

OPSWAT.®

TRAINEE FINAL PROJECT

TU HUYNH

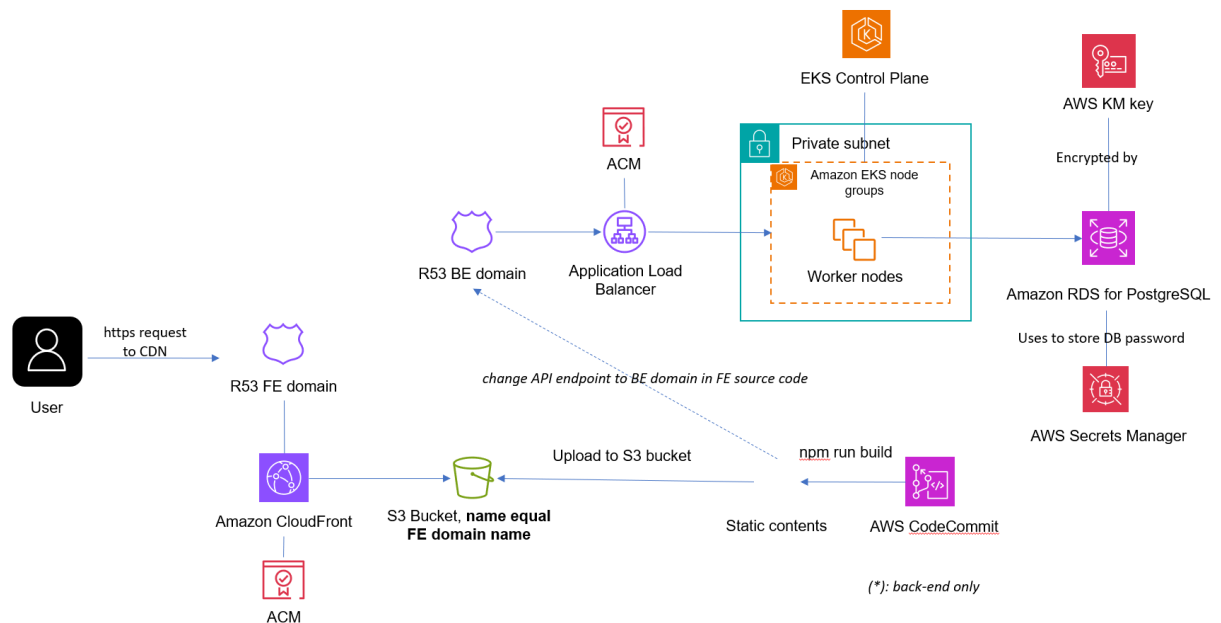
MỤC LỤC

I. GIỚI THIỆU.....	3
1. Yêu cầu.....	3
2. Tổng quan.....	3
II. DATABASE	6
1. Tổng quan.....	6
2. Database	6
III. BACKEND	9
1. Tổng quan.....	9
2. Container image	10
3. Networking.....	14
4. ACM.....	17
5. EKS	19
a. EKS cluster	19
b. AWS ELB.....	21
c. Helm chart – todo app.....	23
d. Monitoring	24
e. Auto-scaling (pod & node level).....	26
6. Route53	27
IV. FRONTEND	30
1. Tổng quan.....	30
3. ACM.....	34
4. Cloudfront	35
5. Route53	37
V. KẾT QUẢ	39
VI. PIPELINE.....	40
VII. TỔNG KẾT	42
VII. LỜI CẢM ƠN	42

I. GIỚI THIỆU

1. Yêu cầu

Project này đặt ra mục tiêu deploy một todo-app đơn giản gồm 3 phần frontend, backend, database đã được viết sẵn. Các resource được sử dụng cung cấp bởi AWS và được tạo thông qua Terraform. Kết quả thu được sẽ là một app như *hình 1*:



Hình 1: Sơ đồ todo-app

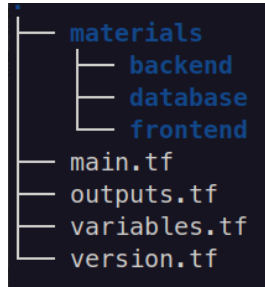
2. Tổng quan

Để thuận tiện cho việc thi công và quản lý, app sẽ được chia thành 3 phần chính:

- Database
- Backend
- Frontend

Trong đó, hầu hết các resource của cả ba phần database, backend, frontend sẽ được tạo ra một cách tự động bằng Terraform. Việc sử dụng Terraform làm cho quá trình quản lý resource trở nên dễ dàng hơn. Khi tạo ra resource bằng Terraform sẽ giảm đi đáng kể thời gian và công sức tạo resource so với thực hiện bằng tay thông qua giao diện người

dùng, do đó giảm đi đáng kể những lỗi liên quan đến human-error. Ngoài ra nếu như tạo resource đúng cách bằng Terraform, có thể hoàn toàn destroy các resource một cách nhanh chóng. Cấu trúc chính của todo-app khi tạo bằng Terraform như *hình 2*:



Hình 2: cấu trúc chính của todo-app

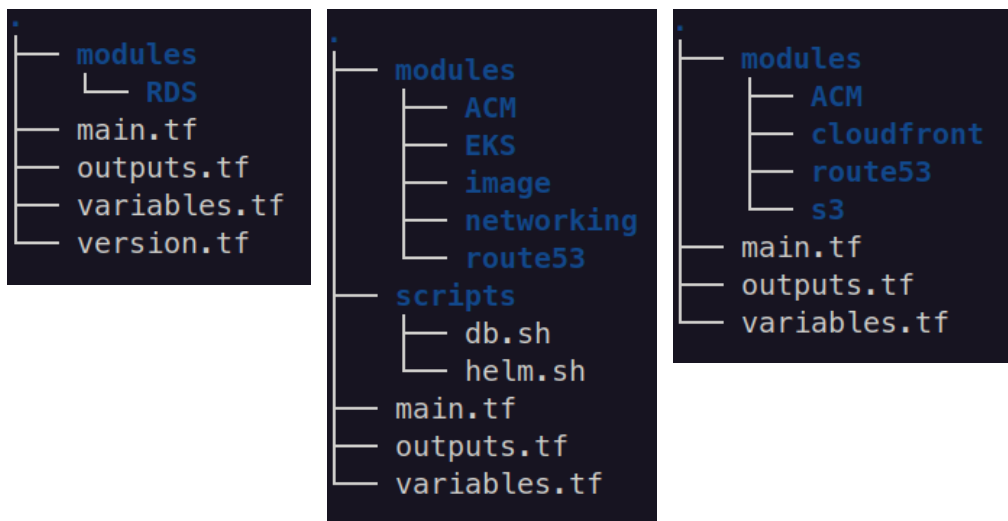
Ứng với 3 folder trong thư mục materials là 3 module độc lập nhau gồm database-module, backend-module và frontend-module. Và trong từng module sẽ chứa các sub-module khác, chi tiết các sub-module sẽ được nêu rõ trong các phần sau.

```
main.tf
main.tf > ...
1 #####
2 # Database
3 #####
4
5 module "database" {
6   source = "./materials/database"
7 }
8
9 #####
10 # Backend
11 #####
12
13 module "backend" {
14   source = "./materials/backend"
15   image_username = module.database.database_username
16   image_password = module.database.database_password
17   image_host = module.database.database_host
18   image_port = module.database.database_port
19   image_database = module.database.database_database
20 }
21
22 #####
23 # Frontend
24 #####
25
26 module "frontend" {
27   source = "./materials/frontend"
28 }
29
30
```

Hình 3: Terraform-root

Trong file *main.tf* của thư mục root sẽ bao gồm cả 3 module là: database, backend và frontend. Như đã đề cập phía trên rằng 3 module này sẽ có thể hoạt động độc lập nhau. Tuy nhiên do phần backend sẽ yêu cầu một số thông số từ database, nên là backend-module sẽ lấy một vài thông số từ database-module. Nếu như backend-module không lấy tham chiếu các thông số từ database-module, nó vẫn có thể hoạt động độc lập với cách thông số mặc định.

Sâu vào bên trong thư mục materials sẽ là các module đã nêu ở trên cùng với các sub-modules khác bên trong đó. Cả 3 module này ban đầu được phát triển một cách độc lập với nhau, sau đó mới được ghép lại thành một module lớn. Chính vì vậy mà khi đi vào từng module sẽ có các cấu file trúc tương tự như ở thư mục root.

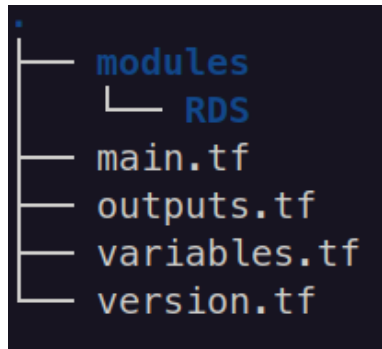


Hình 4: cấu trúc của các sub-modules

II. DATABASE

1. Tổng quan

Theo như *hình 5* thì database-module sẽ có cấu trúc như sau:



Hình 5: cấu trúc của database-module

Thực chất đối với những module chỉ cần tham chiếu với chỉ 1 sub-module thì không cần phải tuân theo cấu trúc này mà có thể định nghĩa các resources ở trong file *main.tf* và không cần tạo thêm các sub-module.

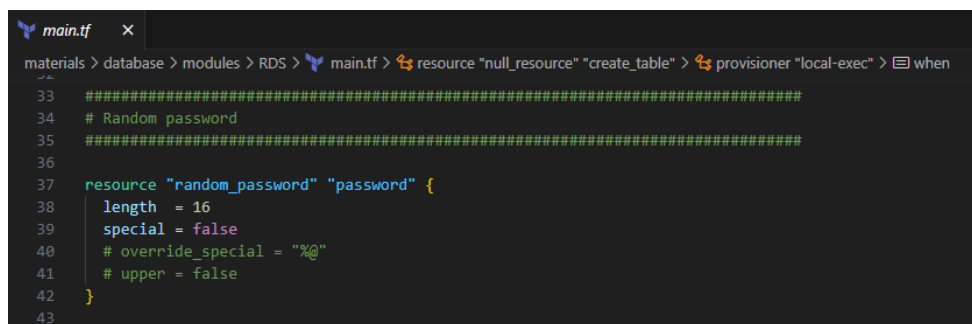
2. Database

```
main.tf x
materials > database > modules > RDS > main.tf > locals
51 #####
52 # RDS
53 #####
54
55 resource "aws_db_instance" "database" {
56   # vpc_security_group_ids = [data.aws_vpc.vpc.id]
57   identifier      = "${var.rds-name}-db-${random_password.random_name.result}"
58   allocated_storage = var.rds-allocated_storage
59   storage_type    = var.rds-storage_type
60   db_name         = var.rds-db_name
61   engine          = var.rds-engine
62   engine_version  = var.rds-engine_version
63   instance_class  = var.rds-instance_class
64   username        = var.rds-username
65   password        = random_password.password.result
66   kms_key_id      = data.aws_kms_alias.kms_key.target_key_arn
67   port            = var.rds-port
68   parameter_group_name = var.rds-parameter_group_name
69   skip_final_snapshot = var.rds-skip_final_snapshot
70   publicly_accessible = var.rds-publicly_accessible
71   storage_encrypted = var.rds-storage_encrypted
72 }
73
```

Hình 6: Terraform - RDS

Vì lý do database-module được tạo trước backend-module và VPC cho project được tạo trong backend-module, nên khi tạo database trên RDS thì VPC cho todo-app vẫn chưa được tạo. Chính vì vậy mà subnets mà database dùng thuộc VPC mặc định. Như vậy để cho phần backend có thể truy cập vào trong database, thẻ `publicly_accessible` sẽ được gán thành “true”. Tuy là database không được đặt trong private subnets, nhưng lại khá tiện cho việc testing ở từ máy tính local. Để đưa database vào hẳn trong private subnets, VPC nên được tạo ra đầu tiên trong toàn bộ project, và dùng các private subnets của VPC đó cho database.

