “HTTP”

    default_action {
        target_group_arn = aws_lb_target_group.app.id
        type = “forward”
    }
}
```

## ECS Service

Terakhir, akan didefinisikan *resource* untuk *service* itu sendiri menggunakan AWS Elastic Cloud Service. Pertama, definisikan terlebih dahulu *task definition* dari *service* menggunakan *docker image* (telah dibuat sebelumnya menggunakan nginx, sumber dapat dilihat disini, original buatan saya) yang membutuhkan Fargate dengan alokasi CPU dan Memory sesuai *variable* app beserta *network mode* menggunakan AWS VPC sebagai berikut: [Referensi: aws_ecs_task_definition | Resources | hashicorp/aws | Terraform Registry]

```
resource "aws_ecs_task_definition" "app" {
    family = var.app.name
    network_mode = "awsvpc"
    requires_compatibilities = ["FARGATE"]
    cpu = var.app.cpu
    memory = var.app.memory
    container_definitions = jsonencode([
        {
            image = "ghcr.io/marfgold1/arcwawan:latest"
            cpu = var.app.cpu
            memory = var.app.memory
            name = var.app.name
            networkMode = "awsvpc"
            portMappings = [
                {
                    containerPort = var.app.container_port,
                    hostPort = var.app.internal_port
                }
            ]
        }
    ])
}
```

Kedua, buat *cluster* baru yang akan digunakan untuk *service* dengan nama “pat-cluster” sebagai berikut: [Referensi: aws_ecs_cluster | Resources | hashicorp/aws | Terraform Registry]

```
resource "aws_ecs_cluster" "app" {
    name = "pat-cluster"
}
```

Ketiga dan terakhir, buat *resource* untuk *service*-nya itu sendiri dengan *launch type* Fargate, *task definition* dan *cluster* yang dipakai telah dibuat sebelumnya, beserta konfiurasi jaringan menggunakan subnet privat dan firewall untuk app, kemudian konfigurasi *load balancer* yang menggunakan *load*

*balancer* yang telah dibuat sebelumnya, sebagai berikut: [Referensi: aws_ecs_service | Resources | hashicorp/aws | Terraform Registry]

```
resource "aws_ecs_service" "app" {
    name = "pat"
    cluster = aws_ecs_cluster.app.id
    task_definition = aws_ecs_task_definition.app.arn
    launch_type = "FARGATE"

    network_configuration {
        security_groups = [ aws_security_group.app.id ]
        subnets = aws_subnet.private.*.id
    }

    load_balancer {
        target_group_arn = aws_lb_target_group.app.id
        container_name = var.app.name
        container_port = var.app.internal_port
    }

    depends_on = [ aws_lb_listener.app ]
}
```

## Output IP untuk Testing

Terakhir, supaya dapat melakukan testing, ditambahkan output app_ip pada file "outputs.txt" yang menampilkan URL dari aplikasi (diakses melalui *load balancer*) sehingga dapat dilihat langsung apakah sudah benar atau belum. Referensi: Output Values - Configuration Language | Terraform | HashiCorp Developer

```
output "app_ip" {
    value = aws_lb.default.dns_name
}
```

## *Plan* dan *Apply*

Untuk melihat konfigurasi yang telah dibuat, dapat dilakukan perintah "terraform plan -out=tfplan" yang menampilkan plan konfigurasi dan disimpan dalam tfplan. Berikut merupakan hasil plan (telah dipotong supaya tidak memenuhi dokumen):

```
C:\Users\marfgold1\Documents\IFTugas\Sem 5\PAT\terraform>terraform plan -out=tfplan
data.aws_availability_zones.available: Reading...
data.aws_availability_zones.available: Read complete after 0s [id=ap-southeast-1]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_ecs_cluster.app will be created
  + resource "aws_ecs_cluster" "app" {
      + arn                = (known after apply)
      + capacity_providers = (known after apply)
      + id                 = (known after apply)
      + name               = "pat-cluster"
      + tags_all           = {
          + "Name" = "pat-terraform-demo"
        }

      + default_capacity_provider_strategy {
          + base             = (known after apply)
          + capacity_provider = (known after apply)
          + weight           = (known after apply)
        }

      + setting {
          + name  = (known after apply)
          + value = (known after apply)
        }
    }

