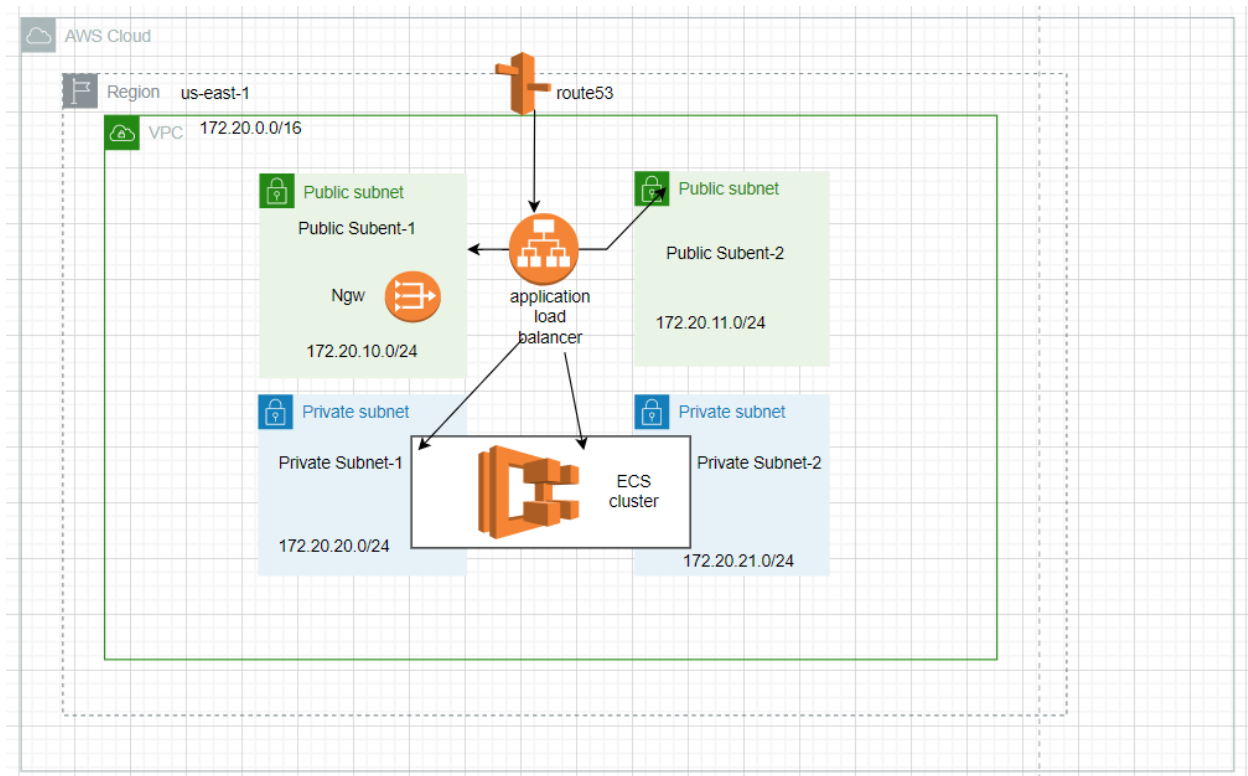


Container Project on AWS

Architecture:



> To Create This Infrastructure we are using Terraform (Infrastructure as code) Tool.

Prerequisite steps:

1. Make sure you have an AWS account.
2. Create aws linux instance(t2.micro), You also need an ssh key to connect to the EC2 instance that you are creating.
3. Connect to the instance using ssh key, Then install the AWS CLI

AWS CLI: The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.

For installation guidance for aws cli go through below link

[Installing or updating the latest version of the AWS CLI - AWS Command Line Interface \(amazon.com\)](#)



Confirm the installation by executing the following command.

```
[ec2-user@ip-172-31-26-25 ~]$ aws --version
aws-cli/1.18.147 Python/2.7.18 Linux/5.10.102-99.amzn2.x86_64 botocore/1.18.6
[ec2-user@ip-172-31-26-25 ~]$ |
```

Configure the AWS CLI :

> After Successful installation of aws cli, you have to configure the settings that AWS CLI uses to interact with the AWS.

For general purpose, The aws configure command is the fastest way to set up your credentials. When you enter this command AWS cli prompts you for four pieces of information.

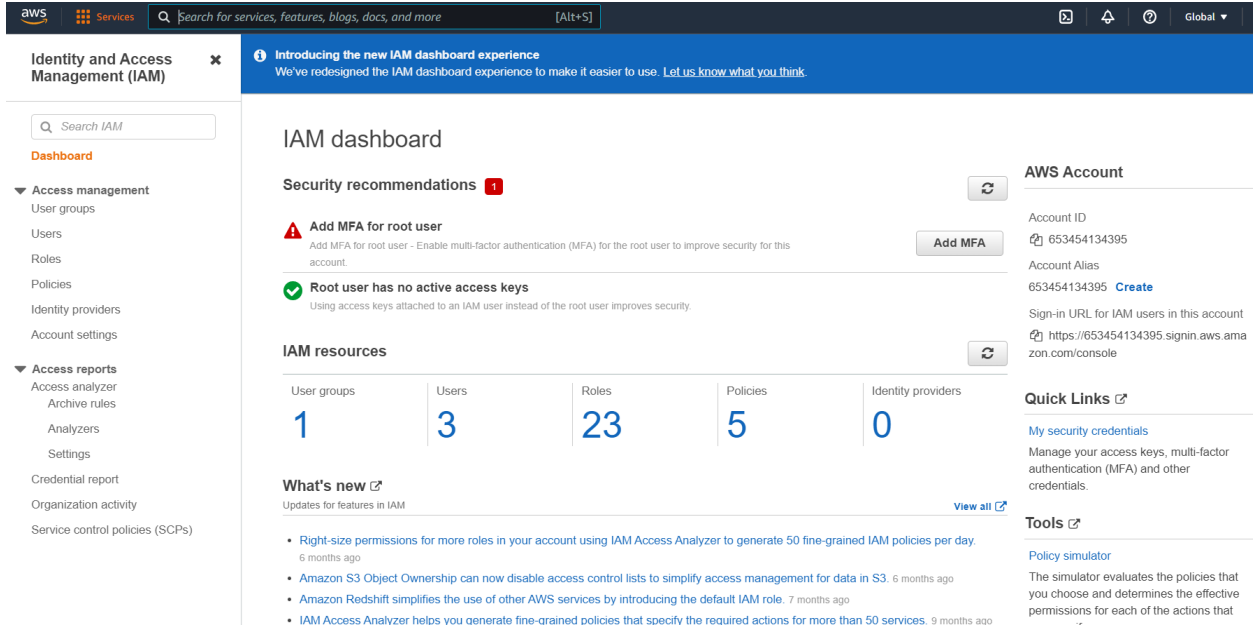
- 1) Access Key ID
- 2) Secret Access Key
- 3) AWS Region
- 4) Output Format

Create IAM user:

> An IAM User is an entity created in AWS that provides a way to interact with AWS resources.