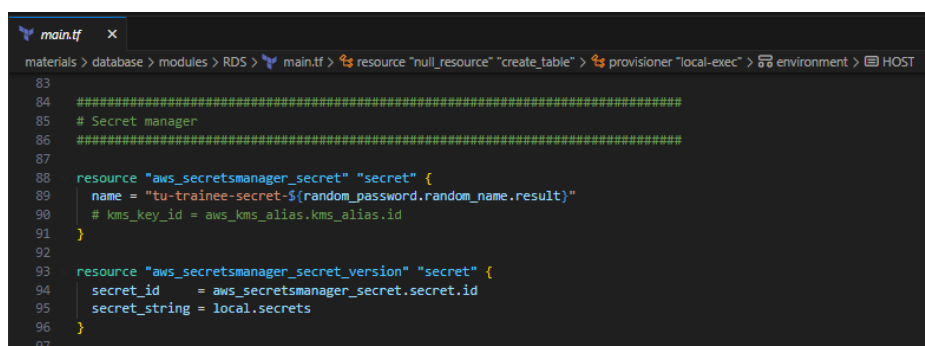
Phần mật khẩu sẽ được tạo ra một cách tự động và ngẫu nhiên bằng Terraform. Mật khẩu được tạo ra cho todo-app có độ dài 16 ký tự bao gồm: số, chữ cái thường và chữ cái hoa.



```
main.tf
materials > database > modules > RDS > main.tf > resource "null_resource" "create_table" > provisioner "local-exec" > when
33 #####
34 # Random password
35 #####
36
37 resource "random_password" "password" {
38     length = 16
39     special = false
40     # override_special = "%@"
41     # upper = false
42 }
43
```

Hình 7: Terraform - random password

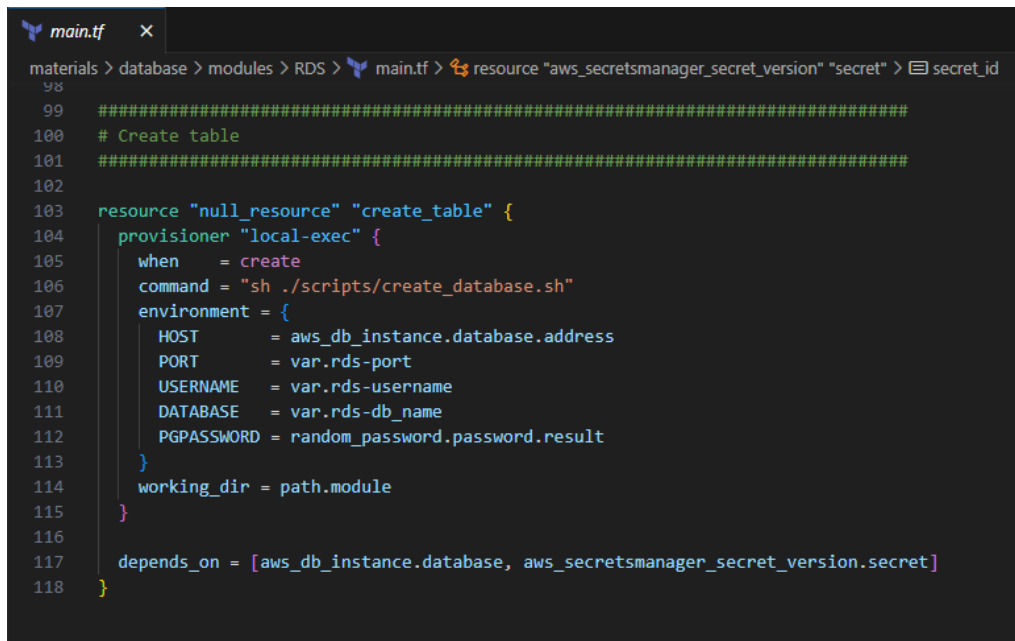
Password sau khi tạo ra sẽ được lưu ở trên AWS Secret Manager. Ngoài password ra, các thông số khác của database như username, host, port, database name cũng sẽ được lưu trên AWS Secret Manager.



```
main.tf
materials > database > modules > RDS > main.tf > resource "null_resource" "create_table" > provisioner "local-exec" > environment > HOST
83
84 #####
85 # Secret manager
86 #####
87
88 resource "aws_secretsmanager_secret" "secret" {
89     name = "tu-trainee-secret-${random_password.random_name.result}"
90     # kms_key_id = aws_kms_alias.kms_alias.id
91 }
92
93 resource "aws_secretsmanager_secret_version" "secret" {
94     secret_id = aws_secretsmanager_secret.secret.id
95     secret_string = local.secrets
96 }
97
```

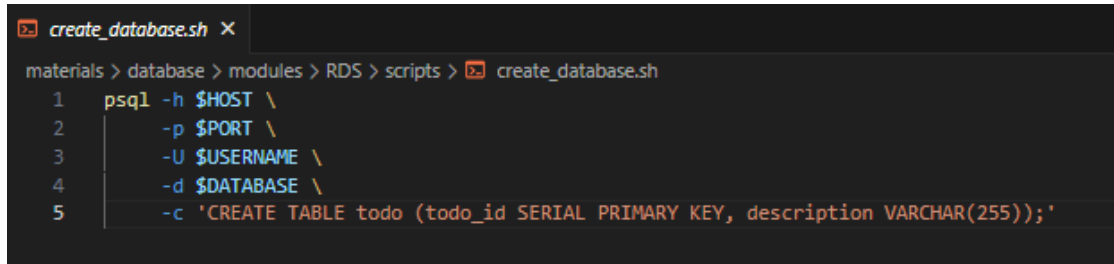
Hình 8: Terraform - Secret Manager

Bước cuối cùng trong database-module là tạo ra các table cho todo-app. Việc tạo bảng được thực hiện thông qua bash scripting và Terraform null_resources.



```
main.tf
materials > database > modules > RDS > main.tf > resource "aws_secretsmanager_secret_version" "secret" > secret_id
98
99 #####
100 # Create table
101 #####
102
103 resource "null_resource" "create_table" {
104   provisioner "local-exec" {
105     when      = create
106     command   = "sh ./scripts/create_database.sh"
107     environment = {
108       HOST      = aws_db_instance.database.address
109       PORT      = var.rds-port
110       USERNAME  = var.rds-username
111       DATABASE  = var.rds-db_name
112       PGPASSWORD = random_password.password.result
113     }
114     working_dir = path.module
115   }
116
117   depends_on = [aws_db_instance.database, aws_secretsmanager_secret_version.secret]
118 }
```

Hình 9: Terraform - create table



```
create_database.sh
materials > database > modules > RDS > scripts > create_database.sh
1  psql -h $HOST \
2    -p $PORT \
3    -U $USERNAME \
4    -d $DATABASE \
5    -c 'CREATE TABLE todo (todo_id SERIAL PRIMARY KEY, description VARCHAR(255));'
```

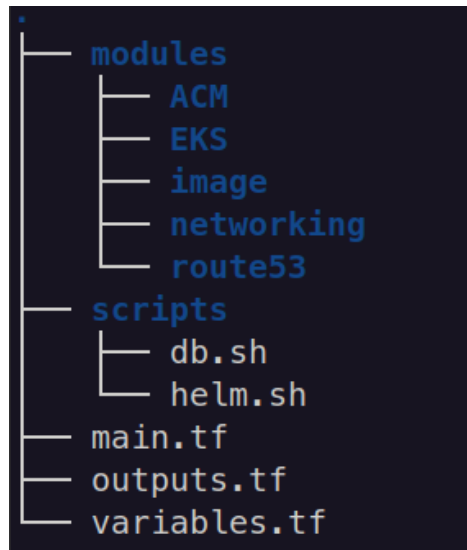
Hình 10: Bash scripting - create table

Nếu như database mang thuộc tính publicly_accessible, các table có thể được tạo ra một cách dễ dàng bằng bash script ở mọi nơi nếu như biết được các parameters của database. Nếu như database được đặt trong private subnets và không thể truy cập từ bên ngoài, table có thể được tạo thông qua các docker container chạy một lần sau khi đã có Kubernetes cluster. Tuy cách này có phần phức tạp hơn, nhưng sẽ mang tính bảo mật hơn.

III. BACKEND

1. Tổng quan

Phần backend của todo-app là phần phức tạp nhất trong projec và sẽ được cấu thành từ 5 sub-module là: acm-module, eks-module, image-module, network-module và route53-module.



Hình 11: cấu trúc của backend-module

Các module như ACM-module, image-module, networking-module có thể chạy cùng một lúc để tạo các resource vì các resource trong cả 3 module này đều không chứa các parameter tham chiếu qua lại từ các module. Chính vì thế khi tạo backend của todo-app, 3 module trên sẽ chạy song song với nhau, sau đó eks-module sẽ tham chiếu các parameter của 3 module đó vào bắt đầu tạo resources, và sau cùng sẽ là route53-module.

```
main.tf M x
materials > backend > main.tf > module "route53"
1 #####
2 # Networking
3 #####
4
5 module "networking" {
6   source = "../modules/networking"
7 }
8
9 #####
10 # ACM certificate
11 #####
12
13 module "acm" {
14   source = "../modules/ACM"
15 }
16
17
18 #####
19 # Container image
20 #####
21
22 module "image" {
23   source = "../modules/image"
24 }
25
26 # data "aws_ecr_repository" "image" {
27 #   name = var.ecr-image_name
28 # }
29
30 #####
31 # EKS
32 #####
33
34 module "eks" {
35   source           = "../modules/EKS"
36   private_subnets_id = module.networking.private_subnets_id
37   depends_on       = [ module.image, module.networking ]
38 }
39
40 #####
41 # Route 53: DNS for backend application
42 #####
43
44 module "route53" {
45   source       = "../modules/route53"
46   depends_on = [ module.eks ]
47 }
```

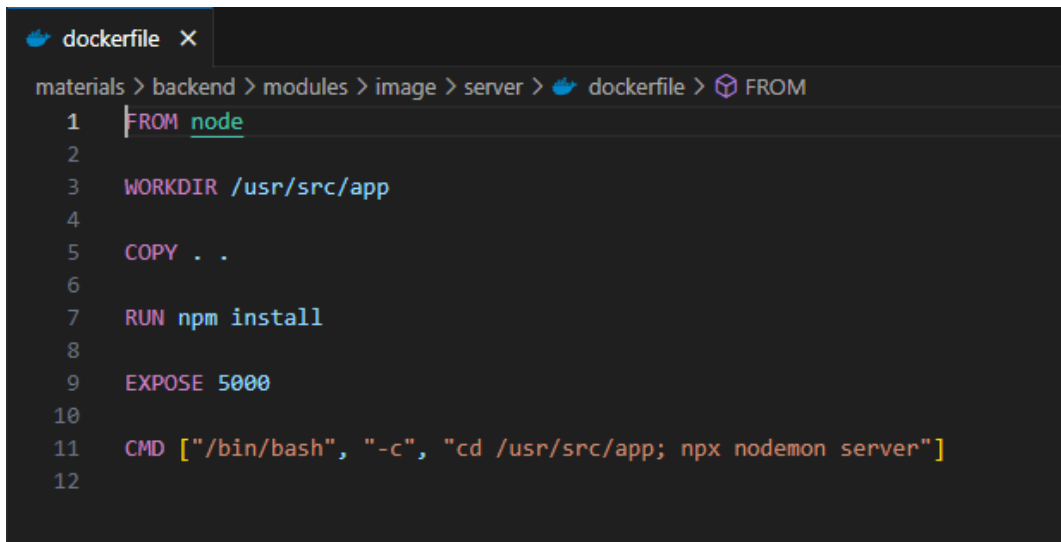
Hình 12: Terraform - backend

2. Container image

Phần backend của todo-app sẽ được đóng gói dưới dạng Docker image. Sau khi đã có Docker image được build sẵn ở trên máy tính local, image sẽ được upload lên trên các Docker repository như Dockerhub hay AWS ECR. Docker repository được sử dụng trong project này là AWS ECR. Tổng quan các bước làm trong phần này như sau:

- Viết dockerfile và để vào trung thư mục backend của todo-app
- Tạo image repo trên AWS ECR
- Build phần backend thành docker image
- Upload các image đã build lên trên AWS ECR

Bước đầu tiên sẽ là viết dockerfile và đặt vào trong cùng thư mục backend của todo-app. Dockerfile chủ yếu là những dòng lệnh cơ bản, đa phần là vì phần backend của todo-app cũng khá đơn giản và không cần phải set up quá nhiều.



```
materials > backend > modules > image > server > dockerfile > FROM
1 FROM node
2
3 WORKDIR /usr/src/app
4
5 COPY . .
6
7 RUN npm install
8
9 EXPOSE 5000
10
11 CMD ["/bin/bash", "-c", "cd /usr/src/app; npx nodemon server"]
12
```

Hình 13: Dockerfile

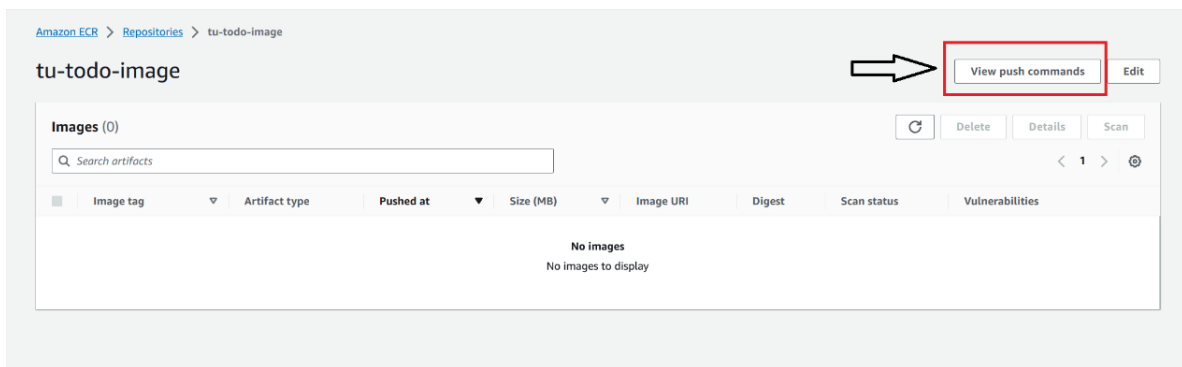
Sau khi đã có dockerfile, backend của app hoàn toàn có thể được build ở dưới local. Tuy nhiên bước build image sẽ được dời lại sau, vì sau khi tạo ra repository AWS ECR sẽ cung cấp sẵn các lệnh để build và push cái image. Điều này tiết kiệm một chút thời gian và công sức cho các lập trình viên không phải tự viết ra các dòng lệnh để build và push các image một cách thủ công, thay vào đó các lập trình viên có thể copy và paste các câu lệnh đó và chạy ở local.