  # aws_ecs_service.app will be created
```

```
Plan: 23 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + app_ip = (known after apply)

Saved the plan to: tfplan

To perform exactly these actions, run the following command to apply:
    terraform apply "tfplan"
```

Setelah puas dengan hasil plan, maka jalankan perintah “terraform apply tfplan” untuk deploy infrastuktur ke AWS sesuai plan yang telah dibuat. Berikut hasil *apply* plan terraform yang telah dibuat (dipotong supaya tidak memenuhi dokumen)

```
C:\Users\marfgold1\Documents\IFTugas\Sem 5\PAT\terraform>terraform apply tfplan
aws_ecs_cluster.app: Creating...
aws_vpc.default: Creating...
aws_ecs_task_definition.app: Creating...
aws_ecs_task_definition.app: Creation complete after 0s [id=app]
aws_vpc.default: Creation complete after 2s [id=vpc-0448b2aff39fb83d7]
aws_internet_gateway.gateway: Creating...
aws_subnet.public[1]: Creating...
```

```
aws_lb.default: Still creating... [3m10s elapsed]
aws_lb.default: Creation complete after 3m14s [id=arn:aws:elasticloadbalancing:ap-northeast-1:352287323990:loadbalancer/app/pat-lb/1963a7633bc888f5]
aws_lb_listener.app: Creating...
aws_lb_listener.app: Creation complete after 0s [id=arn:aws:elasticloadbalancing:ap-northeast-1:352287323990:listener/app/pat-lb/1963a7633bc888f5/c89355c855ddb464]
aws_ecs_service.app: Creating...
aws_ecs_service.app: Creation complete after 2s [id=arn:aws:ecs:ap-northeast-1:352287323990:service/pat-cluster/app]

Apply complete! Resources: 23 added, 0 changed, 0 destroyed.

Outputs:

app_ip = "pat-lb-509681441.ap-northeast-1.elb.amazonaws.com"
```

## Kesulitan

Selama pengerjaan tugas, terdapat beberapa kesulitan, seperti akun AWS yang awalnya tidak bisa dipakai sehingga harus mencoba dari *free trial* akun sendiri (hal ini baru diperbaiki beberapa hari sebelum deadline), kemudian sulitnya konfigurasi jaringan dengan VPC (*trial and error*) serta perlu banyak mencari referensi, terutama dokumentasi untuk provider AWS itu sendiri di terraform. Selain itu, karena menggunakan provider AWS langsung, maka akan terkena *lock-in* provider, sehingga mneyulitkan jika ingin berpindah provider. Ada baiknya sebagai contoh menggunakan Kubernetes yang bisa menyesuaikan ke beberapa provider dalam satu konfigurasi yang sama.

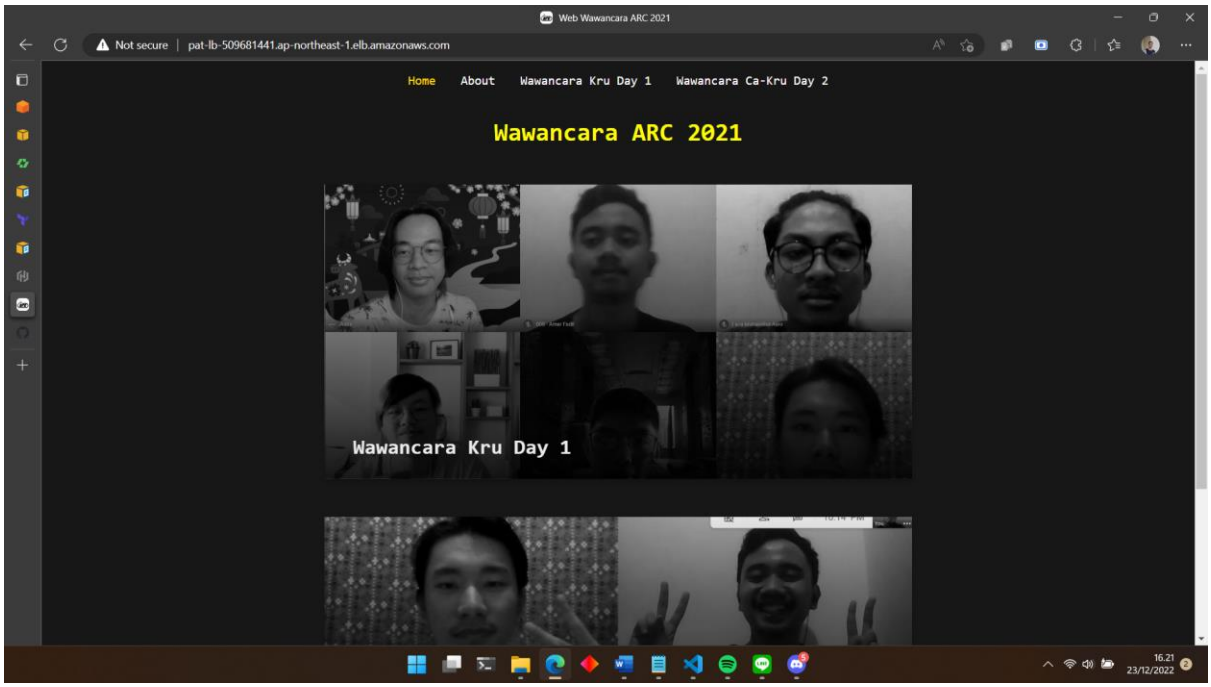
Overall, keberjalanan tugas lumayan lancar. Pada awal apply ada masalah yakni *service* tidak dijalankan. Masalah tersebut ternyata berasal dari *desired count* service yang tidak diatur, sehingga tidak ada task yang dijalankan. Solusinya dengan membuat *desired count* menjadi 1 (minCount).

## *Testing*

Ketika selesai menjalankan apply terraform, maka akan muncul output app_ip yang menunjukkan URL dari *load balancer*. URL tersebut dapat diakses dan diverifikasi apakah sudah dikonfigurasi dengan benar. Dari hasil apply sebelumnya, didapatkan URL *load balancer* sebagai berikut:

```
Apply complete! Resources: 23 added, 0 changed, 0 destroyed.

Outputs:

app_ip = "pat-lb-509681441.ap-northeast-1.elb.amazonaws.com"
```

Ketika mengakses app_ip, ternyata webnya muncul dengan benar.




Hal ini memastikan bahwa *load balancer*, jaringan dan *service* berjalan lancar.

## Pelajaran yang diambil

Terraform dapat mengotomasi pekerjaan infrastuktur dalam awan menjadi lebih mudah. Selain itu, definisi task dapat dibuat lebih modular dan *programmable*. Kemudian, fitur-fitur yang ditawarkan

seperti *state* yang dapat *tracking* status infrastruktur sekarang, sehingga memungkinkan untuk *apply* atau *destroy* instance secara otomatis tanpa perlu manual menghapus satu-satu *resource* tersebut yang tentunya menghabiskan waktu lebih sedikit hanya dengan satu perintah saja. Kemudian, *terraform* juga memberikan kemudahan berupa *file separation* dan pembuatan *module* (belum sempat digunakan) sehingga *file* dapat dipisah-pisah menjadi beberapa bagian.

Secara umum, untuk membuat infrastruktur menggunakan AWS ECS dengan *launch type* Fargate, digunakan *credential* terlebih dahulu dengan AWS CLI. Kemudian, kita dapat membuat inisialisasi dan *version* vendor provider yang ingin digunakan beserta konfigurasinya. Lalu, terraform juga memberikan *datasource* yang bisa digunakan salah satunya adalah *availability zones* sehingga jaringan VPC yang akan digunakan bisa tersebar dan pemanfaatan ECS jauh lebih baik karena *high availability*. Selanjutnya, kita dapat membuat VPC beserta Subnet yang diperlukan untuk jaringan. Kemudian, jaringan dapat dibuat memanfaatkan *routing*, NAT dan Internet *gateway*, serta EIP. Lalu, untuk masalah keamanan, diperlukan *firewall* memanfaatkan *security group*. Setelah itu, kita bisa menambahkan *load balancer* supaya *request* dapat di-*serve* oleh beberapa *service* yang berjalan. Terakhir, kita bisa menambahkan ECS dengan mendefinisikan *task* serta membuat *cluster* dan *service* yang akan digunakan, memanfaatkan *network, firewall,* dan *load balancer* yang telah dibuat sebelumnya. Untuk testing, kita bisa mengakses DNS *load balancer* yang diberikan pada akhir *apply* dan melihat apakah web berjalan lancer atau memiliki masalah.

## Bonus: *Autoscaling*

### Implementasi

Untuk melakukan *autoscaling*, dapat memanfaatkan fitur Application Autoscaling. Pertama, definisikan target aplikasi yang akan di-*autoscale* dengan definisi berikut: [Referensi: aws_appautoscaling_target | Resources | hashicorp/aws | Terraform Registry]