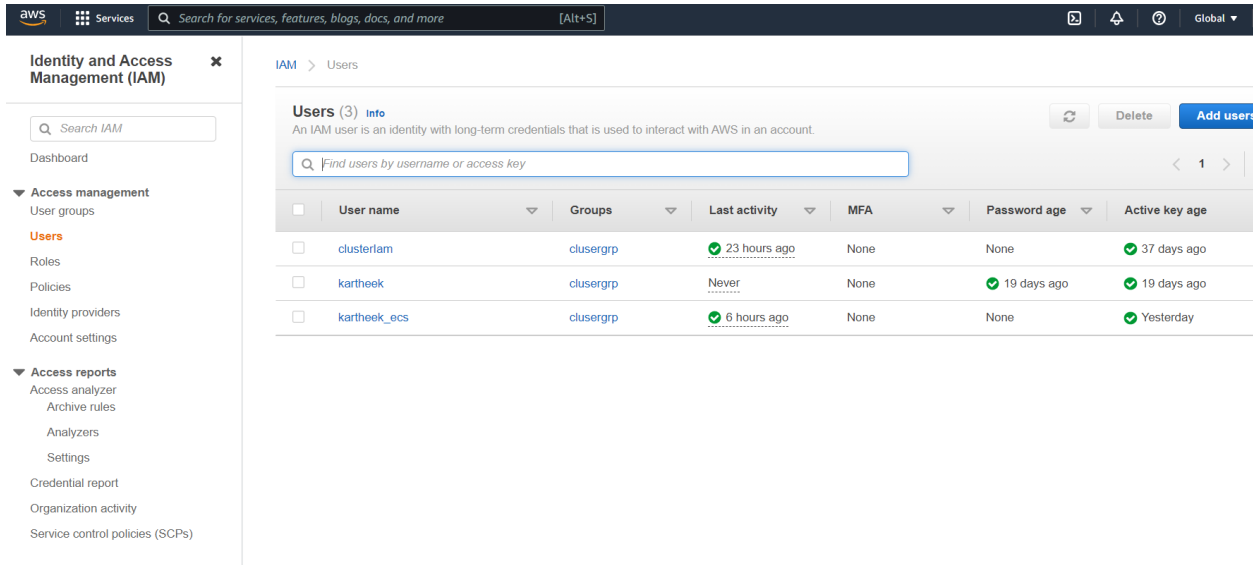
> To create an IAM user, Sign into the AWS management console.

> Open the IAM console, The screen appears which is shown below.



The screenshot shows the AWS IAM dashboard. The left navigation pane includes sections for Access management (User groups, Users, Roles, Policies, Identity providers, Account settings), Access reports (Access analyzer, Archive rules, Analyzers, Settings, Credential report, Organization activity, Service control policies (SCPs)), and a search bar. The main content area displays the IAM dashboard with a security recommendations section (Add MFA for root user, Root user has no active access keys), IAM resources (User groups: 1, Users: 3, Roles: 23, Policies: 5, Identity providers: 0), and a 'What's new' section with updates for features in IAM. The right sidebar shows the AWS Account information (Account ID, Account Alias, Sign-in URL) and Quick Links (My security credentials, Tools).

> On the navigation pane, click on the Users. After clicking on the Users, the screen appears which is shown below



The screenshot shows the AWS IAM Users page. The left navigation pane is the same as the previous screenshot, but the 'Users' link under 'Access management' is highlighted. The main content area displays the 'Users (3)' page with a search bar and a table of users. The table has columns for User name, Groups, Last activity, MFA, Password age, and Active key age. The users listed are clusteriam, kartheek, and kartheek_ecs.

	User name	Groups	Last activity	MFA	Password age	Active key age
<input type="checkbox"/>	clusteriam	clusergrp	23 hours ago	None	None	37 days ago
<input type="checkbox"/>	kartheek	clusergrp	Never	None	19 days ago	19 days ago
<input type="checkbox"/>	kartheek_ecs	clusergrp	6 hours ago	None	None	Yesterday

> Click on the Add User to add new users to your account. After clicking on the Add User, the screen appears which is shown below:



Add user



Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

[+ Add another user](#)

Select AWS access type

Select how these users will primarily access AWS. If you choose only programmatic access, it does NOT prevent users from accessing the console using an assumed role. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

- Select AWS credential type*
- ☐ **Access key - Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
 - ☐ **Password - AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

* Required

[Cancel](#)

[Next: Permissions](#)

> Enter the user name as you required, and select the programmatic access then click next.

> You have to give the permission to the user, After creating success it will prompt the screen as below.


Add user

1 2 3 4 5

✓ Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://653454134395.signin.aws.amazon.com/console>

 Download .csv

	User	Access key ID	Secret access key
▶ ✓	any	AKIAZQJHGZR5R4TEKPH5 	***** Show

Close

> Download the csv file and save it in a safe folder.

> Now you have to run the **aws configure** command, it will ask you for the secret key and access key id.

```
ec2-user@ip-172-31-26-25 ~]$ aws configure
AWS Access Key ID [*****BMAR]:
AWS Secret Access Key [*****rHv4]:
default region name [us-region-1]:
default output format [None]: |
```

Creating s3 Bucket:

> We are using an s3 bucket to store the state files.

> **Statefile:** Terraform stores information about your infrastructure in a state file. This state file keeps track of resources created by your configuration and maps them to real-world resources

> To create the s3 bucket open the s3 service in aws console.

- a) open the s3 service in aws console.
- b) Choose Create bucket. The Create bucket wizard opens.
- c) In Bucket name, enter a DNS-compliant name for your bucket.

The bucket name must:

- Be unique across all of Amazon S3.
- Be between 3 and 63 characters long.
- Please do not contain uppercase characters.
- Start with a lowercase letter or number.

NOTE: After you create the bucket, you cannot change its name

The screenshot displays the Amazon S3 console interface. On the left, the 'Amazon S3' sidebar is visible with a search bar and a list of navigation items including 'Buckets', 'Access Points', 'Object Lambda Access Points', 'Multi-Region Access Points', 'Batch Operations', 'Access analyzer for S3', 'Block Public Access settings for this account', 'Storage Lens', 'Dashboards', 'AWS Organizations settings', 'Feature spotlight', and 'AWS Marketplace for S3'. The main panel shows the 'kartheeks-bucket' page. At the top, there's a breadcrumb trail 'Amazon S3 > Buckets > kartheeks-bucket' and a title 'kartheeks-bucket' with an 'info' link. Below this are tabs for 'Objects', 'Properties', 'Permissions', 'Metrics', 'Management', and 'Access Points'. The 'Objects' tab is selected, displaying a section titled 'Objects (1)'. A descriptive text explains that objects are fundamental entities stored in Amazon S3 and provides a link to 'Amazon S3 inventory' for a list of all objects. Below the text are several action buttons: 'Refresh', 'Copy S3 URI', 'Copy URL', 'Download', 'Open', 'Delete', 'Actions', 'Create folder', and 'Upload'. A search bar labeled 'Find objects by prefix' is also present. At the bottom, a table lists the objects with columns for 'Name', 'Type', 'Last modified', 'Size', and 'Storage class'. One object is listed: 'terraform_state/' with a folder icon, type 'Folder', and dashes for the other fields.

Create route53 zone:

A public hosted zone is a container that holds information about how you want to route traffic on the internet for a specific domain, such as example.com, and its subdomains (acme.example.com, zenith.example.com). After you create a hosted zone, you create records that specify how you want to route traffic for the domain and subdomains.

Create hosted zone:

> Goto the route53 service in aws management console. And click on hosted zones in the left side panel. It leads you to a dashboard like below.

The screenshot displays the AWS Route 53 'Hosted zones' dashboard. On the left, a navigation sidebar lists various services, with 'Hosted zones' highlighted. The main content area shows a table of hosted zones. A single zone is listed with the domain 'kartheek123.online', type 'Public', created by 'Route 53', and containing 4 records. Above the table, there are buttons for 'View details', 'Edit', 'Delete', and 'Create hosted zone'. A search bar is also present above the table.

Domain name	Type	Created by	Record count	Description	Hosted zone ID
kartheek123.online	Public	Route 53	4	-	Z03840633HHYSQJYWHJ9

> Click on Create Hosted Zones button to create a new hosted zone. Write the name of the domain in the Domain name field and add some description in the description field. description field is an optional field. Type leave as default. Before pressing the create button check the name of the domain. Finally press the create button. For example I am adding 'testapp.com' as a domain name and adding 'test App ' comment in the

comment field.

Route 53 > Hosted zones > Create hosted zone

Create hosted zone Info

Hosted zone configuration

A hosted zone is a container that holds information about how you want to route traffic for a domain, such as example.com, and its subdomains.

Domain name Info

This is the name of the domain that you want to route traffic for.

Valid characters: a-z, 0-9, ! " # \$ % & ' () * + , - . / : ; < = > ? @ [\] ^ _ ` { | } . ~

Description - optional Info

This value lets you distinguish hosted zones that have the same name.

The description can have up to 256 characters. 8/256

Type Info

The type indicates whether you want to route traffic on the internet or in an Amazon VPC.

☒ **Public hosted zone**
 A public hosted zone determines how traffic is routed on the internet.