Bước tiếp theo là tạo ra một repository để upload các image. Việc tạo ra repository sẽ được thực hiện bằng Terraform

```
main.tf x
materials > backend > modules > image > main.tf > ...
1 #####
2 # ECR container image repo
3 #####
4
5 resource "aws_ecr_repository" "repo" {
6     name                = var.image-image_name
7     image_tag_mutability = "MUTABLE"
8     force_delete        = true
9
10    image_scanning_configuration {
11        scan_on_push = false
12    }
13 }
14
```

Hình 14: Terraform - AWS ECR

Sau khi đã có một repository, có thể nhấn vào “View push commands” ở trên giao diện người dùng của AWS ECR để tham khảo cách lệnh build và push image



Hình 15: View push commands

Push commands for tu-todo-image

macOS / Linux

Windows

Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see [Getting Started with Amazon ECR](#).

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see [Registry Authentication](#).

- Retrieve an authentication token and authenticate your Docker client to your registry.
Use the AWS CLI:

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin 492804330065.dkr.ecr.us-west-2.amazonaws.com
```

Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.
- Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions [here](#). You can skip this step if your image is already built:

```
docker build -t tu-todo-image .
```
- After the build completes, tag your image so you can push the image to this repository:

```
docker tag tu-todo-image:latest 492804330065.dkr.ecr.us-west-2.amazonaws.com/tu-todo-image:latest
```
- Run the following command to push this image to your newly created AWS repository:

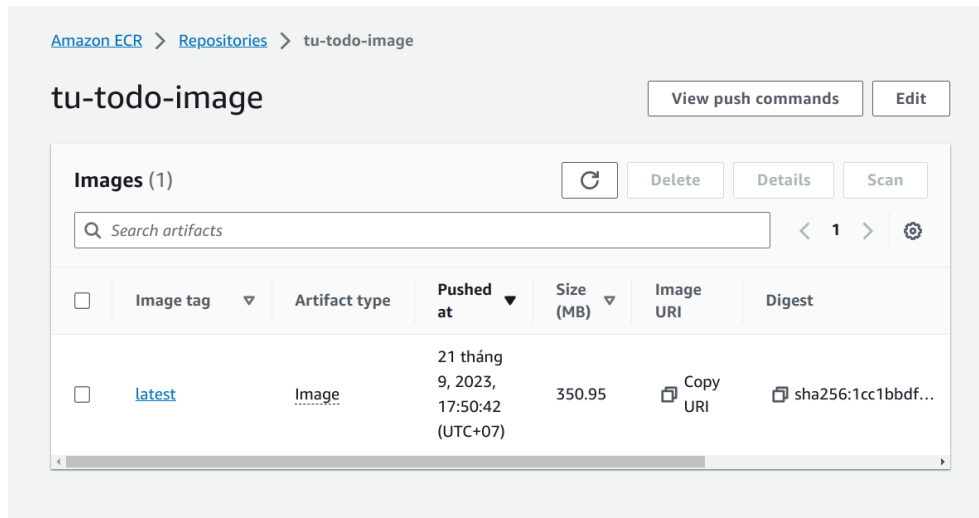
```
docker push 492804330065.dkr.ecr.us-west-2.amazonaws.com/tu-todo-image:latest
```

Close

Hình 16: build & push commands

Những lệnh này có thể thực hiện bằng bash script ở local và dùng `null_resource` của Terraform để chạy những dòng lệnh script đó.

Kết quả sau cùng đạt được khi upload thành công image lên trên AWS ECR sẽ như hình 17



Hình 17: kết quả sau khi push thành công docker image

3. Networking

Trong networking-module, những resource như: VPC, subnet, route table, internet gateway và nat-gateway sẽ được tạo ra. Đầu tiên sẽ tạo VPC, mọi config của phần backend sẽ được đặt trong VPC vừa tạo.

```

main.tf x
materials > backend > modules > networking > main.tf > ...
1 #####
2 # VPC
3 #####
4 provider "aws" {
5   region = var.region-oregon
6 }
7
8 data "aws_availability_zones" "available" {
9   state = "available"
10 }
11
12 resource "aws_vpc" "vpc" {
13   cidr_block           = var.vpc-cidr
14   instance_tenancy     = var.vpc-instance_tenancy
15   assign_generated_ipv6_cidr_block = var.vpc-enable_ipv6
16   tags = {
17     "Name" = var.vpc-name
18   }
19 }
20

```

Hình 18: Terraform - VPC

Từ VPC vừa tạo ở trên, 6 subnet sẽ được tạo ra bao gồm 3 public subnet và 3 private subnet. Khi tạo subnet, để phân biệt subnet này là private hay public sẽ dựa vào thẻ `map_public_ip_on_launch`. Nếu giá trị “true” được gán, subnet sẽ là public subnet và ngược lại đối với private subnet.

Tuy nhiên để cho public subnet kết nối được với internet sẽ phải cần đến một internet gateway. Sau khi internet gateway được tạo ra, một routing table cần được tạo ra để liên kết public subnet với internet gateway.



```
main.tf
materials > backend > modules > networking > main.tf > ...
21 #####
22 # Public subnets
23 #####
24
25 resource "aws_subnet" "subnet_public" {
26   vpc_id            = aws_vpc.vpc.id
27   count             = var.subnet-public_subnet_count
28   cidr_block        = cidrsubnet(aws_vpc.vpc.cidr_block, 8, count.index + 1)
29   availability_zone  = data.aws_availability_zones.available.names[count.index]
30   map_public_ip_on_launch = true
31
32   tags = {
33     "Name" = "${var.name}-public_subnet-${count.index + 1}"
34     "kubernetes.io/role/elb" = 1
35     "kubernetes.io/cluster/tu-cluster" = "owned"
36   }
37 }
38
39 resource "aws_route_table" "public" {
40   vpc_id = aws_vpc.vpc.id
41   tags = {
42     Name = "${var.name}-rtb-public"
43   }
44 }
45
46 resource "aws_route_table_association" "public_assoc" {
47   count = var.subnet-public_subnet_count
48   subnet_id = aws_subnet.subnet_public[count.index].id
49   route_table_id = aws_route_table.public.id
50 }
51
52 resource "aws_internet_gateway" "internet_gateway" {
53   vpc_id = aws_vpc.vpc.id
54   tags = {
55     Name = "${var.name}-igw"
56   }
57 }
58
59 resource "aws_route" "public" {
60   route_table_id = aws_route_table.public.id
61   destination_cidr_block = "0.0.0.0/0"
62   gateway_id = aws_internet_gateway.internet_gateway.id
63 }
```

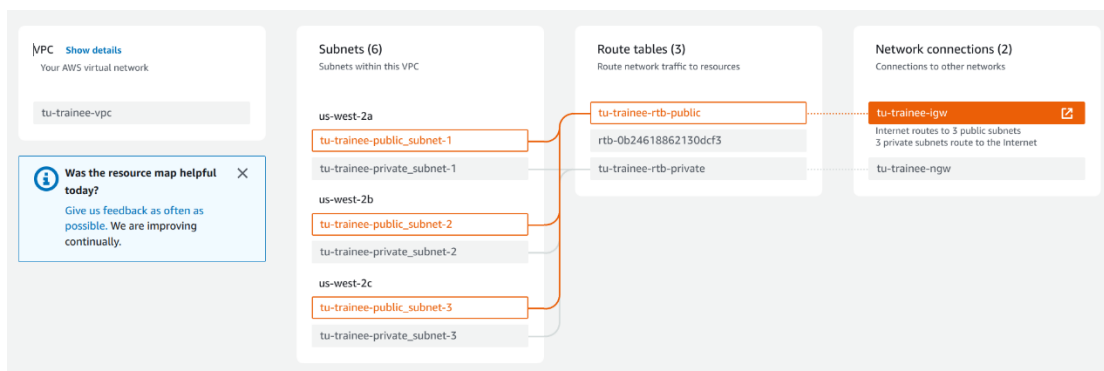
Hình 19: Terraform - public subnets

Đối với private subnet, để có thể kết nối với internet bên ngoài sẽ cần phải nhờ đến nat gateway. Cũng tương tự với public subnet, sau khi tạo ra nat gateway sẽ cần nhờ đến routing table để liên kết private subnet với nat gateway.

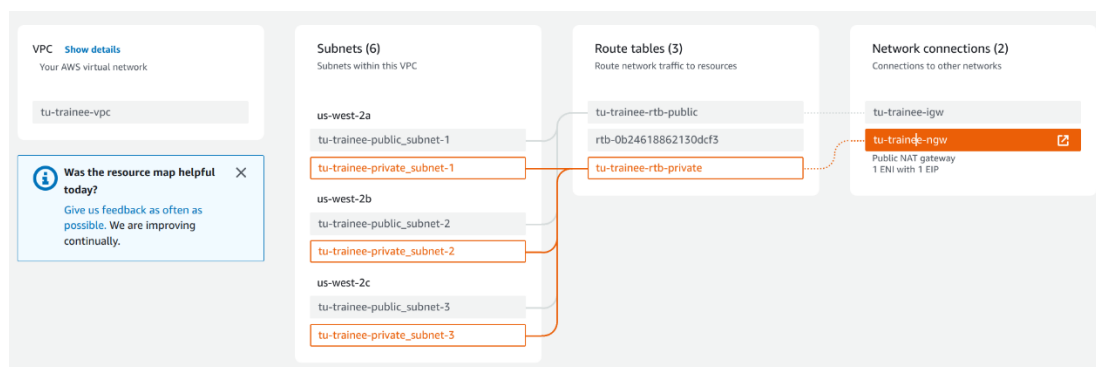
```
main.tf M X
materials > backend > modules > networking > main.tf > resource "aws_route" "public"
64
65 #####
66 # Private subnets
67 #####
68
69 resource "aws_subnet" "subnet_private" {
70   vpc_id            = aws_vpc.vpc.id
71   count             = var.subnet-private_subnet_count
72   cidr_block        = cidrsubnet(aws_vpc.vpc.cidr_block, 8, count.index + 101)
73   availability_zone = data.aws_availability_zones.available.names[count.index]
74   map_public_ip_on_launch = false
75
76   depends_on = [
77     aws_subnet.subnet_public
78   ]
79
80   tags = {
81     "Name" = "${var.name}-private_subnet-${count.index + 1}"
82     "kubernetes.io/role/internal-elb" = 1
83     "kubernetes.io/cluster/tu-cluster" = "owned"
84   }
85 }
86
87 resource "aws_eip" "eip" {
88   # domain = "vpc"
89   depends_on = [aws_vpc.vpc]
90 }
91
92 resource "aws_nat_gateway" "ngw" {
93   allocation_id = aws_eip.eip.id
94   subnet_id     = aws_subnet.subnet_public[0].id
95
96   tags = {
97     Name = "${var.name}-ngw"
98   }
99 }
100
101 resource "aws_route_table" "private" {
102   vpc_id = aws_vpc.vpc.id
103
104   tags = {
105     Name = "${var.name}-rtb-private"
106   }
107 }
108
109 resource "aws_route_table_association" "private_assoc" {
110   count = var.subnet-private_subnet_count
111   subnet_id = aws_subnet.subnet_private[count.index].id
112   route_table_id = aws_route_table.private.id
113 }
114
115 resource "aws_route" "private" {
116   route_table_id = aws_route_table.private.id
117   destination_cidr_block = "0.0.0.0/0"
118   gateway_id = aws_nat_gateway.ngw.id
119 }
```

Hình 20: Terraform - private subnets

Sau khi tạo ra các subnet thành công và tùy chỉnh đúng, sơ đồ subnet – route table – gateway sẽ có dạng như *hình 21* đối với public subnet và như *hình 22* đối với private subnet.



Hình 21: kết quả sau khi tạo public subnets thành công



Hình 22: kết quả sau khi tạo private subnets thành công

4. ACM

Sau khi tạo ra VPC có thể tiến thẳng qua bước tạo Kubernetes cluster, nhưng do các phần tiếp theo sẽ có bước tạo AWS ELB (AWS Elastic Load Balancer), và loadbalancer này sẽ cần đến một HTTPS certificate. Vì thế ACM có thể tạo trước và tạo song song cùng với VPC khi Terraform tạo ra các resource