```
resource "aws_appautoscaling_target" "app" {
    min_capacity = var.app.minCount
    max_capacity = var.app.maxCount
    resource_id =
"service/${aws_ecs_cluster.app.name}/${aws_ecs_service.app.name}"
    scalable_dimension = "ecs:service:DesiredCount"
    service_namespace = "ecs"
}
```

Selanjutnya, definisikan *policy autoscaling* yang akan digunakan sebagai berikut: [Referensi: aws_appautoscaling_policy | Resources | hashicorp/aws | Terraform Registry untuk *syntax reference* pada terraform dan RegisterScalableTarget - Application Auto Scaling (amazon.com) untuk penjelasan lebih lengkap beserta PredefinedScalingMetricSpecification - AWS Auto Scaling (amazon.com) untuk penjelasan metrik yang digunakan]

```
resource "aws_appautoscaling_policy" "app" {
    name = "pat-autoscale-target"
    policy_type = "TargetTrackingScaling"
    resource_id = aws_appautoscaling_target.app.resource_id
    scalable_dimension = aws_appautoscaling_target.app.scalable_dimension
    service_namespace = aws_appautoscaling_target.app.service_namespace

    target_tracking_scaling_policy_configuration {
```

```
        predefined_metric_specification {
            predefined_metric_type = "ALBRequestCountPerTarget"
            resource_label =
"app/${aws_lb.default.name}/${basename("${aws_lb.default.id}")}/targetgroup/${
aws_lb_target_group.app.name}/${basename("${aws_lb_target_group.app.id}")}"
        }

        target_value = var.app.request_per_target
        scale_in_cooldown = 300
        scale_out_cooldown = 300
    }
}
```

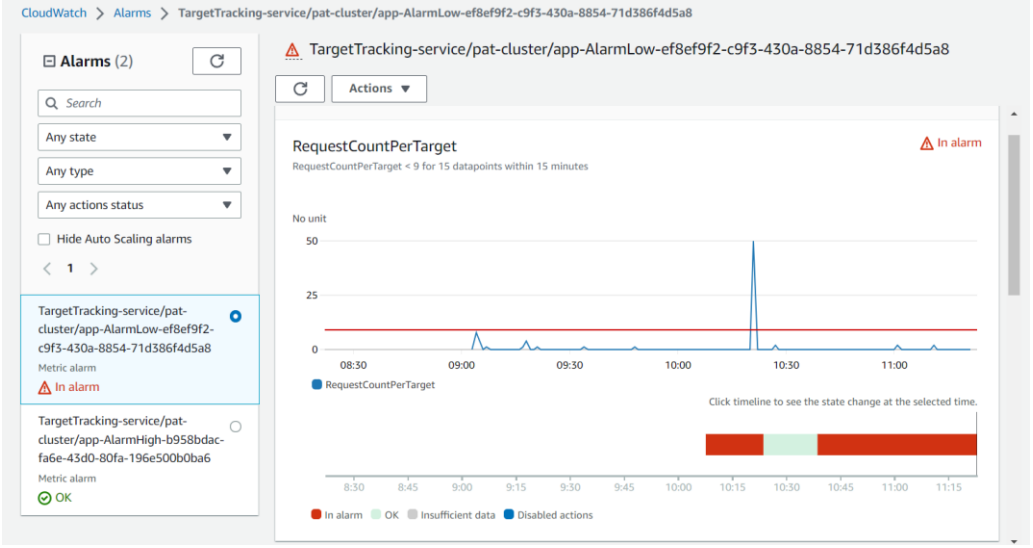
Dapat dilihat bahwa telah didefinisikan *autoscaling policy* dengan nama "pat-autoscale-target" dengan tipe *policy* berupa *target tracking scaling*. Metrik yang digunakan adalah Request Count per Target, dimana target yang diberikan adalah 10 request/target (var.app.request_per_target) dengan *cooldown* untuk *scaling in* dan *out* masing-masing sebesar 300 detik (5 menit). Metrik ini dicatat berdasarkan *request* dari *target group* app dari *load balancer* yang telah didefinisikan sebelumnya.

Berikut hasil dari apply *load balance*:

```
C:\Users\marfgold1\Documents\IFTugas\Sem 5\PAT\terraform>terraform apply tfplan
aws_appautoscaling_target.app: Creating...
aws_appautoscaling_target.app: Creation complete after 0s [id=service/pat-cluster/app]
aws_appautoscaling_policy.app: Creating...
aws_appautoscaling_policy.app: Creation complete after 1s [id=pat-autoscale-target]

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

Outputs:

app_ip = "pat-lb-509681441.ap-northeast-1.elb.amazonaws.com"
```

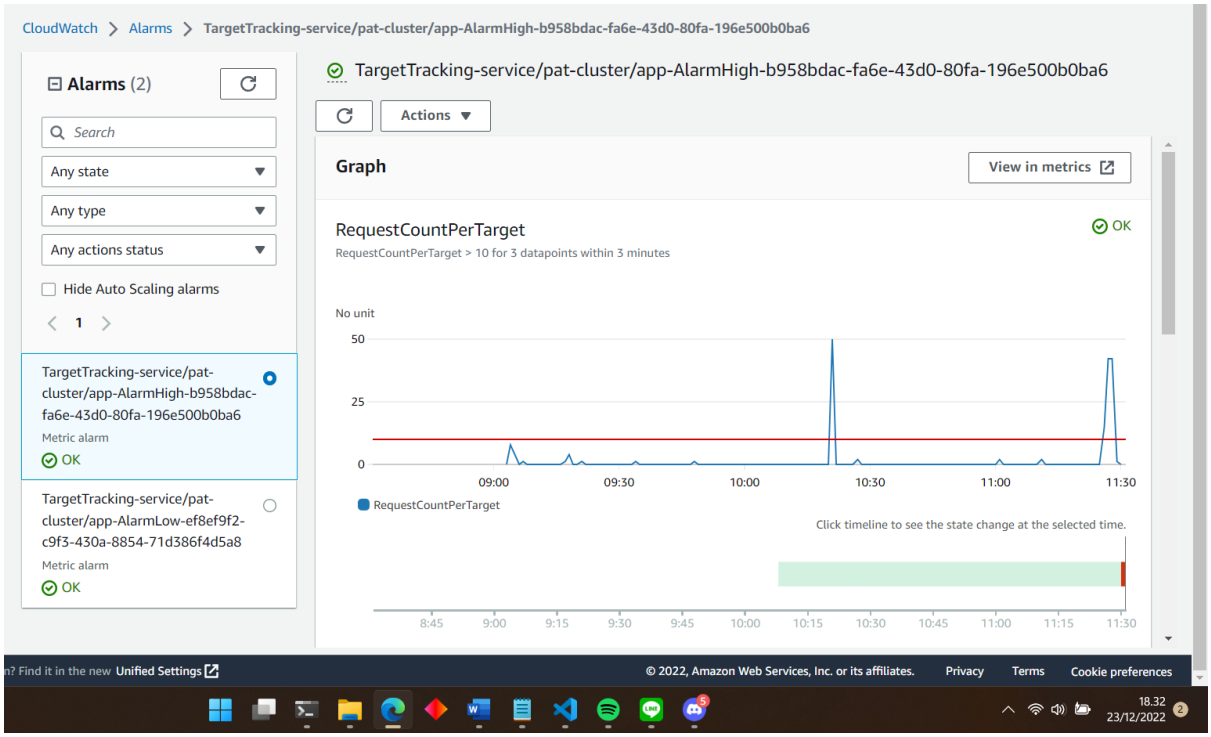
## Testing

Menggunakan CloudWatch, kita dapat memonitor perkembangan *cluster* sekarang, yakni masih *in alarm* karena < 10 request/target.

