TUGAS TAMBAHAN HENNGE
IF4031 PENGEMBANGAN APLIKASI TERDISTRIBUSI

Nginx ECS *Cluster Behind an Application Load Balancer Using Fargate Launch Type*

Disusun oleh:
Hansel Valentino Tanoto
13520046/K-02

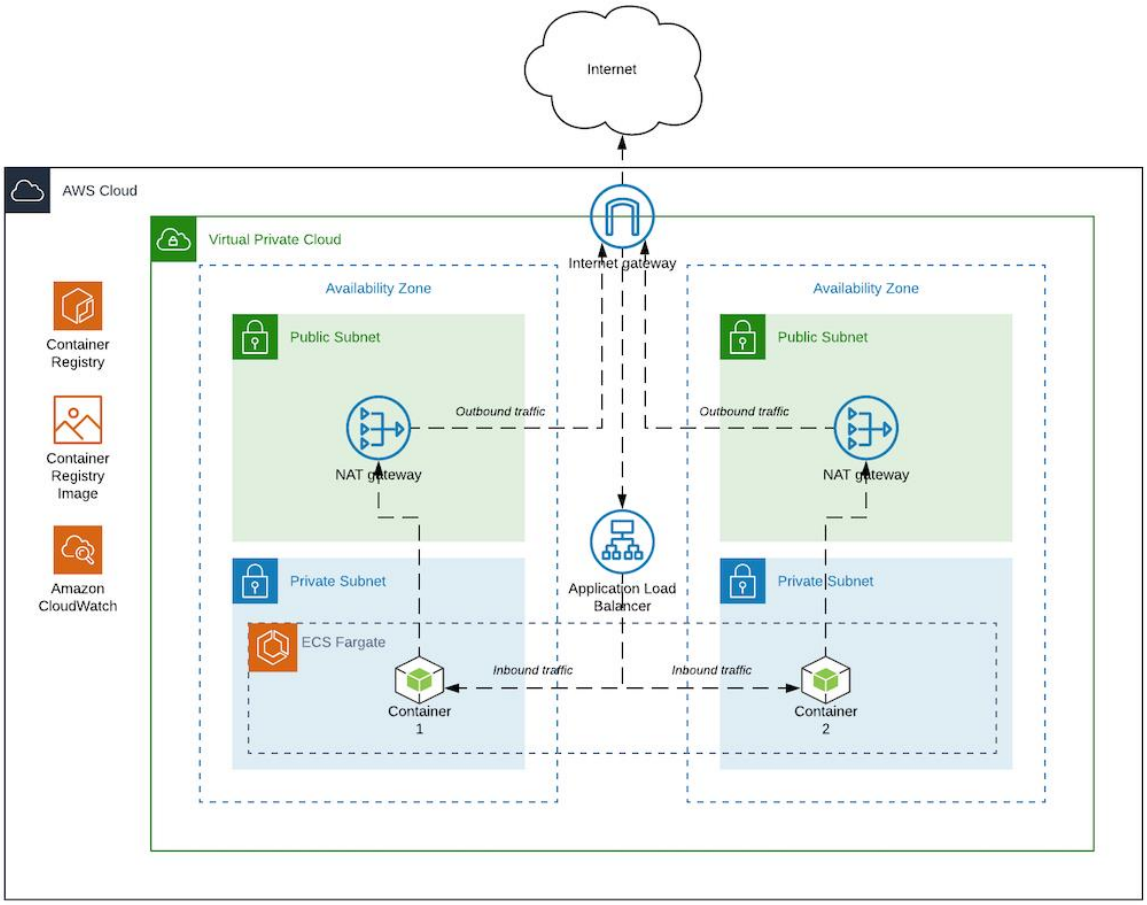
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

1. *Implementation Story*

Dalam pengerjaan tugas ini, saya mengacu pada beberapa dokumentasi berikut ini:

- https://docs.aws.amazon.com/AmazonECS/latest/userguide/what-is-fargate.html yang merupakan dokumentasi resmi dari AWS mengenai definisi dan penggunaan layanan Amazon ECS dengan AWS Fargate.
- https://registry.terraform.io/providers/hashicorp/aws/latest/docs, berisi dokumentasi resmi dari HashiCorp Terraform mengenai penggunaan *syntax* yang pada Terraform.
- https://engineering.finleap.com/posts/2020-02-20-ecs-fargate-terraform/, yang berisi dokumentasi langkah-langkah mengimplementasikan Amazon ECS dengan Fargate dan Terraform yang dipublikasikan oleh Finleap, salah satu perusahaan *fintech* di Eropa.