```

main.tf x
materials > backend > modules > ACM > main.tf > resource "null_resource" "delay"
1 #####
2 # ACM certificate
3 #####
4
5 resource "aws_acm_certificate" "tu-be" {
6     domain_name     = var.acm-domain_name
7     validation_method = var.acm-validation_method
8
9     lifecycle {
10         create_before_destroy = true
11     }
12 }
13
14 resource "aws_acm_certificate_validation" "tu-be" {
15     certificate_arn = aws_acm_certificate.tu-be.arn
16     depends_on     = [aws_acm_certificate.tu-be]
17 }
18

```

Hình 23: Terraform - backend certificate

Ở đây, certificate được request sẽ dùng cho domain name “<https://tu-be.devops-training.opswat.com>”, domain name này sau này sẽ được dùng để truy cập vào phần backend của todo-app. Tuy là load balancer sẽ không dùng domain name này, nhưng vẫn có thể dùng chung certificate. Khi certificate được tạo và verify thành công, kết quả sẽ như hình 24.

	Certificate ID	Domain name	Type	Status
<input type="checkbox"/>	72b73b22-00ea-42ea-8aab-365a0ddf41eb	tu-be.devops-training.opswat.com	Amazon Issued	Issued
<input type="checkbox"/>	a6228737-5a9b-4a75-8040-83ffa971f50a	nhat-fe.devops-training.opswat.com	Amazon Issued	Issued
<input type="checkbox"/>	2f5687a3-0a3c-4f9a-8a31-17ed208247c2	phong-argo-cd.devops-training.opswat.com	Amazon Issued	Issued
<input type="checkbox"/>	10c3795e-b540-4169-8c49-7efb60f5a7ba	phong-serverless.devops-training.opswat.com	Amazon Issued	Issued

Hình 24: kết quả sau khi verify thành công HTTPS certificate

5. EKS

a. EKS cluster

Trước khi tạo ra EKS cluster, một số IAM policy và IAM role cần được tạo ra để có thể sử dụng các resource của AWS.

```
main.tf x
materials > backend > modules > EKS > main.tf > resource "aws_eks_cluster" "tu_cluster" > [ ] depends_on
1 #####
2 # EKS cluster
3 #####
4
5 data "aws_iam_policy_document" "assume_role" {
6   statement {
7     effect = "Allow"
8
9     principals {
10      type       = "Service"
11      identifiers = ["eks.amazonaws.com"]
12    }
13
14    actions = ["sts:AssumeRole"]
15  }
16 }
17 resource "aws_iam_role" "cluster_role" {
18   name       = var.eks-cluster_role_name
19   path       = "/"
20   assume_role_policy = data.aws_iam_policy_document.assume_role.json
21 }
22 resource "aws_iam_role_policy_attachment" "role-AmazonEKSVPCResourceController" {
23   policy_arn = "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"
24   role       = aws_iam_role.cluster_role.name
25 }
26 resource "aws_iam_role_policy_attachment" "role-AmazonEKSClusterPolicy" {
27   policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
28   role       = aws_iam_role.cluster_role.name
29 }
30 }
```

Hình 25: Terraform - policy & role cho EKS cluster

Sau khi các policy và role được tạo ra, có thể được dùng để tạo ra EKS cluster. Subnet được dùng để tạo ra cluster là private subnet.

```
31 resource "aws_eks_cluster" "tu_cluster" {
32   name       = var.eks-cluster_name
33   role_arn   = aws_iam_role.cluster_role.arn
34
35   vpc_config {
36     subnet_ids = var.private_subnets_id
37     endpoint_public_access = true
38     public_access_cidrs   = ["0.0.0.0/0"]
39   }
40
41   # Ensure that IAM Role permissions are created before and deleted after EKS Cluster handling.
42   # Otherwise, EKS will not be able to properly delete EKS managed EC2 infrastructure such as Security Groups.
43   depends_on = [
44     aws_iam_role_policy_attachment.role-AmazonEKSClusterPolicy,
45     aws_iam_role_policy_attachment.role-AmazonEKSVPCResourceController,
46   ]
47 }
```

Hình 26: Terraform - EKS cluster

Tương tự với EKS cluster, EKS node-group sẽ cần đến một số IAM policy và IAM role được tạo trước.

```
main.tf M X
materials > backend > modules > EKS > main.tf > resource "aws_eks_node_group" "tu_nodes" > subnet_ids
48
49 #####
50 # EKS node-group
51 #####
52
53 resource "aws_iam_role" "tu_node_role" {
54   name = var.eks-node_role_name
55
56   assume_role_policy = jsonencode({
57     Statement = [{
58       Action = "sts:AssumeRole"
59       Effect = "Allow"
60       Principal = {
61         Service = "ec2.amazonaws.com"
62       }
63     }]
64     Version = "2012-10-17"
65   })
66 }
67
68 resource "aws_iam_role_policy_attachment" "example-AmazonEKSWorkerNodePolicy" {
69   policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
70   role       = aws_iam_role.tu_node_role.name
71 }
72
73 resource "aws_iam_role_policy_attachment" "example-AmazonEKS_CNI_Policy" {
74   policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
75   role       = aws_iam_role.tu_node_role.name
76 }
77
78 resource "aws_iam_role_policy_attachment" "example-AmazonEC2ContainerRegistryReadOnly" {
79   policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
80   role       = aws_iam_role.tu_node_role.name
81 }
82
83 resource "aws_iam_role_policy_attachment" "example-CloudWatchAgentServerPolicy" {
84   policy_arn = "arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy"
85   role       = aws_iam_role.tu_node_role.name
86 }
```

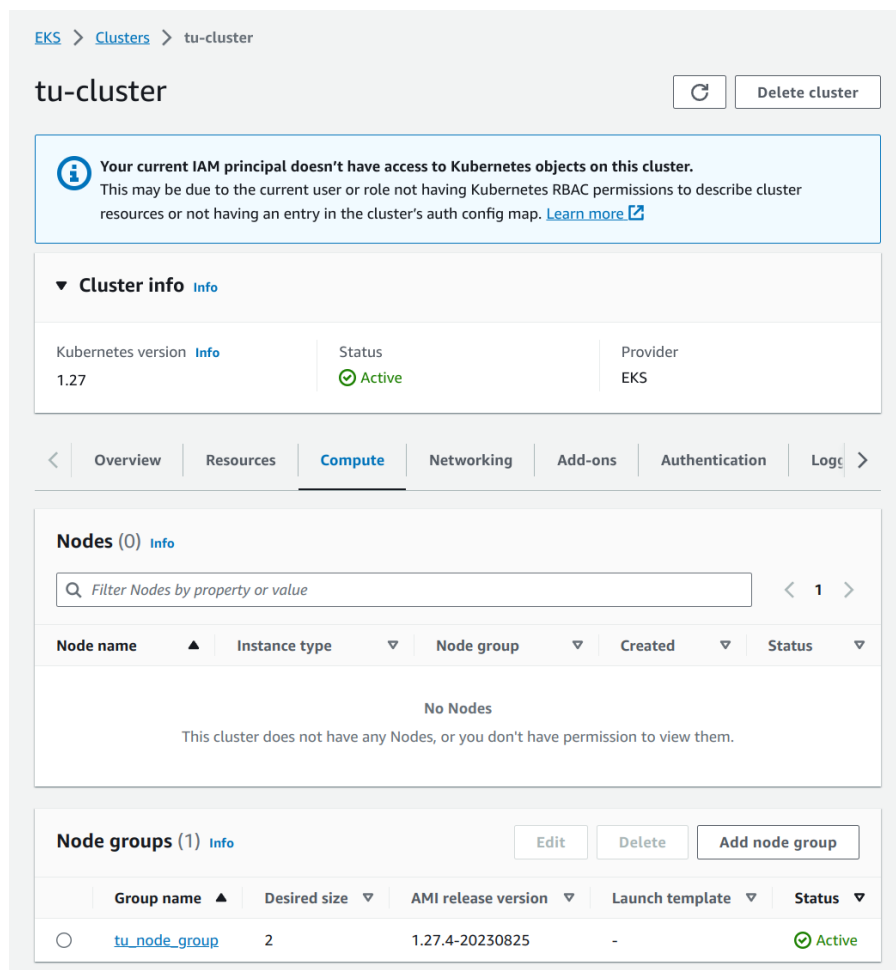
Hình 27: Terraform - policy & role cho EKS node-group

Sau khi các poli và role đã tạo xong, có thể dùng để tạo EKS node-group. Tương tự với EKS cluster, private subnet sẽ được dùng cho EKS node-group

```
87 resource "aws_eks_node_group" "tu_nodes" {
88   cluster_name = aws_eks_cluster.tu_cluster.name
89   node_group_name = var.eks-node_group_name
90   node_role_arn = aws_iam_role.tu_node_role.arn
91   subnet_ids = var.private_subnets_id
92   instance_types = [var.eks-instance_types]
93   scaling_config {
94     desired_size = 2
95     max_size = 4
96     min_size = 1
97   }
98
99   update_config {
100     max_unavailable = 1
101   }
102
103   # Ensure that IAM Role permissions are created before and deleted after EKS Node Group handling.
104   # Otherwise, EKS will not be able to properly delete EC2 Instances and Elastic Network Interfaces.
105   depends_on = [
106     aws_iam_role_policy_attachment.example-AmazonEKSWorkerNodePolicy,
107     aws_iam_role_policy_attachment.example-AmazonEKS_CNI_Policy,
108     aws_iam_role_policy_attachment.example-AmazonEC2ContainerRegistryReadOnly,
109     aws_iam_role_policy_attachment.example-CloudWatchAgentServerPolicy
110   ]
111 }
```

Hình 28: Terraform - EKS node-group

EKS cluster và EKS node-group có thể sẽ mất một khoản thời gian để tạo ra. Sau khi EKS cluster và EKS node-group được tạo thành công, giao diện người dùng của AWS sẽ tương tự như *hình 29*.



Hình 29: kết quả sau khi tạo ra K8S cluster & node-group thành công

b. AWS ELB

AWS ELB là một add-on của EKS cluster có tác dụng sẽ tạo ra một load balancer cho các node khi một Kubernetes ingress được tạo ra. Việc tạo ra AWS ELB add-on sẽ được thực hiện bằng bash scripting. Các command của script sẽ được làm theo các bước như trên document <https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html>

Trước khi cài đặt EKS ELB lên trên cluster sẽ phải cần đến một số pre-requirement như IAM OIDC provider, IAM policy, IAM service account

```
build.sh x
materials > backend > modules > EKS > scripts > build.sh
9 #####
10 # AWS K8S loadbalancer pre-requirement
11 #####
12
13 eksctl utils associate-iam-oidc-provider \
14   --region us-west-2 \
15   --cluster $CLUSTER_NAME \
16   --approve
17
18 aws iam create-policy \
19   --policy-name AWSLoadBalancerControllerIAMPolicy-$CLUSTER_NAME \
20   --policy-document file:///resources/iam-policy.json
21
22 eksctl delete iamserviceaccount --name aws-load-balancer-controller --cluster $CLUSTER_NAME
23
24 eksctl create iamserviceaccount \
25   --cluster=$CLUSTER_NAME \
26   --namespace=kube-system \
27   --name=aws-load-balancer-controller \
28   --attach-policy-arn=arn:aws:iam::492804330065:policy/AWSLoadBalancerControllerIAMPolicy-$CLUSTER_NAME \
29   --override-existing-serviceaccounts \
30   --region us-west-2 \
31   --approve
32
33 #####
```

Hình 30: bash scripting - AWS ELB pre-requirement

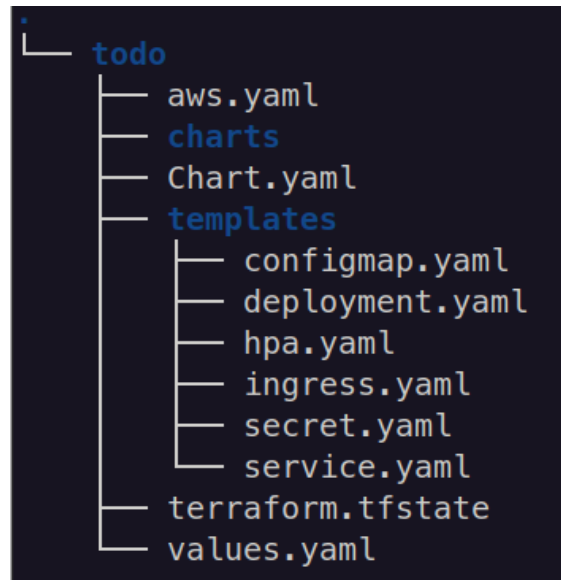
Sau khi các pre-requirement đã được đáp ứng, có thể cài đặt add-on vào trong cluster. Việc cài đặt sẽ được thực hiện thông qua Helm.