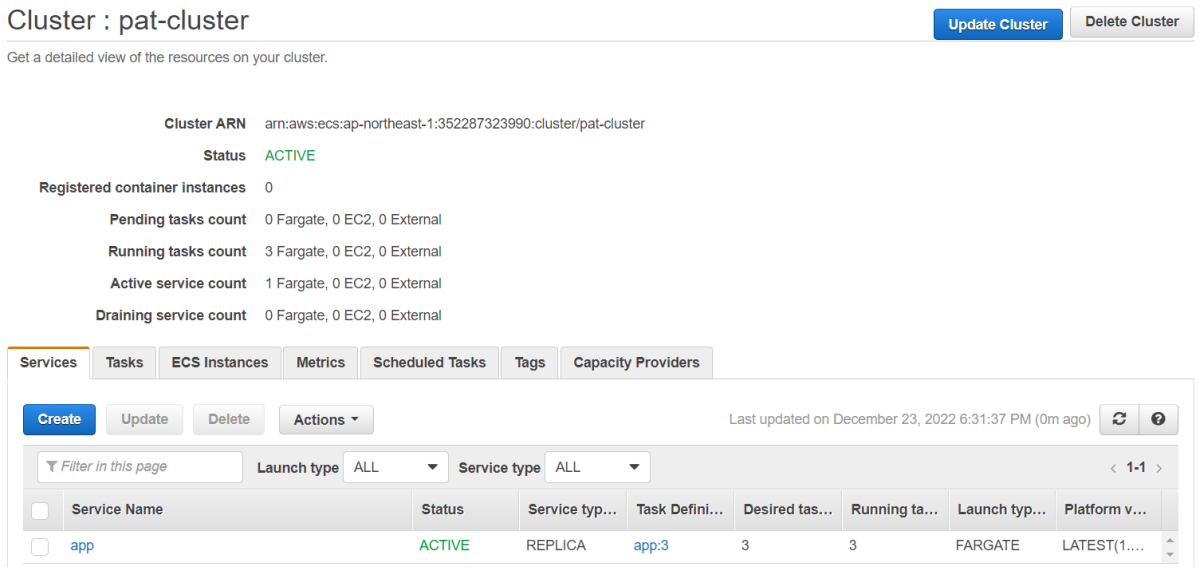


Kemudian, menggunakan ApacheBench, akan dilakukan *load testing* dengan 100 *request*, 1 *request* secara bersamaan, pada DNS *load balancer* yang didapatkan sebelumnya.

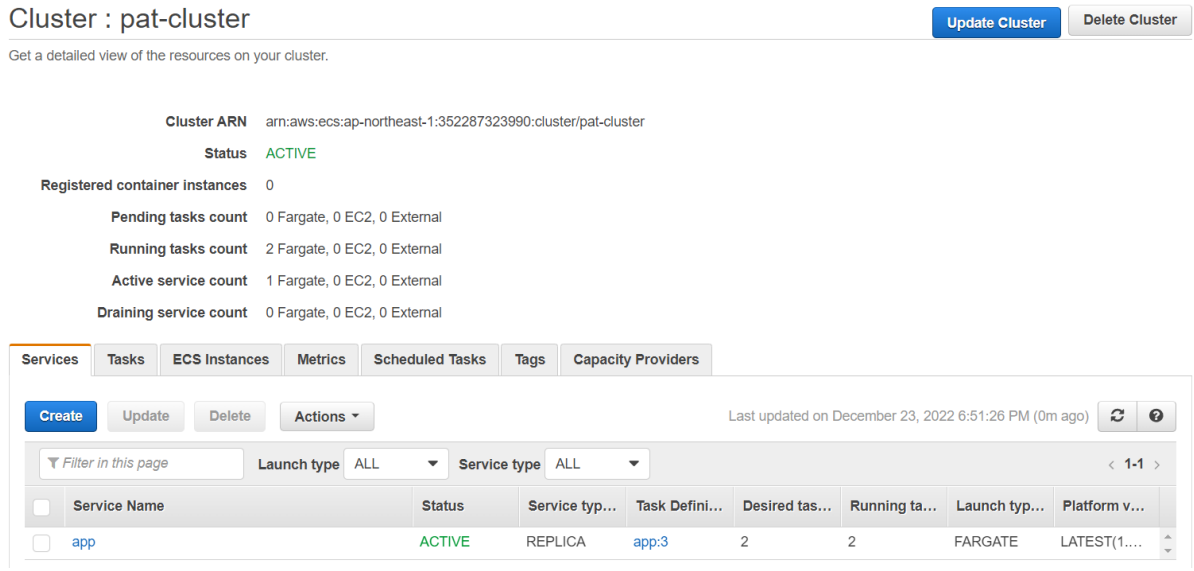
```
C:\xampp\apache\bin>ab -n 100 -c 1 http://pat-lb-509681441.ap-northeast-1.elb.amazonaws.com/
This is ApacheBench, Version 2.3 <$Revision: 1901567 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking pat-lb-509681441.ap-northeast-1.elb.amazonaws.com (be patient).....done