☐ **Private hosted zone**
 A private hosted zone determines how traffic is routed within an Amazon VPC.

> You can see that the dashboard shows a new item in the Hosted Zone. Select the particular domain and click on it. You can see 2 record sets have been already created. One is the NS type and the other is SOA type. From here you can create the hosts of your domain. For example – www.testapp.com, docs.testapp.com, www.docs.testapp.com, blog.testapp.com, www.blog.testapp.com etc.

Creating Certificate:

Aws certificate manager: AWS Certificate Manager is a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with AWS services and your internal connected resources. SSL/TLS certificates are used to secure network communications and establish the identity of websites over the Internet as well as resources on private networks. AWS Certificate Manager removes the time-consuming manual process of purchasing, uploading, and renewing SSL/TLS certificates.

Create aws certificates:

> To create an aws certificate goto the aws management console and goto **certificate management** service.

> Choose Request a certificate to request a new certificate.

The screenshot shows the AWS Certificate Manager console. On the left, the 'AWS Certificate Manager' sidebar is visible with options: 'List certificates', 'Request certificate', 'Import certificate', and 'Private CA'. The main content area has a dark blue header with 'SECURITY, IDENTITY, & COMPLIANCE' and 'AWS Certificate Manager'. Below this, it states 'AWS Certificate Manager (ACM) makes it easy to provision, manage, deploy, and renew SSL/TLS certificates.' On the right, there's a 'New ACM managed certificate' section with three options: 'Request a certificate' (highlighted in orange), 'Import a certificate', and 'Create a private CA'. A 'Pricing (US)' section is partially visible at the bottom right.

> Provide your **domain name**, and choose the dns validation method. Then click on next

The screenshot shows the 'Request a certificate' workflow in the AWS Certificate Manager console. The left sidebar is the same as the previous screenshot, but 'Request certificate' is now highlighted in orange. The main content area is titled 'Domain names' and shows a 'Fully qualified domain name' input field with 'demo.xyz' entered. Below this is a button 'Add another name to this certificate'. A note explains that additional names can be added for different subdomains. The next section is 'Select validation method', where 'DNS validation - recommended' is selected with a radio button. Below this, there's a section for 'Tags' with input fields for 'Tag key' and 'Tag value - optional', and a button 'Add tag'. A note at the bottom states 'You can add 49 more tag(s)'.

> Once the certificate is created, open it and click on **Create records in route53**. then it automatically creates a **CNAME** record in the Route53 hosted zone.

AWS Certificate Manager

- List certificates
- Request certificate
- Import certificate
- Private CA

Certificate status

Identifier 93780fae-f240-4bf2-9143-8ce857919b7f	Status ⌚ Pending validation
ARN arn:aws:acm:us-east-1:653454134395:certificate/93780fae-f240-4bf2-9143-8ce857919b7f	Detailed status The status of this certificate request is "Pending validation". Further action is needed to validate and approve the certificate. Info
Type Amazon Issued	

Domains (1) [Create records in Route 53](#) [Export to CSV](#)

Domain	Status	Renewal status	Type	CNAME name	CNAME value
demo.xyz	⌚ Pending validation	-			

Details

In use?	Serial number	Requested at	Renewal eligibility
No	N/A	June 02, 2022, 09:06:02 (UTC+05:30)	Ineligible

> Refresh, and once the validation is completed the status of the certificate will become issued.

AWS Certificate Manager

- List certificates
- Request certificate
- Import certificate
- Private CA

AWS Certificate Manager > Certificates

Certificates (1) [Refresh](#) [Delete](#) [Manage expiry events](#) [Import](#) [Request](#)

<input type="checkbox"/>	Certificate ID	Domain name	Type	Status	In use?	Renewal eligibility
<input type="checkbox"/>	a5345179-696c-4b5e-be2f-edbaf00e876a	kartheek123.online	Amazon Issued	✅ Issued	Yes	Eligible

Install Terraform:

Terraform: HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle. Terraform can manage



low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

> To create all these resources we are Using terraform as an infrastructure as a code tool.

> To run the code first we need to install the terraform in our instance which we created earlier.

> To install terraform follow this link.

<https://learn.hashicorp.com/tutorials/terraform/install-cli>

> Check installation by executing **terraform --version** command. As shown below.

```
[ec2-user@ip-172-31-26-25 ~]$ terraform --version
Terraform v1.2.1
on linux_amd64
[ec2-user@ip-172-31-26-25 ~]$ |
```

Install Docker:

docker: Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

> Install the docker on the same machine which we created earlier

> To install docker follow the below link

<https://docs.docker.com/desktop/linux/install/>

> Check installation by executing **docker --version** command, as shown below

```
[ec2-user@ip-172-31-26-25 ~]$ docker --version
Docker version 20.10.13, build a224086
[ec2-user@ip-172-31-26-25 ~]$ |
```

Clone repos from Github:

> we have already developed code in the Github repository.

> Do SSH into the instance and create the directory called **ecs**, using mkdir command.

> Go inside that directory and clone the repos from the Github, link as provided below

> clone the repos from github.

Github link: <https://github.com/KarthEEK39>

git clone https://github.com/KarthEEK39/ECS_Fargate-Cluster.git



git clone https://github.com/Kartheek39/ECS_Fargate_Service.git

> Once cloning successfully the folders will look like this

```
[ec2-user@ip-172-31-26-25 ecs]$ ls
ECS_Fargate-Cluster  ECS_Fargate_Service
[ec2-user@ip-172-31-26-25 ecs]$ |
```

Creating ECS Fargate Cluster:

ECS Fargate: AWS Fargate is a technology that you can use with Amazon ECS to run [containers](#) without having to manage servers or clusters of Amazon EC2 instances. With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

When you run your tasks and services with the Fargate launch type, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

Creating ecs fargate:

> Go inside the ecs directory.

/home/ec2-user/ecs

> Then go inside the ECS_Fargate-Cluster folder. And do **ls** you can see below files

```
[ec2-user@ip-172-31-26-25 ECS_Fargate-Cluster]$ ls
alb.tf  ecr.tf  ecs.tf  environment  Fargate_cluster_arch..png  iam.tf  main.tf  network.tf  output.tf  route53.tf  security.tf  variable.tf  vpc.tf
[ec2-user@ip-172-31-26-25 ECS_Fargate-Cluster]$ |
```

Resources:

> As per our architecture we need to create the resources like..

Vpc

Ecs

Ecr



Alb
Route53
IAM roles
Security groups

Vpc.tf

vpc:

> To Create vpc we created a file called vpc.tf,
> This vpc.tf file consist of code for creating vpc and subnets(two public subnets and two private subnets)

```
resource "aws_vpc" "main" {  
  cidr_block      = "172.20.0.0/16"  
  
  tags = {  
    Name = format("%s-%s-vpc", var.dns_name, var.account_environment)  
  }  
}
```

cidr_block: 172.20.0.0/16 allows you to use the IP address that starts with "172.20.X.X". There are 65,536 IP addresses ready to use.

Public subnet:

```
resource "aws_subnet" "public-subnet-1" {  
  vpc_id = "${aws_vpc.main.id}"  
  cidr_block = "172.20.10.0/24"  
  map_public_ip_on_launch = "true"  
  availability_zone = var.availability_zones[0]  
  tags = {  
    Name = format("%s-%s-public-subnet-1", var.dns_name, var.account_environment)  
  }  
}
```

Private subnet:

```
resource "aws_subnet" "private-subnet-1" {  
  vpc_id = "${aws_vpc.main.id}"  
  cidr_block = "172.20.20.0/24"  
  map_public_ip_on_launch = "false"  
  availability_zone = var.availability_zones[0]  
  tags = {  
    Name = format("%s-%s-private-subnet-1", var.dns_name, var.account_environment)  
  }  
}
```

vpc_id: this subnet will be the vpc just created before. We give the created VPC id to the subnet.

cidr_block: 10.0.1.0/24. We have 254 IP addresses in this subnet

map_public_ip_on_launch: This is so important. The only difference between private and public subnet is this line. If it is true, it will be a public subnet, otherwise private.