```
build.sh M x
materials > backend > modules > EKS > scripts > build.sh
33 #####
34 # AWS K8S loadbalancer
35 #####
36
37 helm repo add eks https://aws.github.io/eks-charts
38
39 helm repo update eks
40
41 kubectl apply -k "github.com/aws/eks-charts/stable/aws-load-balancer-controller/crds?ref=master"
42
43 helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
44   -n kube-system \
45   --set clusterName=$CLUSTER_NAME \
46   --set serviceAccount.create=false \
47   --set serviceAccount.name=aws-load-balancer-controller
48
49 # wait for load balancer controller deploys completely in k8s
50 sleep 60
51
```

Hình 31: Hình 28: bash scripting - AWS ELB

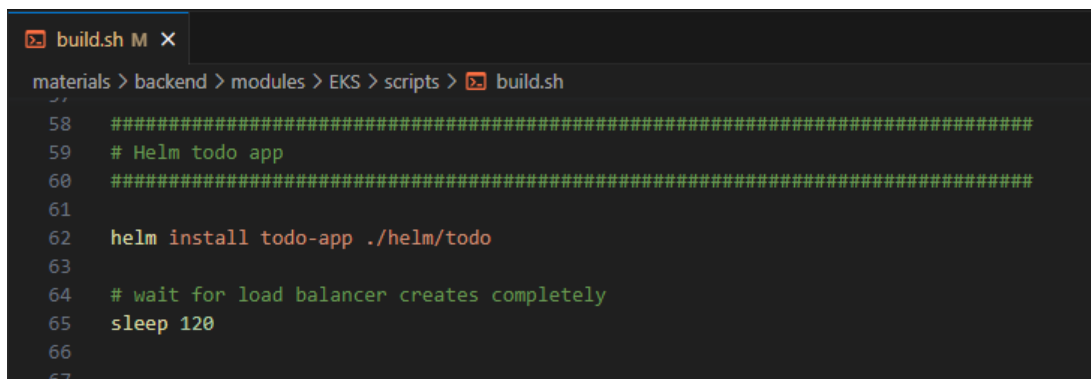
c. Helm chart – todo app

Todo-app sẽ là một Helm chart được tạo ở dưới local với template như *hình 32*.



Hình 32: Helm template

Sau khi đã viết Helm template một cách hoàn chỉnh, có thể cài đặt todo-app lên trên cluster thông qua bash scripting và Terraform `null_resource`.

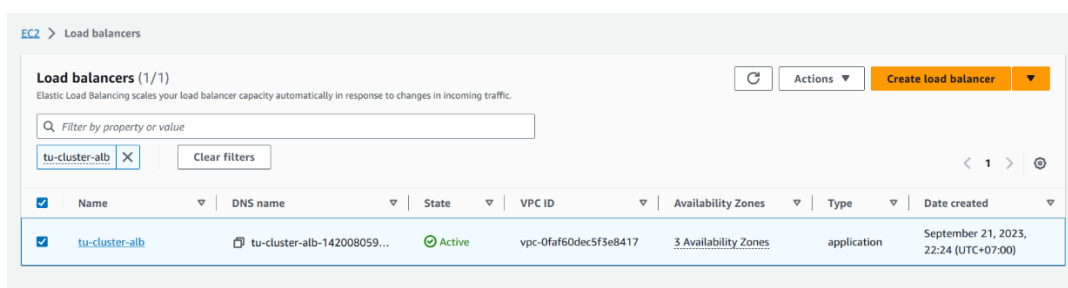


Hình 33: bash scripting - helm todo-app

Sau khi deploy thành công, kết quả sẽ tương tự như *hình 34*. Ngoài ra kết quả của phần AWS ELB cũng được thể hiện qua *hình 35* khi Kubernetes ingress đã được tạo ra cùng với todo-app

```
> kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/todo-app-8c66766c8-m5f62       1/1     Running   0           12m
pod/todo-app-8c66766c8-r2lmf       1/1     Running   0           12m
```

Hình 34: todo-app sau khi deploy thành công



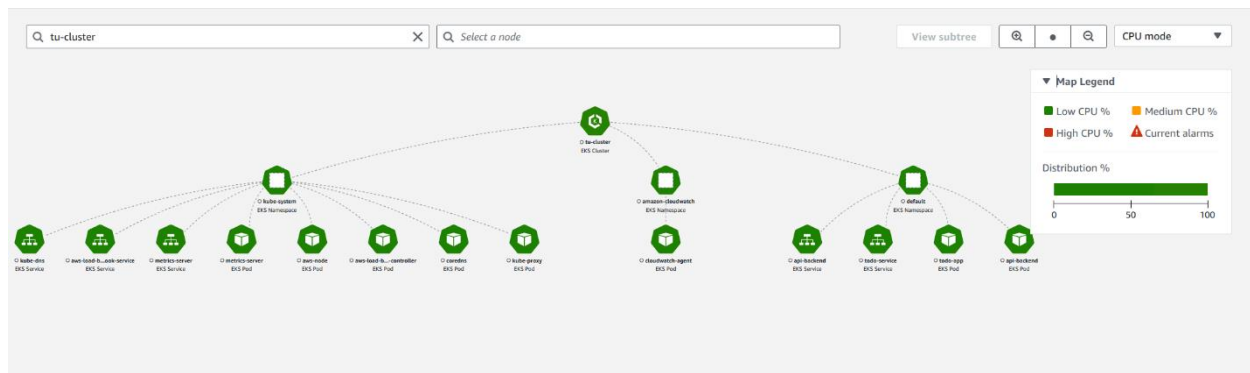
Hình 35: kết quả sau khi ingress của app được tạo thành công

d. Modnitoring

Có nhiều cách để có thể theo dõi các metrics của Kubernetes cluster có thể kể đến như: AWS Cloudwatch, Prometheus-Grafana. Trong project này, công cụ được sử dụng để theo dõi các metrics của các node trong cluster là AWS Cloudwatch.

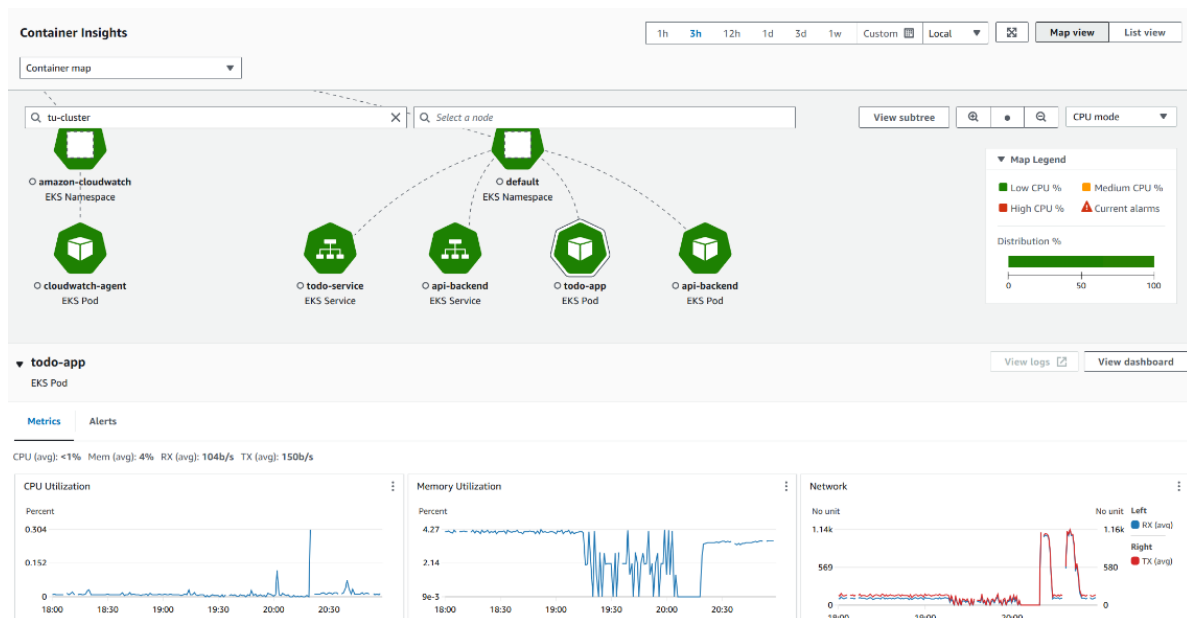
Để AWS Cloudwatch có thể thu thập các metric bên trong các node, Cloudwatch Agent phải được thêm vào trong cluster dưới dạng một application. Hướng dẫn chi tiết ở [“https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Container-Insights-setup-metrics.html”](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Container-Insights-setup-metrics.html)

Sau khi cài đặt thành công Cloudwatch Agent vào trong cluster, có thể dùng giao diện người dùng của AWS Cloudwatch để xem các metric. *Hình 36* là sơ đồ của cluster bao gồm các namespace, service, pod, ... được vẽ tự động bởi Cloudwatch.

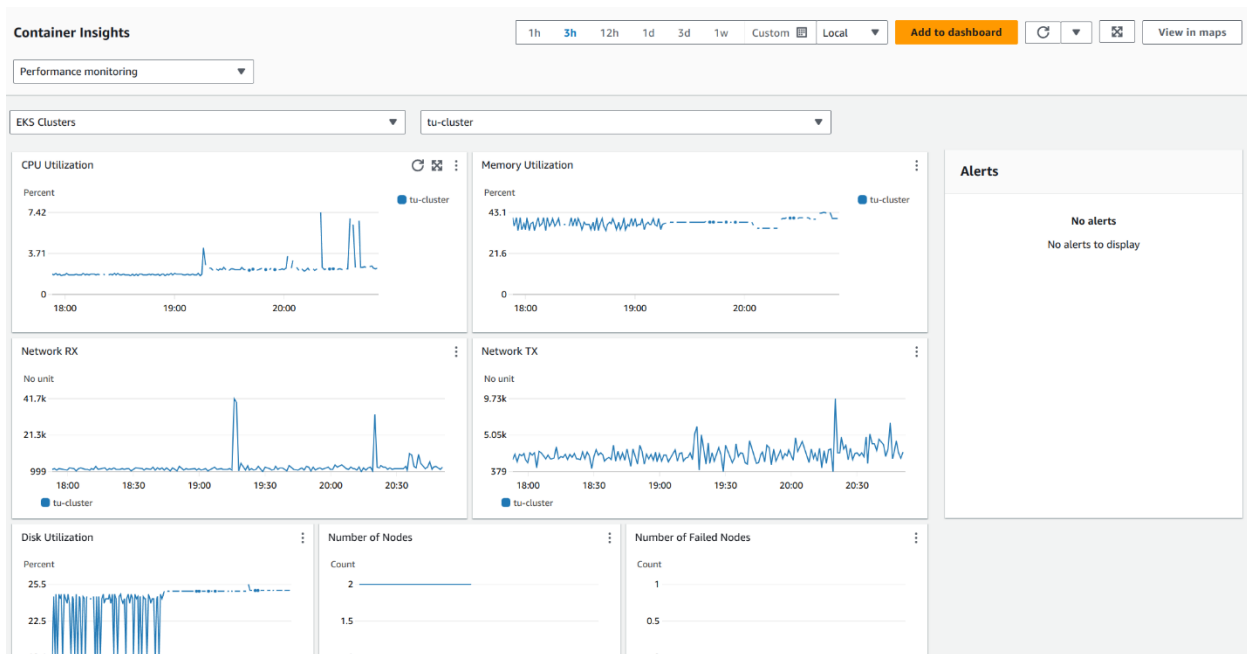


Những resource như CPU, memory có thể được nhìn thấy trên sơ đồ trên thông qua màu sắc. Có thể chuyển qua lại giữa “CPU mode” và “memory mode” để xem các tài nguyên tương ứng trên cluster, ngoài ra còn có thể xem pod hay service nào hiện đang chiếm dụng nhiều tài nguyên.

Một cách chi tiết hơn, có thể nhấn vào từng pod hay service để xem các metric dưới dạng timeline như CPU utilization, memory utilization hay network.



Ngoài ra có thể chọn mục “View dashboard” để xem các metrics của cluster và các node như *hình 38* bên dưới.



Hình 38: các metric của cluster được thể hiện qua dashboard

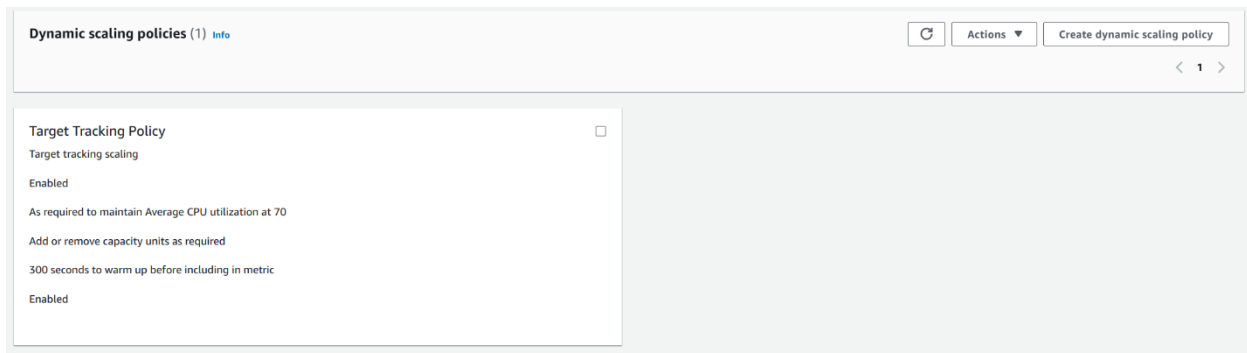
e. Auto-scaling (pod & node level)

Auto-scaling trong project này sẽ có 2 dạng là pod level và node level. Đối với pod level, auto-scaling sẽ được thực hiện thông qua Kubernetes HPA. Trong HPA có thể setup các thông số như CPU utilization, memory utilization hay number of requested packets tùy theo nhu cầu scaling dựa trên thông số nào. Sau khi HPA được tạo thành công, khi dùng lệnh “kubectl get hpa” sẽ cho ra kết quả như hình

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
horizontalpodautoscaler.autoscaling/todo-hpa	Deployment/todo-app	0%/80%	2	10	2	22h

Hình 39: HPA được deploy thành công

Đối với node level, có thể tùy chỉnh auto-scaling thông qua giao diện người dùng ở mục EC2-Auto scaling.



Hình 40: dynamic scaling policy của node

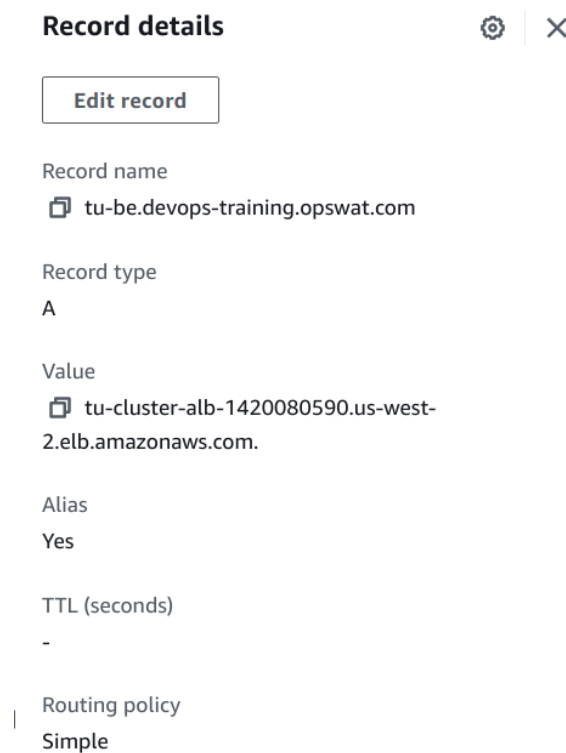
6. Route53

Phần cuối của phần backend là tạo ra một domain name kết nối tới load balancer, ở đây sẽ là “<https://tu-be.devops-training.opswat.com>”.