Berikut adalah diagram arsitektur yang diimplementasikan dalam tugas ini:



(Sumber: https://miro.medium.com/max/1049/1*zmk2GDtVpArs3lysisYEZQ.png)

Berikut adalah struktur *file* kode program dalam implementasi tugas ini:

- `main.tf`

  Berisi pendefinisian *cluster*, *task*, dan *service*. *Cluster* yang didefinisikan di sini, memiliki opsi *Container Insights* yang diaktifkan (*enabled*) agar bisa mengumpulkan *metrics* pada level *cluster*, *task*, dan *service*. Lalu, *Task* didefinisikan sebagai sebuah NGINX *service* (versi 1.23.1) dengan CPU *unit* sebesar 256 dan *memory* sebesar 1024 MB (1 GB). Dan, *Service* didefinisikan dengan *launch type* berupa Fargate

- `variables.tf`

  Berisi pendefinisian beberapa variabel global yang diperlukan dalam *file* konfigurasi Terraform (`.tf`) lainnya, yaitu berupa *region*, *availability zones*, *access key*, *secret key*, VPC CIDR *block*, serta *public* dan *private subnet* CIDR *block*.

- `provider.tf`

  Berisi konfigurasi versi minimal Terraform dan AWS yang digunakan, yaitu versi 0.12.0 untuk Terraform dan versi 2.0.0 untuk AWS.

- `vpc.tf`

  Berisi konfigurasi VPC (*Virtual Private Cloud*), *Internet Gateway*, NAT *Gateway*, *Public Subnet*, *Private Subnet*, *Route Table*, dan *Route Table Association*. VPC adalah representasi virtual dari jarigan aplikasi di AWS *cloud*. Lalu, *Internet Gateway* digunakan sebagai penghubung komunikasi antara internet dengan VPC. Sementara, NAT *Gateway* berfungsi sebagai jembatan komunikasi (mengirimkan *response*) dari aplikasi yang berada di *public subnet* ke dunia luar melalui *internet gateway*. Sedangkan *Route Table* berfungsi untuk mengatur bagaimana *network traffic* dari *subnet* atau *gateway* akan diarahkan dan *route table association* digunakan untuk mengasosiasikan *route table* tersebut dengan *subnet*-nya.

- `load-balancer.tf`

  Berisi konfigurasi dari *load balancer* yang digunakan beserta *target group* dan *listener*-nya. *Load balancer* yang digunakan berupa tipe *application load balancer* dengan *security group*-nya yang telah didefinisikan di *file* `security-group.tf`. *Listener* dari *load balancer* ini berfungsi untuk menerima dan meneruskan (*forward*) *request* yang datang dari dunia luar ke aplikasi yang berada di *public subnet*. *Listener* ini sendiri akan mendengarkan *request* pada *port* 80, yaitu menggunakan protokol HTTP.

Sementara itu, *target group* dari *load balancer* ini adalah aplikasi/*task* yang akan dijalankan.

- `security-group.tf`
  Berisi pendefinisian *security group* untuk ECS *cluster* dan *Application Load Balancer*. *Security group* dari *load balancer* hanya akan memperbolehkan akses dengan protokol TCP pada *port* 80 (HTTP). Sedangkan *security group* pada ECS *cluster* hanya akan memperbolehkan akses ke *port* yang tereskpos oleh *task* tersebut.

- `auto-scaling.tf`
  Berisi *autoscaling policies* untuk ECS *service*. Pada *file* ini, terdapat pendefinisian IAM *role* yang akan mengatur dan melakukan mekanisme *scaling* untuk aplikasi tersebut. *Role* tersebut akan diberikan (di-*attach*) dengan *policy* berupa Amazon EC2 *Container Service Autoscale Role*. Aturan (*policy*) ini akan melakukan *scaling* pada ECS *service* berdasarkan *target value*, yaitu jumlah rata-rata *request* per target yang di-*set* nilainya 10.