Network.tf:

Internet gateway:

```
resource "aws_internet_gateway" "main-igw" {  
  vpc_id = "${aws_vpc.main.id}"  
  tags = {  
    Name = format("%s-%s-igw", var.dns_name, var.account_environment)  
  }  
}
```

It enables your vpc to connect to the internet.



Route table:

```
resource "aws_route_table" "public-subnet-rt" {
  vpc_id = "${aws_vpc.main.id}"

  route {
    //associated subnet can reach everywhere
    cidr_block = "0.0.0.0/0"
    //Route table uses this IGW to reach internet
    gateway_id = "${aws_internet_gateway.main-igw.id}"
  }

  tags = {
    Name = format("%s-%s-igw", var.dns_name, var.account_environment)
  }
}

resource "aws_route_table_association" "RT-public-subnet-1-associate"{
  subnet_id = "${aws_subnet.public-subnet-1.id}"
  route_table_id = "${aws_route_table.public-subnet-rt.id}"
}

resource "aws_route_table_association" "RT-public-subnet-2-associate"{
  subnet_id = "${aws_subnet.public-subnet-2.id}"
  route_table_id = "${aws_route_table.public-subnet-rt.id}"
}
```

Here we are creating a route table with cidr "0.0.0.0/0" on any ip,with Internet gateway as target.

And then associating the route table to a public subnet.

Nat gateway:

```
# Creating an Elastic IP for the NAT Gateway!
resource "aws_eip" "Nat-Gateway-EIP" {
  vpc = true
}

# Creating a NAT Gateway!
resource "aws_nat_gateway" "NAT_GATEWAY" {
  depends_on = [
    aws_eip.Nat-Gateway-EIP
  ]
  allocation_id = aws_eip.Nat-Gateway-EIP.id
  subnet_id = "${aws_subnet.public-subnet-1.id}"
  tags = {
    Name = format("%s-%s-nat-gateway", var.dns_name, var.account_environment)
  }
}
```

NAT Gateway is a highly available AWS managed service that makes it easy to connect to the Internet from instances within a private subnet in an Amazon Virtual Private Cloud (Amazon VPC).

NAT Gateway Only Provides SNAT so private subnet instances can connect to the Internet, and Vice versa connectivity can't be established.

NAT Gateway also required an EIP (Elastic IP), So here we are creating an EIP and then we are creating a NAT gateway.

Note we are creating this NAT gateway in a public subnet because NAT gateway requires Internet connectivity, so instances launched in private subnet have internet connectivity. And NAT gateway will use Internet Gateway for Internet connectivity. Remember we associated a Route to Internet Gateway in public subnet.

Nat route table and association:

```
# Creating a Route Table for the Nat Gateway!
resource "aws_route_table" "NAT-Gateway-RT" {
  depends_on = [
    aws_nat_gateway.NAT_GATEWAY
  ]
  vpc_id = "${aws_vpc.main.id}"
  route {
    cidr_block = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.NAT_GATEWAY.id
  }
  tags = {
    Name = format("%s-%s-nat-gateway-RT", var.dns_name, var.account_environment)
  }
}

resource "aws_route_table_association" "Nat-Gateway-RT-Association-1" {
  depends_on = [
    aws_route_table.NAT-Gateway-RT
  ]
  subnet_id      = "${aws_subnet.private-subnet-1.id}"
  route_table_id = aws_route_table.NAT-Gateway-RT.id
}

resource "aws_route_table_association" "Nat-Gateway-RT-Association-2" {
  depends_on = [
    aws_route_table.NAT-Gateway-RT
  ]
  subnet_id      = "${aws_subnet.private-subnet-2.id}"
  route_table_id = aws_route_table.NAT-Gateway-RT.id
}
```

We have to create a route table with a target as NAT gateway and Associate it to a private subnet So instances in the private subnet can connect to the internet.

Here we are creating a route with cidr "0.0.0.0/0" meaning any ip and target as NAT gateway then we associate this table to a private subnet.

So the request from the instance inside the private subnet goes to NAT gateway in the public subnet and from NAT gateway it goes to Internet Gateway.

ecr.tf:

```
resource "aws_ecr_repository" "service-1" {  
  name = format("%s-%s-int-service-1-ecr", var.dns_name, var.account_environment)  
  image_tag_mutability = "Mutable"  
  
  image_scanning_configuration {  
    scan_on_push = true  
  }  
}
```

The ECR is a repository where we're gonna store the Docker Images of the application we want to deploy. It works like the Docker Hub, if you're familiar with Docker. You can build the Docker Image locally and push it to the ECR.

Ecr.tf:

Before we create the ECS Cluster, we need to create an IAM policy to enable the service to pull the image from ECR.

iam.tf:

```
# ECS task execution role data
data "aws_iam_policy_document" "ecs_task_execution_role" {
  version = "2012-10-17"
  statement {
    sid = ""
    effect = "Allow"
    actions = ["sts:AssumeRole"]

    principals {
      type       = "Service"
      identifiers = ["ecs-tasks.amazonaws.com"]
    }
  }
}

# ECS task execution role
resource "aws_iam_role" "ecs_task_execution_role" {
  name = format("%s-%s-iam-role", var.dns_name, var.account_environment)
  assume_role_policy = data.aws_iam_policy_document.ecs_task_execution_role.json
}

# ECS task execution role policy attachment
resource "aws_iam_role_policy_attachment" "ecs_task_execution_role" {
  role = aws_iam_role.ecs_task_execution_role.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
}
```

Now let's create what we need for ECS. First we create the ECS Cluster:

```
resource "aws_ecr_repository" "service-1" {
  name = format("%s-%s-int-service-1-ecr", var.dns_name, var.account_environment)
  image_tag_mutability = "MUTABLE"

  image_scanning_configuration {
    scan_on_push = true
  }
}
```

Route53.tf:

```
resource "aws_route53_record" "ecs-dns-record" {  
  zone_id = var.public_hosted_zone  
  name = "jenkins"  
  type = "A"  
  
  alias {  
    name = aws_alb.main-ext-lb.dns_name  
    zone_id = aws_alb.main-ext-lb.zone_id  
    evaluate_target_health = false  
  }  
}
```

> Here we are creating the **route53 dns record**, in the hosted zone we created earlier.

Alb.tf:

```
resource "aws_alb" "main-ext-lb" {  
  name = format("%s-%s-load-balancer", var.dns_name, var.account_environment)  
  internal = false  
  load_balancer_type = "application"  
  subnets = ["${aws_subnet.public-subnet-1.id}", "${aws_subnet.public-subnet-2.id}"]  
  security_groups = [  
    aws_security_group.main-ext-alb-sg.id  
  ]  
  tags = {  
    Name = format("%s-%s-load-balancer", var.dns_name, var.account_environment)  
  }  
}
```

> Here we are creating an alb(application load balancer), This is the top level component in the architecture. The ALB handles the incoming traffic, offloads SSL and balances the load.

Alb listeners:

```
resource "aws_alb_listener" "alb-listener-80" {
  load_balancer_arn = aws_alb.main-ext-lb.arn
  port = "80"
  protocol = "HTTP"
  default_action {
    type = "redirect"
    redirect {
      status_code = "HTTP_301"
      protocol = "HTTPS"
      port = "443"
    }
  }
}

resource "aws_alb_listener" "alb-listener-443" {
  load_balancer_arn = aws_alb.main-ext-lb.arn
  port = "443"
  protocol = "HTTPS"
  certificate_arn = var.alb_certificate
  default_action {
    type = "fixed-response"
    fixed_response {
      content_type = "text/html"
      message_body = "<html><head><title>404 Not Found</title></head><body><center><h1>404 Not Found</h1></center><hr><center>nginx</center></body></html>"
      status_code = "404"
    }
  }
}
```

> Listeners are assigned a specific port to keep an ear out for incoming traffic. They can have a maximum of 50 listeners assigned to each load balancer.

Alb listener rules:

```
resource "aws_alb_listener_rule" "rule-1" {  
  listener_arn = aws_alb_listener.alb-listener-443.arn  
  priority     = 1  
  action {  
    type = "forward"  
    target_group_arn = aws_alb_target_group.port8080-tg-2.arn  
  }  
  
  condition {  
    path_pattern {  
      values = ["/*"]  
    }  
  }  
}
```

> this is where things get pretty nifty! Each listener can have many rules which means we can route traffic to different places based on two conditions; the path and/or the host.