```
main.tf x
materials > backend > modules > route53 > main.tf > ...
1 #####
2 # Getting data
3 #####
4
5 data "aws_route53_zone" "opswat" {
6   name = "devops-training.opswat.com"
7 }
8
9 data "aws_lb" "tu-abl" {
10  name = "tu-cluster-alb"
11 }
12
13 #####
14 # Route53: DNS for backend application
15 #####
16
17 resource "aws_route53_record" "tu-be" {
18   zone_id = data.aws_route53_zone.opswat.zone_id
19   name     = "tu-be.devops-training.opswat.com"
20   type     = "A"
21   # ttl     = 300
22   allow_overwrite = true
23   # records = [data.aws_route53_zone.opswat.name]
24   alias {
25     name           = data.aws_lb.tu-abl.dns_name
26     zone_id        = data.aws_lb.tu-abl.zone_id
27     evaluate_target_health = true
28   }
29 }
```

Hình 41: Terraform – backend route53 record

Domain name này thuộc loại Simple routing, với alias trỏ thẳng về domain name của load balancer. Sau khi tạo ra domain name và liên kết với load balancer thành công, kết quả sẽ như *hình 42*.



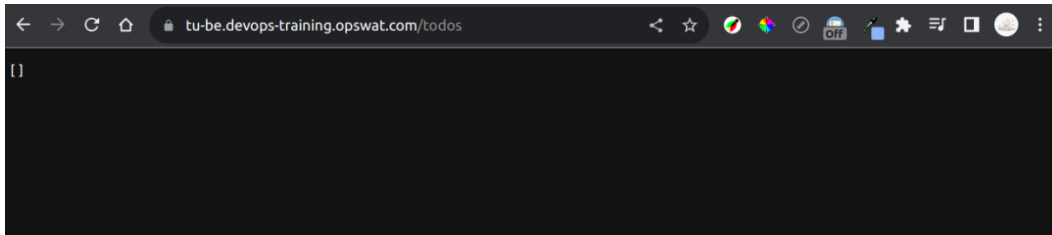
Hình 42: kết quả sau khi liên kết domain name với load-balancer thành công

Domain name này lúc này đã được request một HTTPS certificate và đã được verify thành công. Nên khi truy cập vào domain name này ở các trình duyệt sẽ thấy biểu tượng “ổ khóa” bên cạnh domain name. Nếu như todo-app được deploy thành công, kết quả khi truy cập vào domain name “<https://tu-be.devops-training.opswat.com>” sẽ như *hình 43*.



Hình 43: kết quả sau khi backend được dựng thành công

Và nếu như phần backend của todo-app có thể kết nối với database, khi truy cập vào domain name “<https://tu-be.devops-training.opswat.com/todos>” sẽ có kết quả như *hình 43*. Ngược lại nếu không thể kết nối với database, trình duyệt sẽ cố gắng kết nối một lúc lâu và trả về lỗi 503.

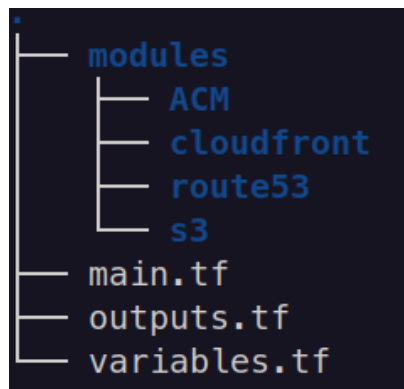


Hình 44: kết quả sau khi backend liên kết với database thành công

IV. FRONTEND

1. Tổng quan

Ở phần trước, backend của todo-app đã được dựng thành công. Tuy nhiên để app hoạt động một cách thân thiện với người dùng, phần giao diện người dùng phải được tạo nên. Giao diện người dùng hay frontend của app sẽ bao gồm thanh bar để nhập dữ liệu và nút bấm. Các dữ liệu được nhập sẽ được gửi tới phần backend, sau đó phần backend sẽ xử lý và đưa các dữ liệu đó lưu trữ ở database. Phần frontend của todo-app sẽ có cấu trúc như hình 45



Hình 45: cấu trúc của frontend-module

Frontend-module sẽ được cấu thành bởi 4 sub-module là: acm-module, cloudfront-module, route53-module, s3-module. Phần frontend của web sẽ được build thành một static web folder, sau đó folder này sẽ được upload lên trên s3 bucket. Từ đây cloudfront sẽ kết nối với S3 bucket và phân phối static web đó kèm theo HTTPS certificate. Cuối cùng sẽ tạo một domain name kết nối tới phần cloudfront để cho URL của todo-app trông có vẻ đẹp hơn.

```

main.tf x
materials > frontend > main.tf > ...
1 #####
2 # S3: static web
3 #####
4
5 module "s3" {
6   source = "../modules/s3"
7 }
8
9 #####
10 # ACM certificate
11 #####
12
13 module "acm" {
14   source = "../modules/ACM"
15 }
16
17 #####
18 # Cloudfront
19 #####
20
21 module "cloudfront" {
22   source = "../modules/cloudfront"
23   acm-acm_certificate_arn = module.acm.acm_certificate_arn_fe
24   s3_bucket_website_endpoint = module.s3.s3_bucket_website_endpoint
25 }
26
27 #####
28 # Route53
29 #####
30
31 module "route53" {
32   source = "../modules/route53"
33   route53-alias_name = module.cloudfront.cloudfront-alias_name
34   route53-hosted_zone_id = module.cloudfront.cloudfront-alias_hosted_zone_id
35 }
36

```

Hình 46: Terraform - frontend

Theo như hình 46, s3-module và acm-module chạy một cách độc lập với nhau và có thể chạy cùng một lúc. Nghĩa là các resource như S3 bucket và certificate của ACM có thể tạo ra cùng một lúc. Sau đó các thông số về bucket và HTTPS certificate sẽ được lấy tham chiếu cho quá trình tạo cloudfront. Ngoài ra certificate HTTPS này cũng sẽ được dùng để cho quá trình tạo domain name lúc sau.

2. S3 bucket

Bước đầu tiên là tạo S3 bucket để chứa static web

```
main.tf x
materials > frontend > modules > s3 > main.tf > ...
1 #####
2 # S3 bucket
3 #####
4
5 resource "aws_s3_bucket" "bucket" {
6   bucket      = var.s3-bucket_name
7   force_destroy = var.s3-force_destroy
8
9   provisioner "local-exec" {
10     when      = destroy
11     command   = "./scripts/remove_contents.sh"
12     interpreter = ["sh"]
13     working_dir = path.module
14     environment = {
15       BUCKET_ID = self.id
16     }
17   }
18 }
19
```

Hình 47: Terraform - S3 bucket

Theo như mặc định, bucket sau khi tạo ra sẽ mang tính private và không thể truy cập từ bên ngoài. Vì vậy, nếu muốn chứa static web trên S3 và truy cập trang web từ bên ngoài thông qua domain name sẽ tùy chỉnh lại một số policy liên quan đến tính public.