- `output.tf`
  Berisi kode program untuk menampilkan DNS dari *load balancer* ke *console* (*terminal*).

- `keys.tfvars`
  Berisi *secret key* dan *access key* yang bersifat rahasia untuk bisa menjalankan *service* (aplikasi) ini. Jadi untuk menjalankan program ini, silakan isi variabel `access_key` dan `secret_key` dengan nilai *key* dari akun AWS masing-masing.

Kode program pada tugas ini dapat diakses pada *link* berikut ini:
https://github.com/HanselTanoto/AWS-ECS-Fargate-behind-Application-Load-Balancer

2. *Difficulties*

Beberapa kesulitan yang saya alami selama pengerjaan tugas ini di antaranya:

1. Kesulitan menentukan konfigurasi yang tepat untuk pengerjaan tugas ini karena banyak sekali opsi yang dimiliki setiap komponen pada AWS *networking*.
2. Perlu mempelajari *syntax* bahasa baru, yaitu Terraform meskipun tidak begitu sulit. Namun, tetap membutuhkan waktu lebih untuk dapat terbiasa dan memahami maknanya.
3. Kesulitan mengetahui error yang terjadi karena kesalahan konfigurasi *resource* karena tidak terdapat pesan kesalahan yang spesifik (eksplisit) sehingga harus mencari informasi dari beberapa sumber/forum di internet.

3. *Testing*

Berikut adalah langkah-langkah menjalankan program ini:

1. *Clone repository* Github dari program ini.
2. Isi *secret key* dan *access key* pada *file* `keysexample.tfvars` sesuai akun AWS masing-masing, lalu *rename file* menjadi `keys.tfvars`. Atau dapat langsung membuat *file* baru bernama `keys.tfvars` dan mengisinya sesuai format pada `keysexample.tfvars`.
3. Jalankan perintah **`terraform init`** pada terminal (CLI) untuk menginisialisasi Terraform.
4. Jalankan perintah **`terraform apply -var-file="keys.tfvars"`** pada terminal (CLI) untuk mengeksekusi kode program tersebut. Apabila muncul pesan "*Do you want to perform these actions*?", jawab dengan `yes`.
5. Pada akhir *line output* di terminal akan muncul DNS dari *load balancer*. Namun apabila tidak muncul, DNS tersebut bisa ditampilkan dengan perintah **`terraform output`** pada terminal (CLI).
6. Untuk mengetes apakah ECS *cluster* sudah berjalan (*running*), tuliskan perintah **`curl http://<DNS-Load-Balancer>`**, misalnya:

```
curl http:// lb-649289153.us-east-1.elb.amazonaws.com
```

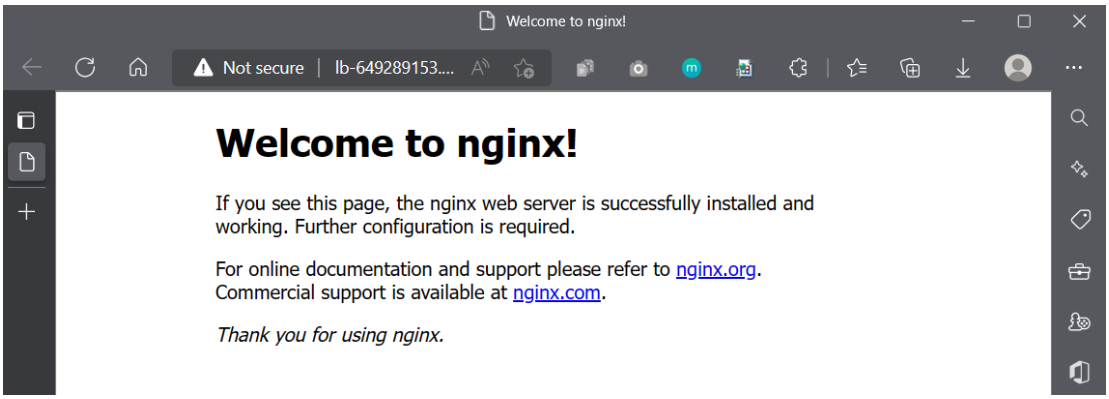
Apabila muncul hasil seperti berikut ini, maka *cluster* berhasil dijalankan.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Selain dengan perintah tersebut, bisa juga dengan membuka *link* `http://<DNS-Load-Balancer>` tadi pada *browser* yang akan menampilkan hal berikut bila *cluster* berhasil dijalankan.