Target group:

```
resource "aws_alb_target_group" "port80-tg-1" {
  name      = format("%s-%s-port80-tg-1", var.dns_name, var.account_environment)
  port      = 80
  protocol  = "HTTP"
  target_type = "ip"
  vpc_id    = "${aws_vpc.main.id}"

  health_check {
    path = "/"
    matcher = "200"
  }
  tags = {
    Name = format("%s-%s-port80-tg-1", var.dns_name, var.account_environment)
  }
}

resource "aws_alb_target_group" "port8080-tg-2" {
  name      = format("%s-%s-port8080-tg-2", var.dns_name, var.account_environment)
  port      = 8080
  protocol  = "HTTP"
  target_type = "ip"
  vpc_id    = "${aws_vpc.main.id}"

  health_check {
    path = "/"
    matcher = "403"
  }
  tags = {
    Name = format("%s-%s-port8080-tg-2", var.dns_name, var.account_environment)
  }
}
```

> Target groups are essentially the end point of the ALB architecture — When the listener rule matches a pattern for a request it gets forwarded to the correlating target group. The cool thing about target groups is they have a health check that can directly check the health of a path



> Go inside the **environment** directory, you can find below files

```
[ec2-user@ip-172-31-26-25 environment]$ ls
dev.backend.tfvars  dev.tfvars
[ec2-user@ip-172-31-26-25 environment]$ |
```

> Edit the dev.backend.tfvars file using vi/vim editor. Give the bucket name which was created earlier and mention the region as required. Leave the key as default.

vim dev.backend.tfvars

```
ec2-user@ip-172-31-26-25:~/ecs/ECS_Fargate-Cluster/environment
```

```
# Terraform state
bucket = "kartheeks-bucket"
key = "terraform_state/infrastructure/dev/terraform.tfstate"
region = "us-east-1"
```


> In the same folder edit the dev.tfvars, This is the variables file which will be used inside the code.

vim dev.tfvars

```
ec2-user@ip-172-31-26-25:~/ecs/ECS_Fargate-Cluster/environment
region = "us-east-1"

#Environment information
dns_name = "indigo"
account_environment = "dev"
availability_zones = [
    "us-east-1a",
    "us-east-1b"
]

public_hosted_zone = "Z03840633HHYSQHJYWHJ9"
alb_certificate = "arn:aws:acm:us-east-1:653454134395:certificate/42606954-6e1d-4810-b01c-8129ecb4f6cf"

#node group details
ami_type = "AL2_x86_64"
disk_size = "20"
instance_types = ["t3.medium"]
desired_size = "1"
max_size = "1"
min_size = "1"
```

Region = < as you required >

Dns_name = < leave as default >

Account_environment = < leave as default >

Availability_zones = < give two availability zones based on selected region >

Public_hosted_zone = <give the hosted zone id >

Alb_certificate = <copy the arn id of the certificate we created and paste is here>

#node group details- leave as default

Terraform init: The terraform init command is used to initialize a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. It is safe to run this command multiple times.

> Then run the terraform init command, Using the below command and the output will look like this.

terraform init -backend-config=environment/dev.backend.tfvars

```
[ec2-user@ip-172-31-26-25 ECS_Fargate-Cluster]$ terraform init -backend-config=environment/dev.backend.tfvars
Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.16.0

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.
```

Terraform plan: The terraform plan command creates an execution plan, which lets you preview the changes that Terraform plans to make to your infrastructure. By default, when Terraform creates a plan it: Reads the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date.

> then run the terraform plan command to check what resources are going to be created.

terraform plan -var-file=environment/dev.tfvars

```
[ec2-user@ip-172-31-26-25 ECS_Fargate-Cluster]$ terraform plan -var-file=environment/dev.tfvars
aws_ecr_repository.service-1: Refreshing state... [id=indigo-dev-int-service-1-ecr]
aws_eip.Nat-Gateway-EIP: Refreshing state... [id=eipalloc-01db4742d52bfa42c]
data.aws_iam_policy_document.ecs_task_execution_role: Reading...
aws_ecs_cluster.ecs-cluster: Refreshing state... [id=arn:aws:ecs:us-east-1:653454134395:cluster/indigo-dev-ecs-cluster]
aws_vpc.main: Refreshing state... [id=vpc-01e8f04f4ab60235f]
data.aws_iam_policy_document.ecs_task_execution_role: Read complete after 0s [id=320642683]
aws_iam_role.ecs_task_execution_role: Refreshing state... [id=indigo-dev-iam-role]
aws_iam_role_policy_attachment.ecs_task_execution_role: Refreshing state... [id=indigo-dev-iam-role-20220531050824122600000001]
aws_security_group.main-ext-alb-sg: Refreshing state... [id=sg-0af6b7993daeece6e]
aws_alb_target_group.port8080-tg-2: Refreshing state... [id=sg-0f3d52f05d2ad6667]
aws_route_table.public-subnet-rt: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:653454134395:targetgroup/indigo-dev-port8080-tg-2/9cbf77678a635cf0]
aws_subnet.private-subnet-1: Refreshing state... [id=subnet-0a8d3cde97b650ba0]
aws_subnet.private-subnet-2: Refreshing state... [id=subnet-0a21b64b214f8e3ab]
aws_subnet.public-subnet-2: Refreshing state... [id=subnet-018cbdddc50809b52]
aws_internet_gateway.main-igw: Refreshing state... [id=igw-08106c5250afba665]
aws_alb_target_group.port80-tg-1: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:653454134395:targetgroup/indigo-dev-port80-tg-1/05ea8b3ca4435a4a]
aws_subnet.private-subnet-1: Refreshing state... [id=subnet-0c00c2eaf18e6567f]
aws_route_table.public-subnet-rt: Refreshing state... [id=rtb-0ed9b2790307e9229]
aws_security_group_rule.main-int-service-sg-ingress: Refreshing state... [id=sgrule-2790353270]
aws_security_group_rule.main-int-service-sg-egress: Refreshing state... [id=sgrule-1898004331]
aws_nat_gateway.NAT_GATEWAY: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:653454134395:loadbalancer/app/indigo-dev-load-balancer/c90bc395731ac8bf]
aws_route_table.NAT-Gateway-RT: Refreshing state... [id=rtb-083b4f2a9c4d06df1]
```

Terraform apply: The terraform apply command performs a plan just like terraform plan does, but then actually carries out the planned changes to each resource using the relevant infrastructure provider's API. It asks for confirmation from the user before making any changes, unless it was explicitly told to skip approval.

> After executing the terraform plan, we have to execute the **terraform apply** to create those resources in the console.

terraform apply -var-file=environment/dev.tfvars

> completion of apply it shows the **apply completed** as shown below

```
aws_cloudwatch_log_group.main: Creating...
aws_cloudwatch_log_group.main: Creation complete after 0s [id=/ecs/indigo/dev/port8080-service]
data.template_file.task_def: Reading...
data.template_file.task_def: Read complete after 0s [id=679eaa6e392979e0d03220cb8255b0d11d65bec1a15daab6aab739d0dd014714]
aws_ecs_task_definition.ecs_task_def: Creating...
aws_ecs_task_definition.ecs_task_def: Creation complete after 0s [id=ecs-indigo-dev-port8080-service]
aws_ecs_service.service: Creating...
aws_ecs_service.service: Creation complete after 1s [id=arn:aws:ecs:us-east-1:653454134395:service/indigo-dev-ecs-cluster/ecs-indigo-dev-port8080-service-service]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
[ec2-user@ip-172-31-26-25 service1]$
```

So it will create resources like ECS, ECR, ALB, IAM roles, VPC, Subnets, etc...
Once check in the console as they created as we mentioned in the code.

ECS:

☐ New ECS Experience
[Tell us what you think](#)

Amazon ECS

Clusters

Task Definitions

Account Settings

Amazon EKS

Clusters

Amazon ECR

Repositories

AWS Marketplace

Discover software

Subscriptions [↗](#)

Clusters

An Amazon ECS cluster is a regional grouping of one or more container instances on which you can run task requests. Each account receives a default cluster the first time you create a cluster. Clusters may contain more than one Amazon EC2 instance type.