```
main.tf x
materials > frontend > modules > s3 > main.tf > ...
31
32 #####
33 # S3 bucket static web
34 #####
35
36 resource "aws_s3_bucket_public_access_block" "access" {
37   bucket = aws_s3_bucket.bucket.id
38
39   block_public_acls       = false
40   block_public_policy     = false
41   ignore_public_acls     = false
42   restrict_public_buckets = false
43 }
44
45 resource "aws_s3_object" "object" {
46   key          = "somekey"
47   bucket       = aws_s3_bucket.bucket.id
48   server_side_encryption = "AES256"
49 }
50
51 resource "aws_s3_bucket_website_configuration" "website" {
52   bucket = aws_s3_bucket.bucket.id
53
54   index_document {
55     suffix = "index.html"
56   }
57
58   # error_document {
59   #   key = "error.html"
60   # }
61 }
62
```



```

62
63 data "aws_iam_policy_document" "allow_access_from_another_account" {
64   statement {
65     principals {
66       type       = "AWS"
67       identifiers = ["*"]
68     }
69
70     actions = [
71       "s3:GetObject",
72       # "s3:ListBucket",
73     ]
74
75     resources = [
76       aws_s3_bucket.bucket.arn,
77       "${aws_s3_bucket.bucket.arn}/*",
78     ]
79   }
80 }
81
82 resource "aws_s3_bucket_policy" "allow_access_from_another_account" {
83   bucket       = aws_s3_bucket.bucket.id
84   policy        = data.aws_iam_policy_document.allow_access_from_another_account.json
85   depends_on   = [aws_s3_bucket_public_access_block.access]
86 }

```

Hình 48: Terraform - S3 bucket static web configuration

Sau khi đã set up mọi thứ, bước tiếp theo sẽ là upload thư mục static web lên trên s3. Để tăng tính đơn giản, việc upload sẽ được thực hiện thông qua bash scripting và Terraform `null_resource`.

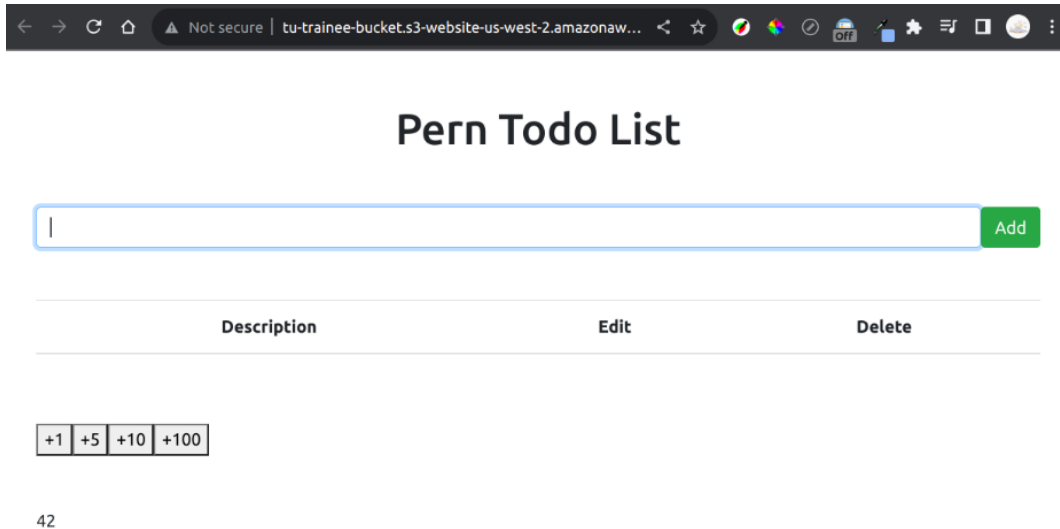
```

main.tf x
materials > frontend > modules > s3 > main.tf > data "aws_iam_policy_document" "allow_access_from_another_account"
88 #####
89 # S3 bucket upload
90 #####
91
92 resource "null_resource" "remove_and_upload_to_s3" {
93   provisioner "local-exec" {
94     command     = "aws s3 sync ./build/ s3://${aws_s3_bucket.bucket.id}"
95     working_dir = path.module
96   }
97 }

```

Hình 49: S3 bucket upload

Hình 50 cho thấy static web đã hoạt động khi truy cập vào website endpoint của bucket. Tuy nhiên rằng việc truy cập static web thông qua bucket sẽ không kèm theo HTTPS certificate. Vì vậy biểu tượng “ổ khóa” sẽ không xuất hiện bên cạnh domain name, thay vào đó là dòng chữ “Not secure”



Hình 50: S3 static web

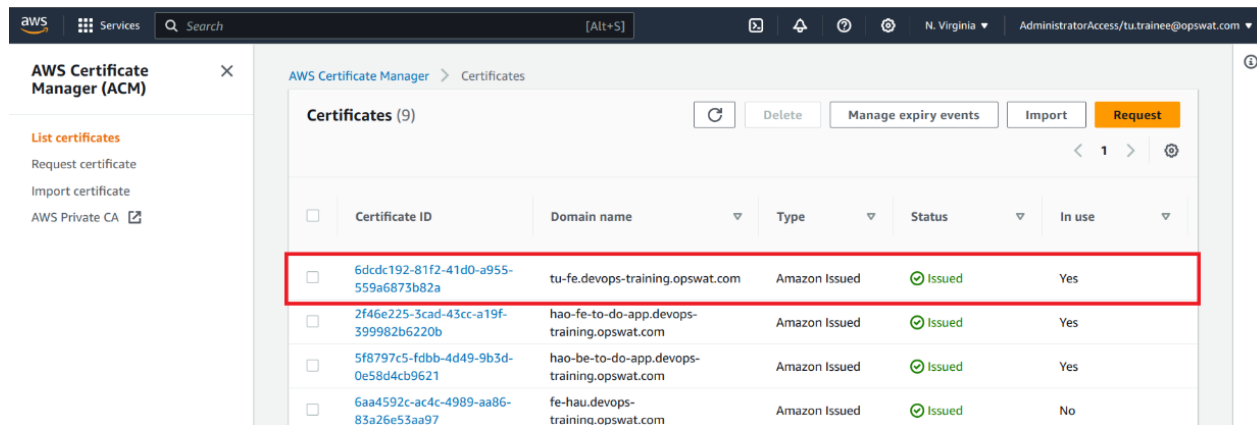
3. ACM

Trong acm-module, HTTPS certificate cho domain name “<https://tu-fe.devops-training.opswat.com>” sẽ được request. Certificate này sẽ ở dưới dạng public certificate. Sau khi certificate chuyển sang trạng thái verified, nó sẽ được dùng để tạo cloudfront và dùng để tạo HTTPS certificate và được gắn cho domain name “<https://tu-fe.devops-training.opswat.com>” khi được tạo ra bằng dịch vụ AWS Route 53 sau này.

```
main.tf x
materials > frontend > modules > ACM > main.tf > resource "null_resource" "delay" > triggers
1 #####
2 # ACM certificate
3 #####
4
5 resource "aws_acm_certificate" "tu-fe" {
6     provider      = aws.virginia
7     domain_name    = var.acm-domain_name
8     validation_method = var.acm-validation_method
9
10    lifecycle {
11        create_before_destroy = true
12    }
13 }
14
15 resource "aws_acm_certificate_validation" "tu-fe" {
16     provider      = aws.virginia
17     certificate_arn = aws_acm_certificate.tu-fe.arn
18     depends_on     = [aws_acm_certificate.tu-fe]
19 }
20
```

Hình 51: Terraform - frontend certificate

Một lưu ý rằng nếu certificate được dùng cho cloudfront thì nên được request ở Virginia region. Khi certificate được request và verify thành công sẽ có kết quả như hình 52.



Hình 52: kết quả sau khi verify thành công HTTPS certificate

4. Cloudfront

Bước tiếp theo là tạo Cloudfront liên kết với static web của S3 bucket. Do Cloudfront khi tạo ra sẽ được gắn với HTTPS certificate đã được request trước đó. Nên khi truy cập vào domain name được tạo ra bởi Cloudfront sẽ thấy được biểu tượng ổ khóa bên cạnh domain name

```
main.tf
materials > frontend > modules > cloudfront > main.tf > ...
19 #####
20 # Cloudfront
21 #####
22
23 resource "aws_cloudfront_distribution" "s3_distribution" {
24   origin {
25     domain_name = var.s3_bucket_website_endpoint
26     origin_id   = local.s3_origin_id
27     custom_origin_config {
28       origin_protocol_policy = "http-only"
29       http_port              = "80"
30       https_port             = "443"
31       # "If the origin is an Amazon S3 bucket, CloudFront always uses TLSv1.2."
32       origin_ssl_protocols = ["TLSv1.2"]
33     }
34   }
35 }
```

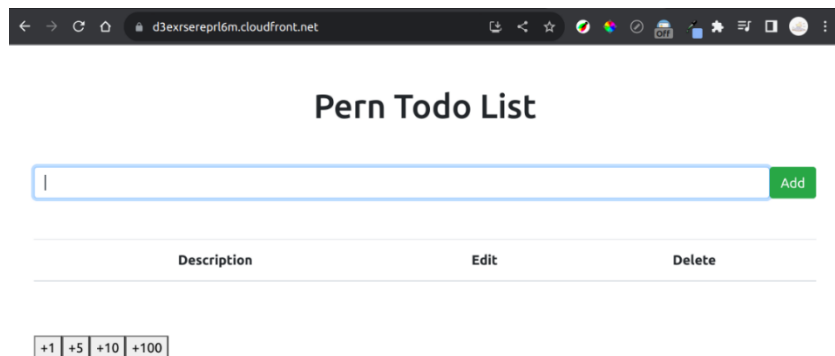
```

36   enabled      = true
37   is_ipv6_enabled = true
38
39   default_cache_behavior {
40     allowed_methods = ["DELETE", "GET", "HEAD", "OPTIONS", "PATCH", "POST", "PUT"]
41     cached_methods  = ["GET", "HEAD"]
42     target_origin_id = local.s3_origin_id
43
44     forwarded_values {
45       query_string = false
46       cookies {
47         forward = "none"
48       }
49     }
50
51     viewer_protocol_policy = "allow-all"
52     min_ttl                = 0
53     default_ttl            = 3600
54     max_ttl               = 86400
55   }
56
57   price_class = "PriceClass_All"
58
59   restrictions {
60     geo_restriction {
61       restriction_type = "none"
62       locations        = []
63     }
64   }
65
66   viewer_certificate {
67     cloudfront_default_certificate = true
68     acm_certificate_arn           = var.acm-acm_certificate_arn
69     # acm_certificate_arn         = data.aws_acm_certificate.tu-fe.arn
70     ssl_support_method            = "sni-only"
71   }
72
73   aliases = ["tu-fe.devops-training.opswat.com"]
74 }
75
76

```

Hình 53: Terraform – cloudfront

Hình 54 cho thấy Cloudfront đã được tạo thành công. Ngoài ra biểu tượng “ổ khóa” bên cạnh domain name cho thấy certificate đã được gắn thành công vào Cloudfront



42

Hình 54: Cloudfront static web

5. Route53

Bước cuối cùng của phần frontend là tạo ra một domain name “<https://tu-fe.devops-training.opswat.com>” để kết nối với Cloudfront.

```
main.tf M X
materials > frontend > modules > route53 > main.tf > ...
1 #####
2 # Route 53: DNS for frontend application
3 #####
4
5 data "aws_route53_zone" "opswat" {
6   name = "devops-training.opswat.com"
7 }
8
9 resource "aws_route53_record" "tu-fe" {
10   zone_id = data.aws_route53_zone.opswat.zone_id
11   name    = "tu-fe.devops-training.opswat.com"
12   type    = "A"
13   # ttl    = 300
14   allow_overwrite = true
15   # records = [data.aws_route53_zone.opswat.name]
16   alias {
17     name           = var.route53-alias_name
18     zone_id        = var.route53-hosted_zone_id
19     evaluate_target_health = true
20   }
21 }
22
23
```

Hình 55: Terraform – frontend route53 record

Tương tự như ở phần backend, sau khi domain name được tạo ra và liên kết với Cloudfront thành công, kết quả sẽ như hình 56

Record details ⚙️ ✕

Edit record

Record name
📄 tu-fe.devops-training.opswat.com

Record type
A

Value
📄 d3exrsereprl6m.cloudfront.net.

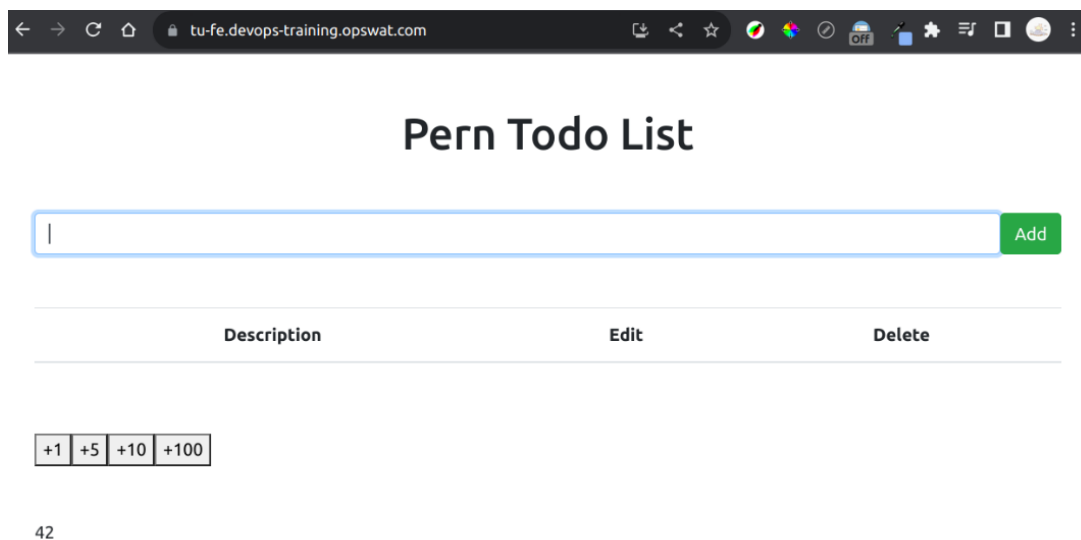
Alias
Yes

TTL (seconds)
-

Routing policy
Simple

Hình 56: kết quả sau khi liên kết domain name với cloudfornt thành công

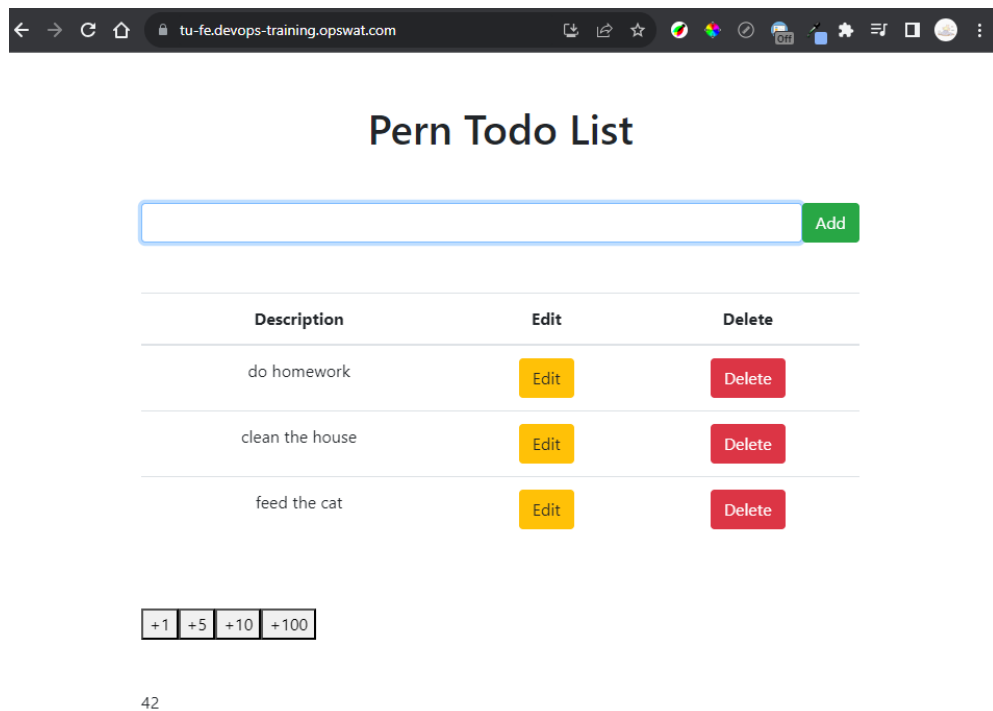
Dù chức năng add dữ liệu có thực hiện được hay không, nếu khi truy cập vào domain name “<https://tu-fe.devops-training.opswat.com>” và trình duyệt trả về giao diện người dùng như hình 57. thì phần frontend đã được dựng thành công.



Hình 57: kết quả sau khi frontend được dựng thành công

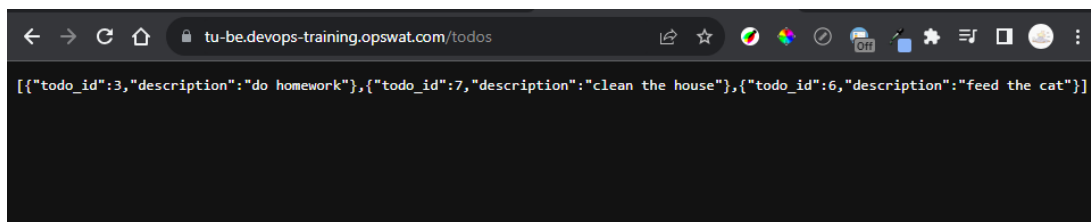
V. KẾT QUẢ

Sau khi 3 module đã tạo xong các resource, bây giờ là lúc kiểm tra xem 3 phần frontend, backend, database của app có liên kết với nhau hay không. Việc kiểm tra sẽ được thực hiện thông qua chức năng add của app. Sau khi nhập dữ liệu vào thanh bar và nhấn nút “Add” hoặc nhấn “Enter” thông qua bàn phím, nếu kết quả trả về tương tự *hình 58* thì app đã được liên kết thành công.



Hình 58: kết quả cho thấy app đã hoạt động thành công

Ngoài ra, nếu truy cập vào “<https://tu-be.devops-training.opswat.com/todos>”, trình duyệt sẽ trả về API như *hình 59*

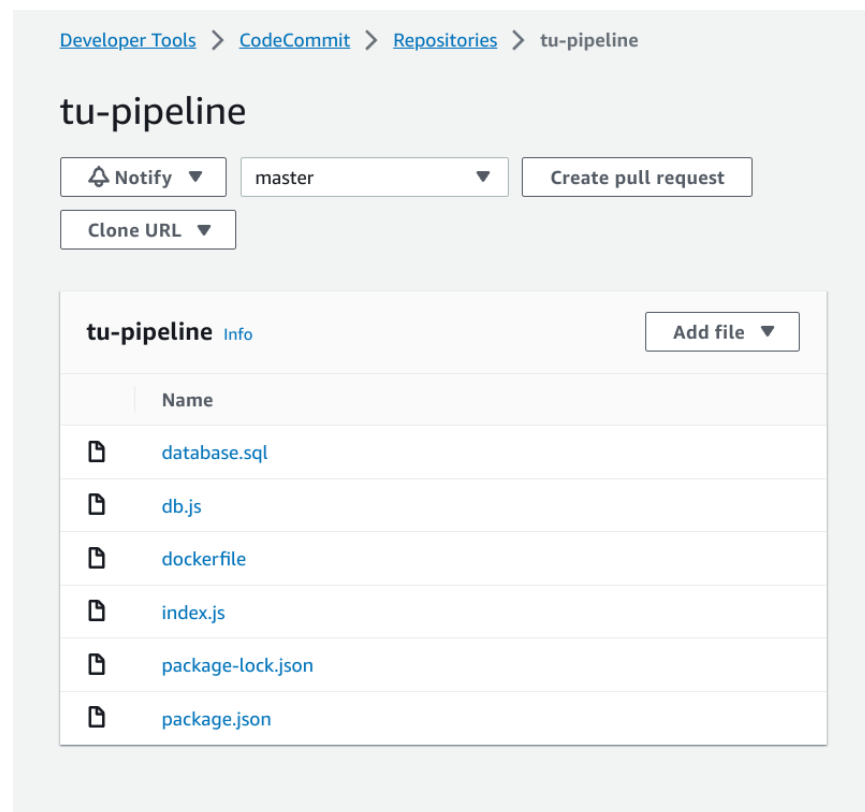


Hình 59: todo-app database API

VI. PIPELINE

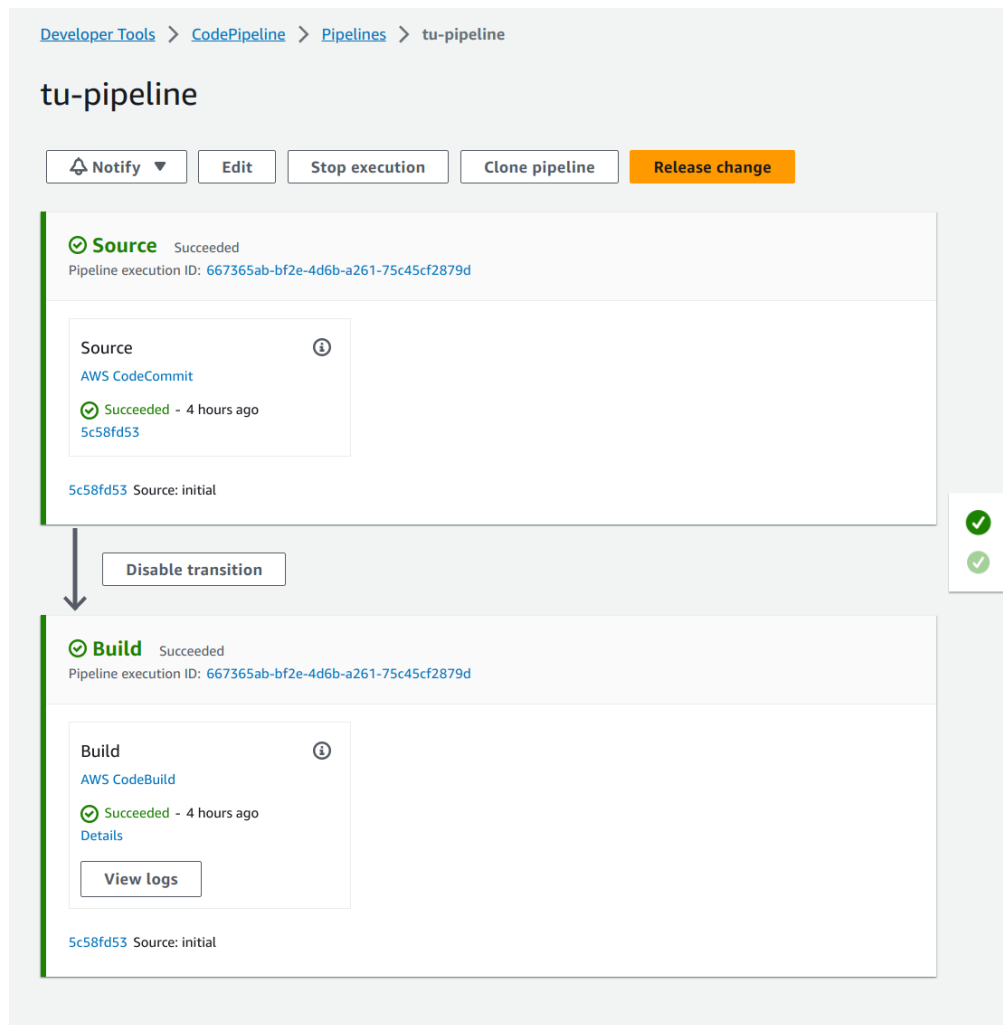
Để có thể modify phần backend của app sau khi đã deploy, AWS Codepipeline và Helm sẽ được sử dụng trong project này. Trong đó AWS Codepipeline sẽ đóng vai trò trong việc update version của docker image, còn Helm sẽ đóng vai trò phân phối image đó sang môi trường production.

Đầu tiên sẽ là tạo một repository trên AWS Codecommit để chứa phần backend cùng với dockerfile.



Hình 60: phần backend của todo-app trên AWS codecommit

Bước tiếp theo là tạo AWS Codepipeline, trong đó pipeline này sẽ liên kết thẳng đến AWS Codecommit, để khi nếu có thay đổi gì ở trong repo, pipeline sẽ được trigger một cách tự động và build phần backend thành docker image và upload lên trên ECR kèm theo version.



Hình 61: AWS Codepipeline sau khi chạy thành công

Sau khi đã update version của Docker image, tiếp theo sẽ deploy lên production image vừa tạo bằng Helm, đồng thời tăng version của todo-app chart lên 1 version sau khi deploy.

```
> helm upgrade todo-app .
Release "todo-app" has been upgraded. Happy Helming!
NAME: todo-app
LAST DEPLOYED: Fri Sep 22 03:17:46 2023
NAMESPACE: default
STATUS: deployed
REVISION: 4
TEST SUITE: None
```

Hình 62: deploy helm chart thành công

VII. TỔNG KẾT

Nhìn chung, đây làm một app đơn giản với các feature không quá phức tạp, tuy nhiên project này đã mang đến nhiều kiến thức có thể kể đến như:

- Deploy một application lên trên môi trường production.
- Hiểu biết thêm một lượng đáng kể các dịch vụ của AWS được sử dụng phổ biến
- Dùng Terraform để quản lý các resource cung cấp bởi AWS
- Các ứng dụng thực tiễn của Kubernetes, Docker và Helm trong việc đóng gói và quản lý các application trên môi trường production.
- Cách kiến thức khác liên quan đến bash scripting và security

VII. LỜI CẢM ƠN

Để project này được hoàn thành một cách suôn sẻ không thể không kể đến sự giúp đỡ của những anh lớn trong OPSWAT đã hướng dẫn, giảng dạy, giải đáp thắc mắc cùng với đưa đến cho khóa trainee một cách dễ hiểu nhất những kiến thức bổ ích cho công việc sau này. Lời cảm ơn đầu tiên xin được gửi đến những anh lớn trong OPSWAT. Ngoài ra xin được gửi thêm một lời cảm ơn đặc biệt nữa đến anh Quang Phuong.

Ngoài ra cũng không thể không gửi lời cảm ơn đến những bạn trainee đã cùng giúp đỡ nhau để hoàn thành project này.

Xin gửi lời cảm ơn đến tất cả.

Hồ Chí Minh ngày 21 tháng 9 năm 2023