7. Untuk mengakhiri program dan men-*destroy resource*, jalankan perintah **`terraform destroy -var-file="keys.tfvars"`**.

4. *Lesson Learned*

Melalui tugas ini, saya belajar banyak hal mengenai *service* yang disediakan oleh Amazon untuk membantu *developer* dalam pengembangan aplikasi berbasis *cloud*, salah satunya dengan ECS *cluster* ini yang memungkinkan penggunanya tidak perlu mengatur banyak konfigurasi server dan lain-lain. Pengguna hanya perlu mengatur beberapa konfigurasi untuk menyesuaikannya dengan kebutuhan masing-masing. Melalui dokumentasi dan *hands on* tugas ini, saya bisa memperoleh pengetahuan dasar mengenai komponen-komponen dalam AWS *networking* dan bagaimana cara mengkonfigurasinya untuk membuat sebuah *service* (*cluster*) sederhana. Selain itu, saya juga bisa belajar penggunaan bahasa Terraform yang sering digunakan untuk pengembangan infrastruktur *cloud*. Berikut adalah sedikit rangkuman dari apa yang saya pelajari dari tugas ini.

Pada intinya, Fargate merupakan teknologi yang bisa digunakan dengan Amazon ECS untuk menjalankan suatu *container* tanpa harus mengelola server atau *cluster* EC2 Amazon sendiri. Amazon ECS (*Elastic Container Service*) sendiri merupakan salah satu layanan dari AWS untuk mengkapsulasi *service* dalam suatu kontainer sehingga memudahkan proses *deployment*, pengembangan, dan *scaling* aplikasi. Beberapa *use case* dari Amazon ECS di antaranya, untuk men-*deploy* aplikasi di *hybrid environment*, untuk aplikasi yang memerlukan *batch processing*, dan aplikasi yang membutuhkan skalabilitas. Jadi, dengan Fargate, kita tidak perlu menyediakan, mengkonfigurasi, atau men-*scaling cluster* VM (*Virtual Machine*) secara mandiri. Ketika kita menjalankan *task* dan *service* Amazon ECS dengan Fargate *launch type* kita hanya perlu mengemas aplikasi dalam suatu *container*, menentukan konfigurasi OS, CPU, dan *memory,* mendefinisikan *networking* dan IAM *policies*, dan terakhir me-*launch* aplikasi tersebut. Komponen yang terdapat pada AWS Fargate, yaitu *culsters*, *task definitions*, *tasks*, dan *services*. *Cluster* adalah *logical grouping* dari *tasks* atau *services* untuk mengisolasi aplikasi. Kemudian, *task definition* adalah *text file* yang mendefinisikan *container* pembentuk aplikasi atau biasa disebut sebagai *blueprint* aplikasi. Sementara, *task* adalah instansiasi dari *task definition* dalam suatu *cluster*. Lalu yang terakhir, *service* digunakan untuk mempertahankan jumlah *tasks* yang ingin dijalankan secara bersamaan dalam Amazon ECS *cluster*.

Terraform sendiri merupakan salah satu *tool* berupa *infrastructure as code* untuk mendefinisikan konfigurasi *cloud resources* dalam bahasa yang mudah dipahami manusia (*human readable code*). Cara kerja dari Terraform adalah dengan membuat dan mengelola *cloud resources* pada *cloud platform provider* seperti AWS melalui API-nya. Alur kerja (*workflow*) utama dari Terraform adalah *write*, *plan*, dan *apply*. *Write* adalah fase pendefinisian

resource, misalnya VPC, *load balancer*, dan *security group*. Lalu, pada fase *plan*, Terraform akan membuat rencana eksekusi untuk *create*, *update*, atau *destroy* infrastruktur *cloud* berdasarkan *file* konfigurasi yang ditulis sebelumnya. Terakhir, pada fase *apply*, Terraform akan menjalankan operasi sesuai rencana eksekusi yang didefinisikan sebelumnya. Beberapa kelebihan dari Terraform, yaitu:

- Bisa mengelola banyak infrastruktur dari berbagai *cloud platform provider*
- Memiliki sistem *tracking* untuk meminta *approval* sebelum memodifikasi infrastruktur yang bersangkutan
- Bisa mengotomisasi perubahan karena bahasanya yang bersifat deklaratif
- Men-*support* penggunaan *module* sehingga bisa meningkatkan *reusability* dari kode program
- Memungkinkan kolaborasi antar anggota tim melalui Terraform Cloud

Untuk alur jalannya program pada tugas ini, yaitu pertama-tama *request* akan diterima oleh *internet gateway* yang menjadi gerbang utama antara VPC dengan internet (*outside world*). Kemudian *request* tersebut akan diproses oleh *load balancer* untuk mengalokasikan *task* ke *resource* yang tersedia. Lalu *containers* pada *private subnet* akan menjalankan *task* tersebut dan mengirimkan hasilnya (*response*) ke NAT *gateway* yang berada di *public subnet* untuk kemudian diteruskan ke *client*/internet melalui *internet gateway*.

5. Bonus: *Target Tracking Autoscaling to Maintain* 10 *Request/Target*

Implementasi dari *target tracking autoscaling* ini terdapat pada *file* `auto-scaling.tf` yang sudah dijelaskan pada bagian implementasi sebelumnya. Berikut adalah potongan kode program yang mengatur mekanisme *autoscaling* ini.

```
# Create the autoscaling policy
# This policy will scale the ECS service up or down based on the target value
# The target value is the average number of requests per target
# The target value is set to 10 requests per target
resource "aws_appautoscaling_policy" "ecs_service_policy" {
    name               = "ecs-service-policy"
    policy_type        = "TargetTrackingScaling"
    resource_id        = aws_appautoscaling_target.ecs_service_target.resource_id
    scalable_dimension = aws_appautoscaling_target.ecs_service_target.scalable_dimension
    service_namespace  = aws_appautoscaling_target.ecs_service_target.service_namespace
    target_tracking_scaling_policy_configuration {
        predefined_metric_specification {
            predefined_metric_type = "ALBRequestCountPerTarget"
            resource_label         =
"app/${aws_lb.lb.name}/${basename("${aws_lb.lb.id}")}/targetgroup/${aws_lb_target_group.
lb_target_group.name}/${basename("${aws_lb_target_group.lb_target_group.id}")}"
        }
        target_value = 10
    }
}
```

Jadi berdasarkan kode tersebut, didefinisikan *metric type* untuk *autoscaling policy* berupa ALB *Request Count per Target* yang akan melakukan *scaling* berdasarkan jumlah *request per target* tertentu. Lalu berdasarkan spesifikasi bonus, *target value* akan di-*set* dengan nilai 10 untuk mempertahankan jumlah 10 *request*/*target*.

Untuk pengetesannya sendiri, saya menggunakan *tool* tambahan, yaitu Apache Bench (AB) untuk mengirimkan *request* konkuren ke DNS *load balancer*. Berikut adalah contoh penggunaan Apache Bench untuk mengetes spesifikasi bonus ini.

```
ab -n 5000 -c 10 http://lb-649289153.us-east-1.elb.amazonaws.com/
```

Perintah tersebut akan mengirimkan total 5000 *request* dengan 10 *request* bersamaan setiap waktu ke *application load balancer* dari ECS *cluster* sesuai DNS yang tertera. Berikut adalah hasil (*output*) dari perintah tersebut:

```
C:\Apache24\bin>ab -n 5000 -c 10 http://lb-649289153.us-east-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 lb-649289153.us-east-1.elb.amazonaws.com (be patient)
Completed 500 requests
Completed 1000 requests
Completed 1500 requests
Completed 2000 requests
Completed 2500 requests
Completed 3000 requests
Completed 3500 requests
Completed 4000 requests
Completed 4500 requests
Completed 5000 requests
Finished 5000 requests


Server Software:        nginx/1.23.1
Server Hostname:        lb-649289153.us-east-1.elb.amazonaws.com
Server Port:            80

Document Path:          /
Document Length:        615 bytes

Concurrency Level:      10
Time taken for tests:   1250.322 seconds
Complete requests:      5000
Failed requests:        0
Total transferred:      4240000 bytes
HTML transferred:       3075000 bytes
Requests per second:    4.00 [#/sec] (mean)
Time per request:       2500.644 [ms] (mean)
Time per request:       250.064 [ms] (mean, across all concurrent requests)
Transfer rate:          3.31 [Kbytes/sec] received
```

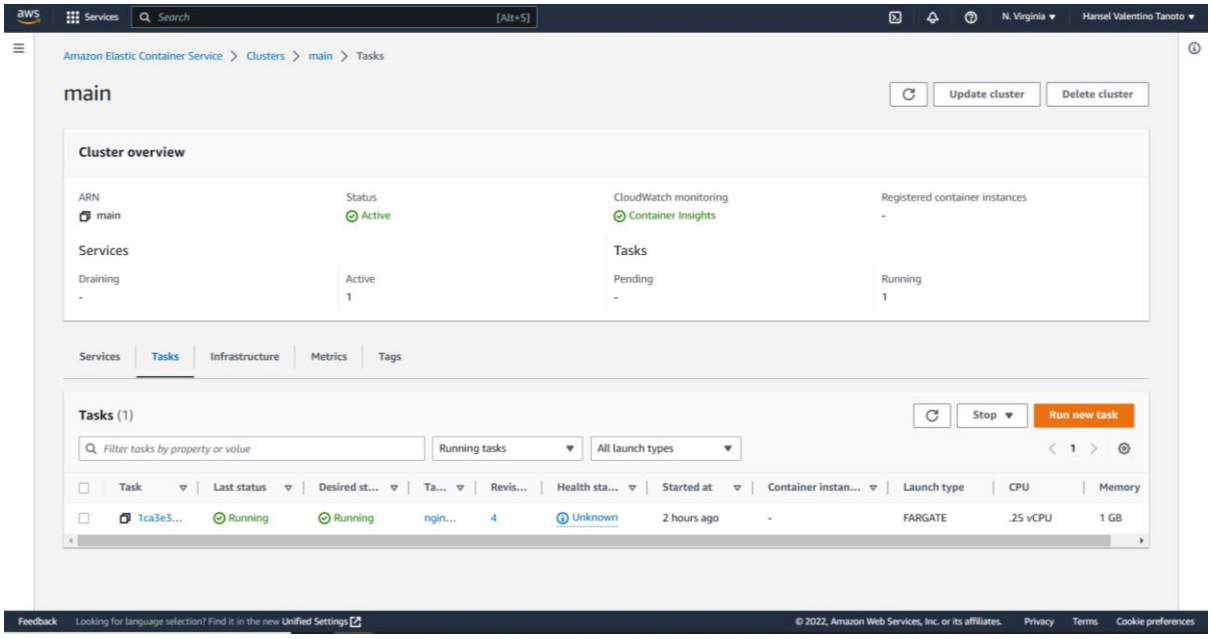
```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:      240  250  21.8    245    1252
Processing:   336 2247 111.8   2230    3896
Waiting:      335 1639 512.3   1724    3601
Total:        578 2497 118.1   2478    4184

Percentage of the requests served within a certain time (ms)
  50%   2478
  66%   2493
  75%   2505
  80%   2515
  90%   2549
  95%   2608
  98%   2709
  99%   2930
 100%   4184 (longest request)

C:\Apache24\bin>
```

Dapat dilihat bahwa semua request berhasil dikirimkan yang dapat dilihat pada *field completed requests* di atas.

Berikut adalah tampilan *tasks list* pada ECS *cluster* sebelum dikirimkan 5000 *request* sesuai perintah di atas.



Dan gambar di bawah ini adalah tampilan *tasks list* setelah dikirimkan 5000 *request* di atas. Dapat dilihat *cluster* menyesuaikan jumlah *tasks* menjadi 10 *request* per *target*.