For more information, see the [ECS documentation](#).

[Create Cluster](#)

[Get Started](#)

View ☒ list ☐ card

[indigo-dev-ecs-cluster](#) >

CloudWatch monitoring

☒ Default Monitoring

FARGATE

1

Services

1

Running tasks

0

Pending tasks

EC2

0

Services

0

Running tasks

0

Pending tasks

No data

CPUUtilization

No data

MemoryUtilization

EXTERNAL

ECR:

Amazon Elastic Container Registry

Private registry

Public registry

Repositories

Getting started

Documentation

Public gallery

Amazon ECR > Repositories

Private Public

Private repositories (1)

Find repositories

View push commands Delete Edit Create repository

	Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type	Pull through cache
<input type="radio"/>	indigo-dev-int-service-1-ecr	653454134395.dkr.ecr.us-east-1.amazonaws.com/indigo-dev-int-service-1-ecr	May 31, 2022, 10:38:24 (UTC+05.5)	Disabled	Scan on push	AES-256	Inactive

Vpc:

New VPC Experience

VPC Dashboard

EC2 Global View

Filter by VPC:

Select a VPC

VIRTUAL PRIVATE CLOUD

Your VPCs

Subnets

Route Tables

Internet Gateways

Firewall

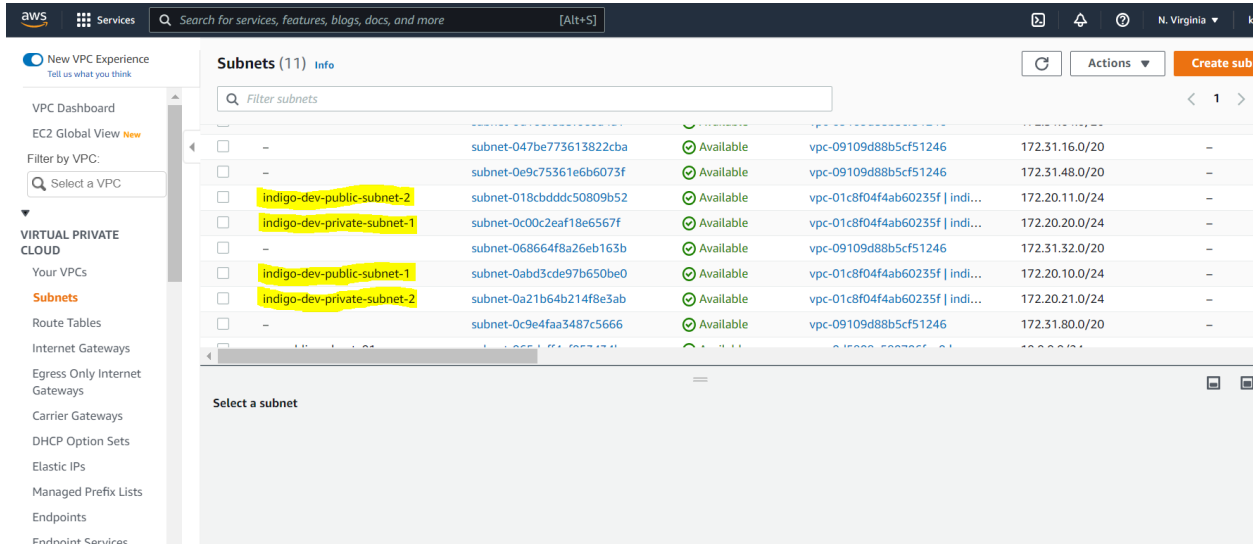
Your VPCs (3)

Filter VPCs

Actions Create VPC

	Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR	DHCP opt
<input type="checkbox"/>	-	vpc-09109d88b5cf51246	Available	172.31.0.0/16	-	dopt-084
<input type="checkbox"/>	indigo-dev-vpc	vpc-01c8f04f4ab60235f	Available	172.20.0.0/16	-	dopt-084
<input type="checkbox"/>	gs-vpc	vpc-0d5808e589796fea9	Available	10.0.0.0/16	-	dopt-084

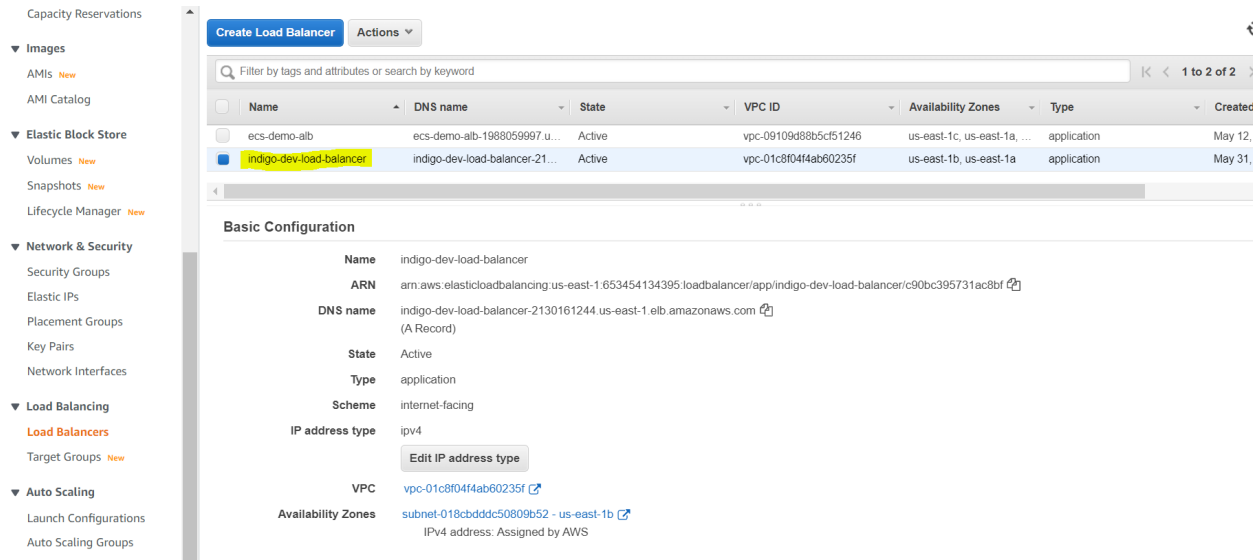
Subnets:



The screenshot shows the AWS Subnets console. On the left, there's a navigation menu with options like VPC Dashboard, EC2 Global View, and VIRTUAL PRIVATE CLOUD. The main area displays a list of 11 subnets. The subnets are listed with their names, IDs, states, VPC IDs, and CIDR blocks. The subnets are:

Name	ID	State	VPC ID	CIDR Block
subnet-047be773613822cba	subnet-047be773613822cba	Available	vpc-09109d88b5cf51246	172.31.16.0/20
subnet-0e9c75361e6b6073f	subnet-0e9c75361e6b6073f	Available	vpc-09109d88b5cf51246	172.31.48.0/20
indigo-dev-public-subnet-2	subnet-018cbdddc50809b52	Available	vpc-01c8f04f4ab60235f indi...	172.20.11.0/24
indigo-dev-private-subnet-1	subnet-0c00c2eaf18e567f	Available	vpc-01c8f04f4ab60235f indi...	172.20.20.0/24
subnet-068664f8a26eb163b	subnet-068664f8a26eb163b	Available	vpc-09109d88b5cf51246	172.31.32.0/20
indigo-dev-public-subnet-1	subnet-0abd3cde97b650be0	Available	vpc-01c8f04f4ab60235f indi...	172.20.10.0/24
indigo-dev-private-subnet-2	subnet-0a21b64b214f8e3ab	Available	vpc-01c8f04f4ab60235f indi...	172.20.21.0/24
subnet-0c9e4faa3487c5666	subnet-0c9e4faa3487c5666	Available	vpc-09109d88b5cf51246	172.31.80.0/20

ALB:



The screenshot shows the AWS Elastic Load Balancing console. On the left, there's a navigation menu with options like Capacity Reservations, Images, Elastic Block Store, Network & Security, Load Balancing, and Auto Scaling. The main area displays a list of load balancers. The load balancers are listed with their names, DNS names, states, VPC IDs, availability zones, types, and creation dates. The load balancers are:

Name	DNS name	State	VPC ID	Availability Zones	Type	Created
ecs-demo-alb	ecs-demo-alb-1988059997 u...	Active	vpc-09109d88b5cf51246	us-east-1c, us-east-1a, ...	application	May 12,
indigo-dev-load-balancer	indigo-dev-load-balancer-21...	Active	vpc-01c8f04f4ab60235f	us-east-1b, us-east-1a	application	May 31,

Below the list, there's a 'Basic Configuration' section for the selected load balancer 'indigo-dev-load-balancer'.