Server Software:        nginx/1.17.1
Server Hostname:        pat-lb-509681441.ap-northeast-1.elb.amazonaws.com
Server Port:            80

Document Path:          /
Document Length:        3279 bytes

Concurrency Level:      1
Time taken for tests:   123.084 seconds
Complete requests:      100
Failed requests:        3
   (Connect: 3, Receive: 0, Length: 0, Exceptions: 0)
Total transferred:      351300 bytes
HTML transferred:       327900 bytes
Requests per second:    0.81 [#/sec] (mean)
Time per request:       1230.842 [ms] (mean)
Time per request:       1230.842 [ms] (mean, across all concurrent requests)
Transfer rate:          2.79 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       75  503 2167.0     93   15146
Processing:    79   96   9.8     95     160
Waiting:       79   96   9.9     95     160
Total:        165  599 2167.2    187   15254

Percentage of the requests served within a certain time (ms)
  50%    187
  66%    194
  75%    198
  80%    200
  90%    212
  95%   3191
  98%  15208
  99%  15254
 100%  15254 (longest request)
```

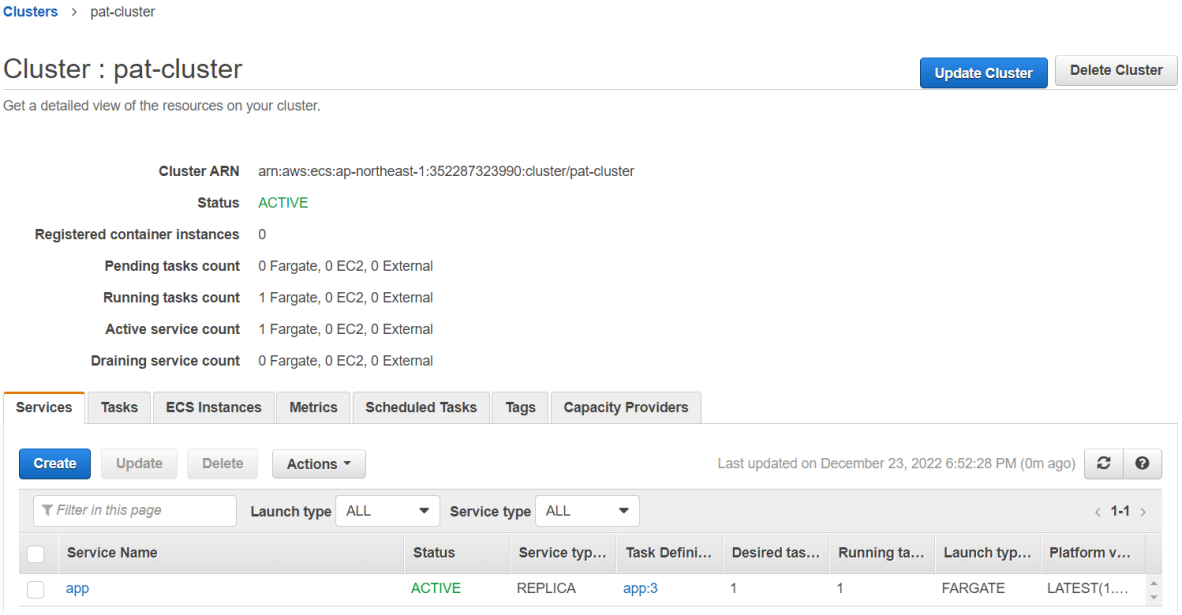



Dapat dilihat bahwa 100 request dengan rata-rata 0.8 request/detik menyebabkan *request count per target* naik melewati batas 10, sehingga *container* akan di-*scale-up* sehingga jumlahnya menjadi 3 (*max count* karena *clamping* melebihi target).




Setelah beberapa menit kemudian, *autoscale* akan melakukan *scale-down* jumlah *instance* sehingga kembali menjadi min count (1).




Oleh karena itu, dapat disimpulkan bahwa *autoscale* telah bekerja dengan baik.