- Name:** indigo-dev-load-balancer
- ARN:** arn:aws:elasticloadbalancing:us-east-1:653454134395:loadbalancer/app/indigo-dev-load-balancer/c90bc395731ac8bf
- DNS name:** indigo-dev-load-balancer-2130161244.us-east-1.elb.amazonaws.com (A Record)
- State:** Active
- Type:** application
- Scheme:** internet-facing
- IP address type:** ipv4
- VPC:** vpc-01c8f04f4ab60235f
- Availability Zones:** subnet-018cbdddc50809b52 - us-east-1b, subnet-0abd3cde97b650be0 - us-east-1a

Push Jenkins Image to ECR:

> After the successful creation of resources, we have to push the image to ecr for that to follow the below commands.

a) Pull the jenkins image to your local instance by executing this command.

Docker pull jenkins/jenkins

Then you can check by running **docker images** command

```
[root@ip-172-31-26-25 ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
jenkins/jenkins      latest             011729edd836       5 days ago        460MB
[root@ip-172-31-26-25 ~]# |
```

- b) Goto container registry and click on push commands button the pop-up will appear like below.

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](#).

1. Retrieve an authentication token and authenticate your Docker client to your registry.

Use the AWS CLI:

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin
653454134395.dkr.ecr.us-east-1.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.

2. 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 indigo-dev-int-service-1-ecr .
```

3. After the build completes, tag your image so you can push the image to this repository:

```
docker tag indigo-dev-int-service-1-ecr:latest 653454134395.dkr.ecr.us-east-1.amazonaws.com/indigo-dev-int-
service-1-ecr:latest
```

4. Run the following command to push this image to your newly created AWS repository:

```
docker push 653454134395.dkr.ecr.us-east-1.amazonaws.com/indigo-dev-int-service-1-ecr:latest
```

Run the first command to authenticate client to your registry.

- c) Then tag the jenkins image which pulled earlier, as shown in the above image.

docker tag jenkins/jenkins

653454134395.dkr.ecr.us-east-1.amazonaws.com/indigo-dev-int-service-1-ecr:latest

```
[root@ip-172-31-26-25 ~]# docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
jenkins/jenkins      latest          011729edd836   5 days ago     460MB
[root@ip-172-31-26-25 ~]# docker tag jenkins/jenkins 653454134395.dkr.ecr.us-east-1.amazonaws.com/indigo-dev-int-service-1-ecr:latest
[root@ip-172-31-26-25 ~]# docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
653454134395.dkr.ecr.us-east-1.amazonaws.com/indigo-dev-int-service-1-ecr   latest          011729edd836   5 days ago     460MB
jenkins/jenkins      latest          011729edd836   5 days ago     460MB
[root@ip-172-31-26-25 ~]#
```

- d) Push that tagged image to registry by executing the following command.
docker push 653454134395.dkr.ecr.us-east-1.amazonaws.com/indigo-dev-int-service-1-ecr:latest
- e) Once push is successful check in the console that image is pushed to our registry.

Amazon Elastic Container Registry

- Private registry
- Public registry
- Repositories
- Summary
- Images**
- Permissions
- Lifecycle Policy
- Tags

Getting started [🔗](#)

Documentation [🔗](#)

Public gallery [🔗](#)

Amazon ECR > Repositories > indigo-dev-int-service-1-ecr

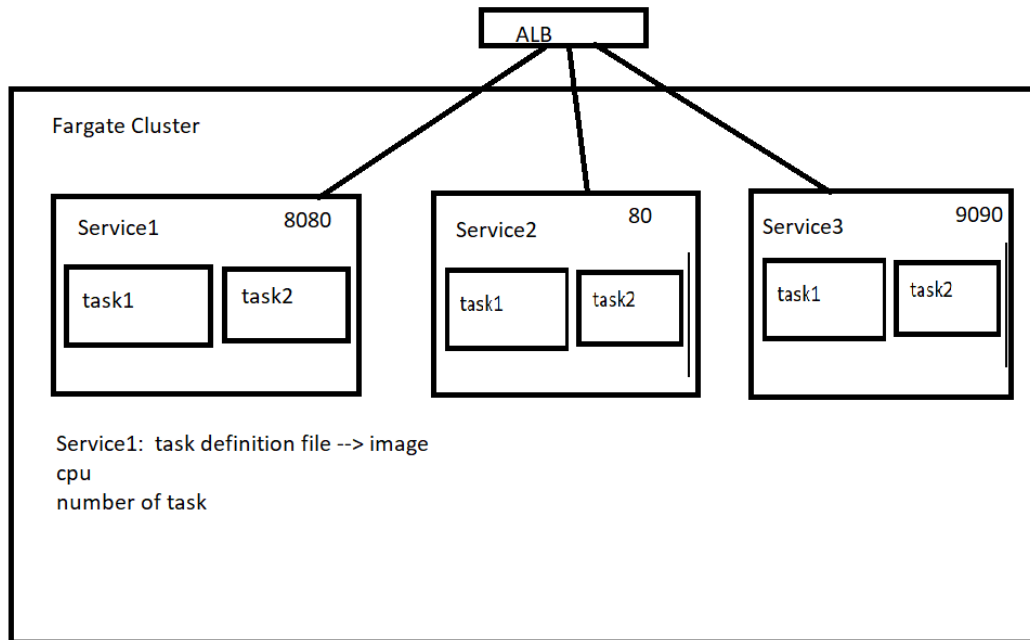
indigo-dev-int-service-1-ecr View push

Images (1) 🔄

<input type="checkbox"/>	Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	Scan status
<input type="checkbox"/>	latest	Image	May 31, 2022, 16:13:46 (UTC+05.5)	285.19	Copy URI	sha256:124c3394c87cf91...	Complete

Creating Service:

Architecture:



Amazon ECS services hosted on Amazon EC2 instances support the Application Load Balancer, Network Load Balancer, and Classic Load Balancer load balancer types. Amazon ECS services hosted on AWS Fargate support Application Load Balancer and Network Load Balancer only. Application Load Balancers are used to route HTTP/HTTPS (or layer 7) traffic. Network Load Balancers are used to route TCP or UDP (or layer 4) traffic. Classic Load Balancers are used to route TCP traffic.

> As of now we have created the ecs cluster and pushed image to the ecr and checked up and running. Now we have to create a service to access from outside. For this we have created an ALB.

> Go inside the service directory directory.

/home/ec2-user/ecs/ECS_Fargate_Service/service1

> Edit the dev.backend.tfvars file using vi/vim editor. Give the bucket name which was created earlier and mention the region as required. Leave the key as default.

cd environment

ec2-user@ip-172-31-26-25:~/ecs/ECS_Fargate_Service/service1/environment

```
# Terraform state
bucket = "kartheeks-bucket"
key = "terraform_state/service-80/dev/terraform.tfstate"
region = "us-east-1"
```

> In the same folder edit the dev.tfvars using vi/vim editor , this is the variables file

```
region = "us-east-1"

#Environment information
dns_name = "indigo"
account_environment = "dev"
#####
image_name = "653454134395.dkr.ecr.us-east-1.amazonaws.com/indigo-dev-int-service-1-ecr:latest"
ecs_role = "arn:aws:iam::653454134395:role/indigo-dev-iam-role"
ecs_cluster_id = "indigo-dev-ecs-cluster"
web_sg = "sg-0f5d52f05d2ad6667"
private_subnets = [
    "subnet-0c00c2eaf18e6567f",
    "subnet-0a21b64b214f8e3ab",
]
service_tg_arn = "arn:aws:elasticloadbalancing:us-east-1:653454134395:targetgroup/indigo-dev-port8080-tg-2/9cbf77678a635cf0"
#####
task_family_name = "port8080-service"
port = "8080"
replicas = "1"
```

Region = <give as required>
 Dns_name = <leave as default>
 Account_environment = <leave as default>
 Image_name = <give the tagged image name we did on earlier steps>
 Ecs_role = <you will find this in output of the **ecs_cluster** creation >
 Ecs_cluster = <leave as default>
 Web_sg = <you will find this in output of the **ecs_cluster** creation>
 Private_subnets = <you will find this in output of the **ecs_cluster** creation>
 Service_tg_arn = <you will find this in output of the **ecs_cluster** creation>
 Task_family_name = <leave as default>
 Port = <leave as default>
 Replicas = <leave as default>

These values will be found in the console if it is difficult to find those, go to the ECS_Fargate-Cluster and run **terraform output** as shown below, So you can find these

values.

```
[ec2-user@ip-172-31-26-25 ECS_Fargate-Cluster]$ terraform output
ecs-cluster = "indigo-dev-ecs-cluster"
ecs-iam-role = "arn:aws:iam::653454134395:role/indigo-dev-iam-role"
private-subnet-1 = "subnet-0c00c2eaf18e6567f"
private-subnet-2 = "subnet-0a21b64b214f8e3ab"
targetgroup-port80 = "arn:aws:elasticloadbalancing:us-east-1:653454134395:targetgroup/indigo-dev-port80-tg-1/05ea8b3ca4435a4a"
targetgroup-port8080 = "arn:aws:elasticloadbalancing:us-east-1:653454134395:targetgroup/indigo-dev-port8080-tg-2/9cbf77678a635cf0"
web-sg = "sg-0f5d52f05d2ad6667"
[ec2-user@ip-172-31-26-25 ECS_Fargate-Cluster]$ |
```

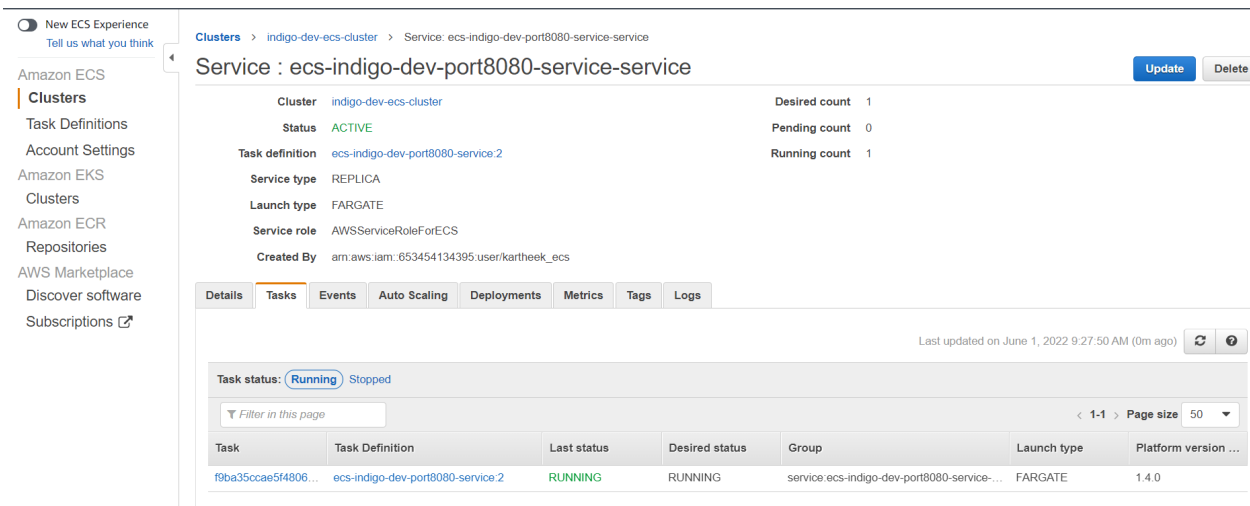
> after successfully changing the values you have to go to the ECS_Fargate_Service folder and run go inside the service1 folder then run the below commands

terraform init -backend-config=environment/dev.backend.tfvars

terraform plan -var-file=environment/dev.tfvars

terraform apply -var-file=environment/dev.tfvars

> So the terraform creates the service for the jenkins , goto console and checks if ecs services status is running.



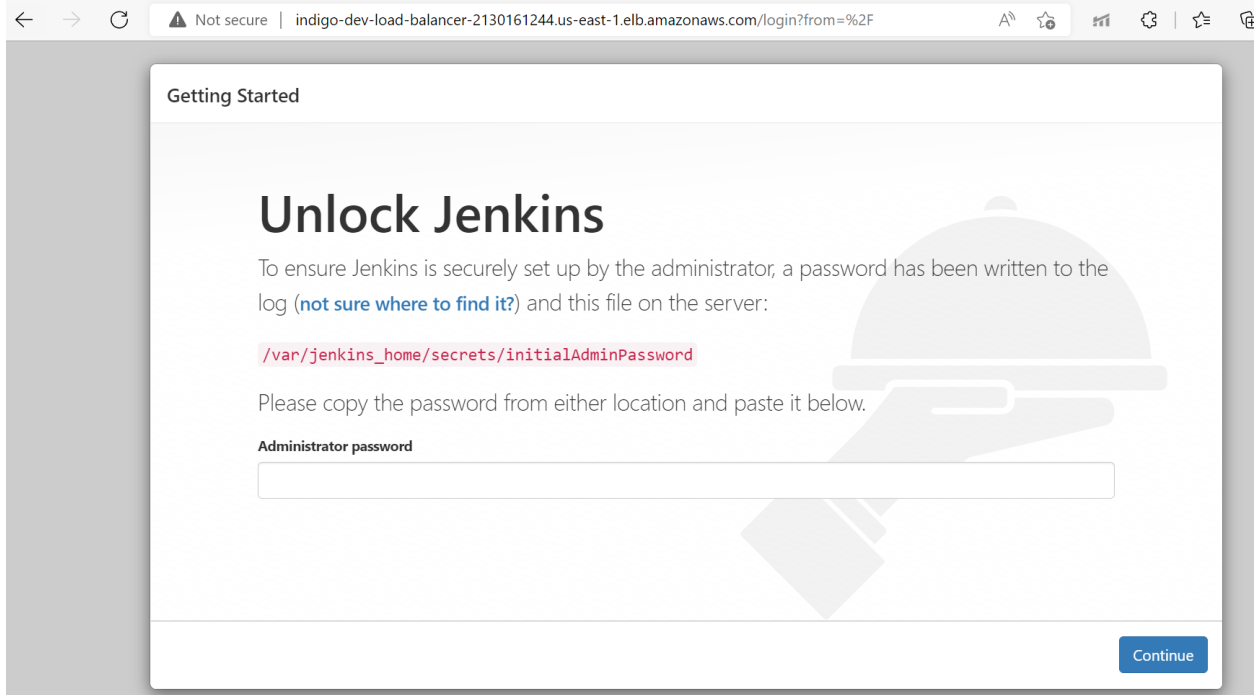
Service : ecs-indigo-dev-port8080-service-service

Cluster	indigo-dev-ecs-cluster	Desired count	1
Status	ACTIVE	Pending count	0
Task definition	ecs-indigo-dev-port8080-service:2	Running count	1
Service type	REPLICA		
Launch type	FARGATE		
Service role	AWSServiceRoleForECS		
Created By	arn:aws:iam::653454134395:user/kartheek_ecs		

Task status: **Running** Stopped

Task	Task Definition	Last status	Desired status	Group	Launch type	Platform version ...
f9ba35ccae5f4806...	ecs-indigo-dev-port8080-service:2	RUNNING	RUNNING	service:ecs-indigo-dev-port8080-service-...	FARGATE	1.4.0

> after successfully creating the service, we have to copy the dns and hit on the browser to check if jenkins dashboard is appearing or not.



Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue