Days:

day1

-AWS console and introduction

-IAM

-IAM is a Global service mainly for Authentication and Authorization

-Route53 is also a global service

Cloud formation: It is mainy for infrastructure as a code

Type of instance:

1. On demand

2. Reserved

3. Spot instance

GLobal infrastructure:

Region

Availability zone

edge location

CDN(CONTENT DELIVERY NETWORK): origin of the data is somewhere else but

Types of machine type

1.General purpose type

2.storage type

3.memory type

4.Network optimized

5.Compute optimized

NACL-Network access control list

IAM is a Global service mainly for Authentication and Authorization

JSON-javascript object notation

IAM policy

-Identity based policy

-Resource based policy-eg:sns,s3

Compute services

Ec2

-on demand

-Reserved

-spot instance

You can connect Aws through console,Cli,sdk/cdk,Iac

6 pillars of Aws

-operational excellence, security, reliability, performance efficiency, cost optimization, and sustainability

Three types of storage

-object storage -s3

-block storage -EBS

-File storage -EFS

-Compute service

-IAM

-Storage

-container

-Integration service (SNS SQS)

-Serverless service(api,dynamo db,lambda)

-Database service-Rds

-Monitoring service-cloud watch

-Management services and Govt services

-Network services(vpc,loadbalancer)

DEVOPS PROFESSIONAL:

-code build

-code pipeline

-system manager

-Difference between dedicated instance and dedicated host

-Difference between SAAS,PAS,IAS

-Purchase options

-What is placement group

**DAY-2**

IAM user

Group

Roles

DAY-3(Launching ec2 isntance,launching vm in windows,ssh connection to windows ,hosting static website in windows with global access enabled and restricted)

Instance:

1:Name of the instance

2.AMI(Amazon machine images)-OS+pre installed softwares

3.Instance type

**-General purpose-Tseries**

**-Compute optimized- High perfomance-cseries**

**-Memory optimized-Datasets -Rseries**

**-storage optimized-Sequential read and write**

4.Key-pair is required for authentication purpose -pem file

If key is lost we can get back using session manager

Ssh connection:

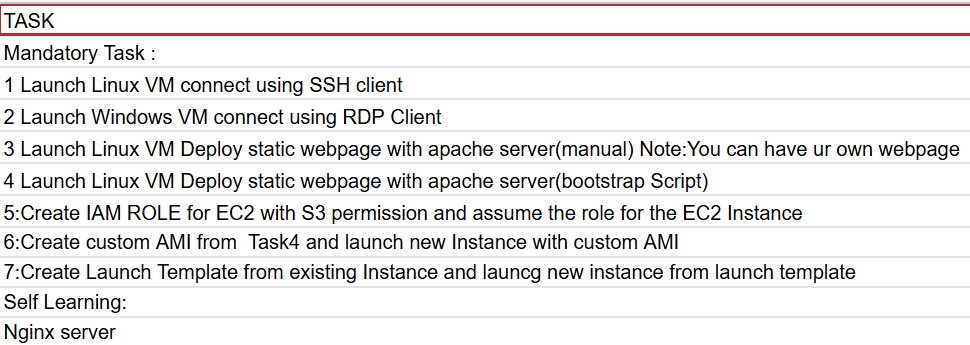
ssh -i “ust1.pem” username@publicip of machine

yum update -y -Linux machine update

yum install httpd -y

systemctl status httpd

systemctl restart httpd



Task1:

Placement group

Spot

Partition

**DAY-3**

-Code commit-source code management in aws

--Iam—user—assign mfa device—

Aws s3 ls

Aws configure

Aws –version

Aws s3 mb s3://my-new-bucket

Task1:

**Create IAM user and attach the policy and validate:**

-Go to IAM → Click Users → Click Add user

- **User details:**

* Username: user1 (or any name)
* Select **Access key - Programmatic access** (for CLI/API)

 **Permissions:**

* Choose **Attach policies directly**
* Select a policy (e.g., AmazonS3FullAccess or AdministratorAccess as needed)
* Then create bucker in user1 account you can create it but now go to root account and change the permission

Then check with keeping policy as s3readonlyaccess you cannot create bucket in the user1 account

**Task2**

**Create IAM groups and add various users and validate**

-Go to AWS Console > IAM > User groups

-Click Create group ->Group name:anything

-**Attach policies (optional)**:

* Example: AmazonS3ReadOnlyAccess for read-only users

-Click **Create group**

**Creating users**

IAM > Users > Add users

Add usernames (e.g., user1, user2)

IAM > User groups > Select your group

Go to **Users tab** → Click **Add users**

Select the users (e.g., user1, user2) → Add

IAM > User groups > [Select your group] > **Permissions** tab

Click **Add permissions**

Choose **Attach policies directly**

Add a policy (e.g., AmazonEC2ReadOnlyAccess, AmazonS3FullAccess)

Click **Add permissions**

**Task3:**

Go to **IAM > Roles > Create Role**

Choose **AWS service**

**Use case**: Choose **EC2**, click **Next**

Attach a policy like AmazonS3ReadOnlyAccess or any policy you want.

Name the role: e.g., EC2\_S3\_ReadOnly\_Role

Create the role.

**Attach IAM Role to EC2 Instance**

Go to **EC2 > Instances**

Select your instance

Choose **Actions > Security > Modify IAM Role**

Attach the role you created (EC2\_S3\_ReadOnly\_Role)

Click **Update IAM role**

**SSH into your EC2 instance:**

aws s3 ls(listing all the available buckets)

aws s3 mb s3://my-buck-name

**Task4:**

Task 6: enable MFA and validate

Security credentials tab, scroll to “Multi-factor authentication (MFA)”

Click “Assign MFA device”

Scan the **QR code** using your app

Enter **two consecutive MFA codes** generated by the app

Click **Assign MFA**

**Task7:**

**DAY-4**

**Class A – Network bit -8**

Host bits -24

Default subnet for class A -255.0.0.0

Cidr -/8

Class B – 16 net bits 16 hostbits

Cidr-/16

Ip-255.255.0.0

-Class c

Default subnet mask -255.255.255.0

Cidr-/24

Nw bits=24 hs bit=8

No of hosts=2^8-2 for aws 2^8-5

**Step-1**

-Create vpc -192.168.3.0/24

Name:NCPL-VPC-A

**Step-2: create two subnet name it public subnt and private subnet**

-2a-Create public subnet -192.168.3.0/25 under VPC-A

NAME:VPC-A-Pubsub

-2b-Create public subnet -192.168.3.128/25 under VPC-A

NAME:VPC-A-PriSub

Step 3:Create two route table ,one for each subnet

3a-route table1

NAME:A-PubSub-RT

3b-route-Table1

NAME:A-PriSub-RT

Step-4:subnet Association

Ie:A-PubSub-RT must be associated with VPC-A-PubSub

Ie:A-PriSub-RT must be associated with VPC-A-PriSub

Step5: Create Internet Gateway and attach it wih VPC-A

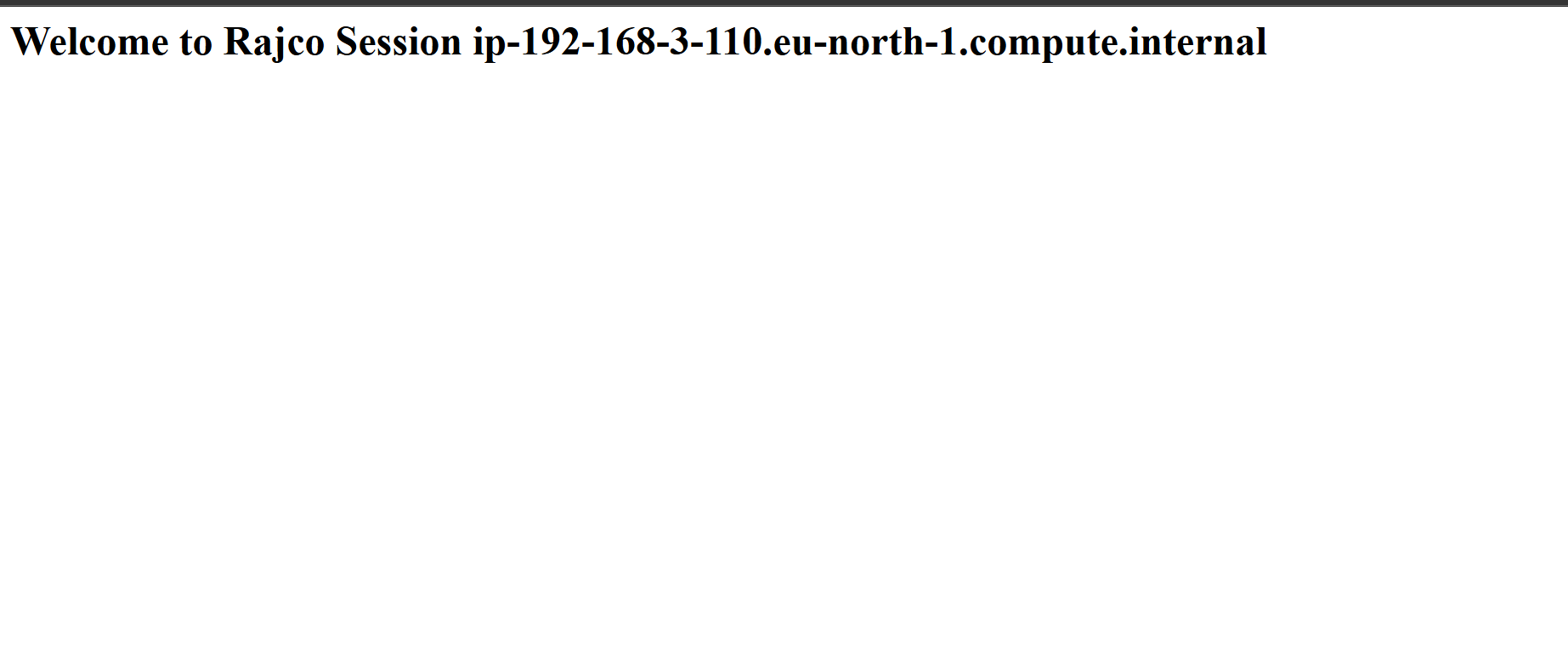
NAME:UST-ITGW

Step6:To make a subnet VPC-A-PubSubas public -go to route table of A-PubSub-RT of the subnet add the route to destination 0.0.0.0./0 through target internetgateway

Step7:Create security group under UST-VPC-A with needed inbounds rule(ssh,http)

Step8:launch ec2 instance under custom UST-VPC-A under public subnet A-PubSub with auto assign public ip enabled

Output:



Doing the above task using Aws command line:

**First**Configure Aws: Aws configure

export AWS\_ACCESS\_KEY\_ID=

AWS\_SECRET\_ACCESS\_KEY=

AWS\_SESSION\_TOKEN=

**Step-1**

**Create a vpc**

aws ec2 create-vpc --cidr-block 192.168.3.0/24 --tag-specifications 'ResourceType=vpc,Tags=[{Key=Name,Value=NCPL-VPC-A}]'

**Step-2**

**Create Subnets**

**Public subnet:**

aws ec2 create-subnet --vpc-id <VPC-ID> --cidr-block 192.168.3.0/25 --availability-zone eu-north-1a --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=VPC-A-PubSub}]'

**Private subnet:**

aws ec2 create-subnet --vpc-id <VPC-ID> --cidr-block 192.168.3.128/25 --availability-zone eu-north-1a --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=VPC-A-PriSub}]'

**Step-3**

**Create Route Tables**

**Public Route Table**

aws ec2 create-route-table --vpc-id <VPC-ID> --tag-specifications 'ResourceType=route-table,Tags=[{Key=Name,Value=A-PubSub-RT}]'

**Private Route Table**

aws ec2 create-route-table --vpc-id <VPC-ID> --tag-specifications 'ResourceType=route-table,Tags=[{Key=Name,Value=A-PriSub-RT}]'

**Step-5**

**Associate Subnets with Route Tables**

aws ec2 associate-route-table --subnet-id <PubSubnet-ID> --route-table-id <PubRT-ID>

aws ec2 associate-route-table --subnet-id <PriSubnet-ID> --route-table-id <PriRT-ID>

**Attach it to the VPC:**

aws ec2 attach-internet-gateway --internet-gateway-id <IGW-ID> --vpc-id <VPC-ID>

**Step6:**

**Add Route to Internet for Public Subnet**

aws ec2 create-route --route-table-id <PubRT-ID> --destination-cidr-block 0.0.0.0/0 --gateway-id <IGW-ID>

**Step7**

**Create Security Group**

aws ec2 create-security-group --group-name UST-VPC-A-SG --description "Allow SSH and HTTP" --vpc-id <VPC-ID>

aws ec2 authorize-security-group-ingress --group-id <SG-ID> --protocol tcp --port 22 --cidr 0.0.0.0/0

aws ec2 authorize-security-group-ingress --group-id <SG-ID> --protocol tcp --port 80 --cidr 0.0.0.0/0

**Step8**

**Launch EC2 Instance with Bootstrap Script**

#!/bin/bash

while ! ping -c 1 -W 5 8.8.8.8 &>/dev/null; do

echo "Waiting for internet connection..."

sleep 5

done

yum update -y

yum install -y httpd

chmod 777 -R /var/www/html

echo "<h1> Welcome to Rajco Session $(hostname -f) </h1>" > /var/www/html/index.html

systemctl start httpd

systemctl enable httpd

**Then launch instance:**

aws ec2 run-instances \

--image-id ami-0c259f0e2b5c5d817 \ # Replace with the correct Amazon Linux 2 AMI ID for eu-north-1

--count 1 \

--instance-type t2.micro \

--key-name <Your-Key-Pair-Name> \

--security-group-ids <SG-ID> \

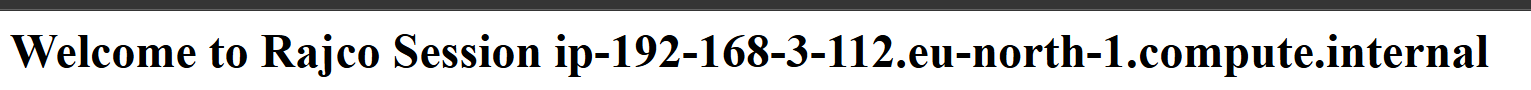
--subnet-id <PubSubnet-ID> \

--associate-public-ip-address \

--user-data file://user-data.sh \

--tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=NCPL-WebServer}]'

Output:



**DAY-7**

Load balancer

Three types of loadbalancer

-Application load balancer

>It works on layer 7(https,http)

-network load balancer

>it works on layer4 (Tcp,udp)

-Gateway Load balancer

>Third security application

Name fo lb

Internet or internal

Voc-and availability zone

Security group for lb

Various scaling policy

-Target policy

-Step policy

-Simple scaling policy

-Predicitve scaling policy

-Creating the launch template

-Selecting the vpc and subnets

-Decide the Group size min-3 and max-5

**DAY-8**

**RDS tasks**

**Versioning-**

Same objects can have multiple features

Standard-the object that your storing freq

Standard ia-obj not gets stored freq

Standard 1 zone ia-object get stord in 1 availabilty zone

Intelligent tier

Glacier

Cheapest class is glacier

Access control bucket policy

-object level ownership

-resource level ownership

-Bucket level policy

**Storage life cycle-Transfer access c**

**Replication rule**

**Cors Cross origin resource sharing**

**S3 Event handling**

**S3 logging**

**Server side encryption**

**Signed url**

**DAY-8**

**Teraform init -**

**Teraform plan-compare and review**

|  |
| --- |
| 1)Connecting ec2 instance with VScode using ssh |
| 2)Launcing an ec2 instance by using Terraform |
| 3)what happens when we give terraform init in background |
| 4)what happens when we give terraform plan in background |
| 5)what happens when we give terraform apply in background |
| 6)what happens when we give terraform show in background |

Step 1: Open VS Code and install the "Remote - SSH" extension from the Extensions view.

Step 2: Use the Remote Explorer panel in VS Code to add your SSH configuration. Click on the + button and add your connection details, which should look like this:

**ssh -I “mykey1.pem” ec2-user@public-ip**

**eg: ssh -i ~/.ssh/mykey1.pem -p 80 ec2-user@** 44.219.250.91

**ssh -i ~/.ssh/mykey1.pem -p 80 ec2-user@44.219.250.91**

Connect by clicking on the SSH host in the Remote Explorer panel, and VS Code will establish an SSH connection to the EC2 instance. Ensure that your security group allows inbound SSH traffic on port 22.

**Step-2 Launching an EC2 Instance using Terraform**

Create a main.tf file containing the EC2 resource configuration:

provider "aws" {

region = "us-west-2"

}

resource "aws\_instance" "example" {

ami = "ami-12345678"

instance\_type = "t2.micro"

key\_name = "your-key-pair"

tags = {

Name = "ExampleInstance"

}

}

# S3 Bucket

resource "aws\_s3\_bucket" "example\_bucket" {

bucket = "example-bucket-123456"

}

 Step 3:first configure aws credentials then Initialize Terraform with terraform init.

 Step 4: Apply the configuration using terraform apply.

 Step 5: Terraform will create an EC2 instance as per the configuration.

**DAY-9**

**Terraform fmt**

**Adminuser -Admin@123**

**DAY-10**

|  |  |  |  |
| --- | --- | --- | --- |
| 4) Place your module in git repo and create resource using it |  |  |  |

**Manually Upload Files Using GitHub UI**

* Navigate to your repo
* Click **"Add file" → "Create new file"** or **"Upload files"**
* Add your module files. For example:

css

CopyEdit

modules/

└── ec2/

├── main.tf

├── variables.tf

└── outputs.tf

Example main.tf for an EC2 module:

hcl

CopyEdit

resource "aws\_instance" "this" {

ami = var.ami

instance\_type = var.instance\_type

tags = {

Name = var.name

}

}

Example variables.tf:

h

CopyEdit

variable "ami" {

description = "AMI ID"

type = string

}

variable "instance\_type" {

description = "Instance type"

type = string

}

variable "name" {

description = "EC2 instance name"

type = string

}

Example outputs.tf:

hcl

CopyEdit

output "instance\_id" {

value = aws\_instance.this.id

}

**3️⃣ Use Your GitHub Module in Another Terraform Project**

In a separate Terraform project (on your EC2 or local system), use this module like this:

hcl

CopyEdit

module "my\_ec2" {

source = "git::https://github.com/karthikeya964/aws-module-project1.git//modules/ec2"

ami = "ami-0abcdef1234567890"

instance\_type = "t2.micro"

name = "my-ec2-instance"

}

**4️⃣ Run Terraform as Usual**

terraform init

terraform apply

Place your module in terraform registry and create a resource using it

Publishing a module to the **Terraform Registry** (public or private) involves a few key steps. Below is a complete **step-by-step guide** for placing your module on the **Terraform Public Registry** and using it to create a resource.

**✅ Prerequisites**

1. **GitHub account**
2. **Terraform module following the standard naming convention**
3. **Git installed locally**
4. **Terraform module structured properly**

**1️⃣ Module Naming Convention**

Your **GitHub repo** name must follow this exact format:

php-template

CopyEdit

terraform-<PROVIDER>-<NAME>

✅ Example:

bash

CopyEdit

terraform-aws-ec2-example

This is **mandatory** for the Terraform Registry to recognize your module.

**2️⃣ Folder Structure Example**

terraform-aws-ec2-example/

├── main.tf

├── variables.tf

├── outputs.tf

├── README.md

**3️⃣ Create the GitHub Repository**

* Go to <https://github.com/new>
* Repository name: terraform-aws-ec2-example
* Description: "A simple EC2 instance module"
* Make it **Public**
* Click **Create Repository**
* Upload your Terraform module files or push via git

**4️⃣ Tag a Release Version**

Terraform Registry uses **Git tags** for module versions. On your local system:

bash

CopyEdit

git init

git remote add origin https://github.com/<your-username>/terraform-aws-ec2-example.git

git add .

git commit -m "Initial commit"

git push -u origin master

# Tag it

git tag v1.0.0

git push origin v1.0.0

**5️⃣ Publish to Terraform Registry**

1. Go to: <https://registry.terraform.io>
2. Sign in with **GitHub**
3. Go to your profile → **"Add Module"**
4. Select the repo: terraform-aws-ec2-example
5. Click **"Publish Module"**

Terraform Registry will now index your module.

**6️⃣ Use the Module in Any Terraform Project**

Once published, use it like this:

h

CopyEdit

module "ec2\_instance" {

source = "karthikeya964/ec2-example/aws"

version = "1.0.0"

ami = "ami-0abcdef1234567890"

instance\_type = "t2.micro"

name = "my-ec2"

}

Replace karthikeya964 with your GitHub username and update the module name as appropriate.

|  |  |  |
| --- | --- | --- |
| 6) Create a s3 bucket and store your statefile in it |  |  |
| : Manually create it from AWS Console   1. Go to [AWS S3 Console](https://s3.console.aws.amazon.com/s3/home) 2. Click “Create bucket” 3. Enter bucket name: buckbroz-zkb (must match your backend config exactly) 4. Choose Region: ap-south-1 (or the one you are using) 5. Leave defaults or enable versioning (recommended) 6. Click Create bucket   Create a backend.tf file (if not already created)  Example backend.tf  terraform {  backend "s3" {  bucket = "buckbroz-zkb" # Replace with your S3 bucket name  key = "terraform/statefile.tfstate" # Path within the bucket  region = "ap-south-1" # Your region  encrypt = true  }  } |  |  |

DAY-11

Meta Arguement

For each

Count

Dynamic

Data

Local

**Life cycle**

**DAY-zz**

Interface endpoint : it uses security groups and network interface to allow access

Gateway endpoint: it uses route tables to give access to major aws services such as s3 and db

Gateway endpoint

* A VPC with at least one **private subnet**
* The instance must have:
  + **S3 access IAM role or policy**
  + **Route to the S3 endpoint**
  + Security group allowing **HTTPS (port 443)** egress

**🔹 Step 1: Open the VPC Console**

* Go to **VPC Dashboard** → **Endpoints** → **Create Endpoint**

**🔹 Step 2: Configure Endpoint**

1. **Service category**: AWS services
2. **Service name**: Find and select the S3 service for your region. Example:  
   com.amazonaws.<region>.s3
3. **VPC**: Choose the correct VPC
4. **Endpoint type**: Choose **Gateway**
5. **Route Tables**: Select the route table(s) associated with **your private subnet**
6. Click **Create Endpoint**

**🔹 Step 3: Route Table Automatically Updated**

* The selected private route table will now have a route:
  + Destination: pl-xxxxxxx (S3 prefix list)
  + Target: Gateway endpoint

**🔹 Step 4: IAM Role or Policy for EC2**

Make sure the private EC2 instance has an IAM role or access key that allows S3 access:

**Example IAM Policy:**

json

CopyEdit

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": ["s3:GetObject", "s3:ListBucket"],

"Resource": ["arn:aws:s3:::your-bucket-name", "arn:aws:s3:::your-bucket-name/\*"]

}

]

}

**🔹 Step 5: Test from EC2 Private Instance**

SSH into your private EC2 and run:

bash

CopyEdit

aws s3 ls s3://your-bucket-name --region <your-region>

✅ If it works without NAT, you're successfully using the endpoint!

Interface endpoint

Enable https in sec group

Enable dns in instance action

And enable dns while configuring endpoint only first one

**Step 1: Create an Interface Endpoint for S3**

1. Go to **VPC Console → Endpoints → Create Endpoint**
2. **Service category**: AWS services
3. **Service name**: Filter for com.amazonaws.<region>.s3
4. **Endpoint type**: Choose **Interface**
5. **VPC**: Select your VPC
6. **Subnets**: Choose the **private subnet** where your EC2 instance resides
7. **Security Group**: Attach a security group that allows **HTTPS (TCP port 443) egress**
8. Click **Create Endpoint**

**✅ Step 2: Update Route Table (Not needed for interface endpoints)**

Unlike gateway endpoints, **interface endpoints** are based on **DNS + private IPs**, so **no route table updates** are needed.

**✅ Step 3: Configure Security Group for Interface Endpoint**

Ensure the security group attached to the endpoint **allows inbound HTTPS (port 443)** from your **private EC2's security group**.

**✅ Step 4: IAM Role for EC2 (Same as before)**

Attach a role to your EC2 instance with at least this IAM policy:

json

CopyEdit

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": ["s3:ListBucket", "s3:GetObject"],

"Resource": "\*"

}

]

}

Or use the AWS managed policy: AmazonS3ReadOnlyAccess.

**✅ Step 5: Test from EC2 Instance**

SSH into your private EC2 instance, and run:

bash

CopyEdit

aws s3 ls

You should see a list of buckets (if any), or create and access a new bucket:

bash

CopyEdit

aws s3 mb s3://your-bucket-name --region <region>

aws s3 ls s3://your-bucket-name

✅ This request will now go through the **interface endpoint's private IP**, not the public internet or NAT.

DAY-18

Step:1 two vpc or four or many

Step:2 create transit gateway

Step3: create attachments for the vpc’s

Step4**:**

To verify if your AWS Transit Gateway (TGW) is working correctly between VPC A and VPC B, follow these steps:

✅ Step-by-Step Verification

🔄 1. Verify TGW Attachments

* Go to the AWS VPC Console → Transit Gateway Attachments
* Check that both VPC A and VPC B attachments are in the available state.

📍 2. Verify Route Tables

In each VPC's route table, ensure:

* There's a route to the other VPC’s CIDR block
* The Target is the Transit Gateway

Example (for VPC A):

| Destination CIDR | Target |
| --- | --- |
| 10.1.0.0/16 (VPC B) | tgw-xxxxxxx |

Do the same in VPC B’s route table for VPC A.

📡 3. Launch EC2 Instances for Testing

* Launch one EC2 instance in VPC A
* Launch another EC2 instance in VPC B
* Place them in subnets attached to the TGW
* Make sure both instances:
  + Are in public or private subnets with proper routing
  + Have security group rules allowing ICMP (ping) or SSH

🔁 4. Test Connectivity

Option 1: Ping

* SSH into the EC2 instance in VPC A
* Run: ping <Private IP of EC2 in VPC B>

If ping is successful, TGW is routing traffic correctly.

Option 2: SSH

* Try: ssh ec2-user@<Private IP of EC2 in VPC B>

🔐 5. Check Security Groups & NACLs

If the ping or SSH doesn't work:

* Make sure Security Groups allow:
  + Inbound: ICMP or TCP/22 (for SSH)
  + Outbound: Allow all or to destination CIDR
* Network ACLs also must not block the traffic

🛠️ 6. Use VPC Reachability Analyzer (Optional)

* Go to VPC Console → Reachability Analyzer
* Create an analysis:
  + Source: EC2 instance in VPC A
  + Destination: EC2 instance in VPC B
* It will tell you if the TGW path is reachable or blocked

Create 2 vpc

1.aws 2.onprim vpc

Create 2 subnets -1.aws 2.onprim

Create 2 ec2 instances one on prim and one on aws vpc

Install strongswan in on premsis ec2

Sudo apt update

Sudo apt Install strongswan -y

-Create customer gateway

On Customer gatway mention on prim ip of public instance

-Create virtual private gateway and attach to ur aws vpc

Create by passing the name

Attach it to the aws vpc after creating

Creating vpn connection

-Attach the customer gateway

-Attach the virtual private gateway

Select the static under the routing option

Pass the on prim vpc subnet cider where the on prim ec2 is created

Download the vpn configuration

Select the strongswan in the provider section and click on download

Open the vpn config file

Connect to on prim ec2 instance

-Open /etc/sysctl.conf and uncomment the following line to enable packet forwarding net.ipv4.ip\_forward=1

-apply the changes in step1 by executing the command ‘sudo sysctl1 -p’

-create a new file at /etc/ipsec.conf uncommend uniquieid=no under config setup section

13.218.154.229 52.2.100.139 : PSK “”

**Create Two VPCs**

**VPC-A (AWS VPC)**

* CIDR: 196.128.0.0/24
* Subnet: 196.128.0.0/25
* Launch EC2 (e.g., Amazon Linux) — used for testing.

**VPC-B (On-Prem VPC)**

* CIDR: 20.24.0.0/24
* Subnet: 20.24.0.0/25
* Launch **Ubuntu EC2** — used to install StrongSwan.

**🟦 2. Enable IP Forwarding on "On-Prem" EC2 (Ubuntu)**

bash

CopyEdit

sudo apt update

sudo apt install strongswan -y

sudo vi /etc/sysctl.conf

# Uncomment or add:

net.ipv4.ip\_forward = 1

sudo sysctl -p

**🟦 3. Create a Virtual Private Gateway (VGW) for VPC-A**

* Go to **VPC → Virtual Private Gateways**
* Create VGW (give a name)
* Attach VGW to **VPC-A**

**🟦 4. Create a Customer Gateway (CGW) for On-Prem EC2**

* Go to **VPC → Customer Gateways**
* Create a new CGW:
  + Name: OnPrem-CGW
  + IP address: Public IP of EC2 in VPC-B
  + Routing: Static
  + BGP ASN: Leave default (unless using BGP)

**🟦 5. Create the VPN Connection**

* Go to **VPC → Site-to-Site VPN Connections**
* Create VPN Connection:
  + Target Gateway Type: **Virtual Private Gateway**
  + Attach the VGW (VPC-A)
  + Customer Gateway: **Existing**
  + Static Routing
  + Static CIDRs: 20.24.0.0/25 (On-Prem subnet)

**🟦 6. Update Route Tables**

**In VPC-A:**

* Go to **Route Tables** attached to VPC-A's subnet.
* Add a route:
  + Destination: 20.24.0.0/25
  + Target: **Virtual Private Gateway**

**In VPC-B (On-Prem):**

* You will manually add a route via ip route add (later).

**🟦 7. Download StrongSwan Configuration for On-Prem**

* In the VPN Connection page:
  + Select the VPN connection
  + Click **Download Configuration**
  + Choose **StrongSwan**
  + Save the file

**🟦 8. Configure StrongSwan on VPC-B EC2**

**Edit /etc/ipsec.conf**

bash

CopyEdit

sudo vi /etc/ipsec.conf

Append tunnel config from AWS config file:

ini

CopyEdit

config setup

uniqueids=no

conn Tunnel1

auto=start

left=%defaultroute

leftid=<public\_ip\_of\_on\_prem\_ec2>

right=<AWS\_VPN\_Outside\_IP>

type=tunnel

leftauth=psk

rightauth=psk

keyexchange=ikev1

ike=aes128-sha1-modp1024

ikelifetime=8h

esp=aes128-sha1-modp1024

lifetime=1h

keyingtries=%forever

leftsubnet=20.24.0.0/25

rightsubnet=196.128.0.0/25

dpddelay=10s

dpdtimeout=30s

dpdaction=restart

**Edit /etc/ipsec.secrets**

sudo vi /etc/ipsec.secrets

Add line from AWS config file:

text

CopyEdit

<onprem\_public\_ip> <aws\_vpn\_ip> : PSK "<shared\_secret>"

**🟦 9. Start the IPsec Tunnel**

bash

CopyEdit

sudo ipsec restart

sudo ipsec up Tunnel1

**🟦 10. Add Route in On-Prem EC2**

bash

CopyEdit

sudo ip route add 196.128.0.0/25 via <local\_gateway>

**🧪 11. Test the VPN**

* Ping AWS EC2 instance in 196.128.0.0/25 from On-Prem EC2
* Or the reverse from AWS EC2 to On-Prem EC2 in 20.24.0.0/25

**DAY-19**

PART 1: Create and Mount Data Volume on First EC2 Instance

Step 1: Launch EC2 Instance

1. Go to AWS Management Console → EC2 Dashboard.
2. Click Launch instance:
   * Name: Instance-1
   * AMI: Amazon Linux 2 (or Ubuntu)
   * Instance type: t2.micro
   * Key pair: Select or create one.
   * Network: Choose a VPC & Subnet (e.g., us-east-1a).
   * Storage: Root volume (default 8 GB).
   * Security Group: Allow SSH (port 22) from your IP.
3. Launch the instance.

Step 2: Create a Data Volume

1. Go to EC2 Dashboard → Elastic Block Store → Volumes.
2. Click Create Volume:
   * Size: e.g., 1 GiB
   * Volume type: General Purpose (gp2)
   * Availability Zone: Same as the instance (e.g., us-east-1a)
   * Leave the rest default.
3. Click Create Volume.

Step 3: Attach Data Volume to Instance-1

1. Go to the Volumes section.
2. Select the volume → Actions → Attach Volume.
3. Select Instance-1.
4. Device name (e.g., /dev/xvdf).
5. Click Attach.

Step 4: Connect to the Instance

1. Go to EC2 → Instances → Select Instance-1.
2. Click Connect → Choose EC2 Instance Connect or use SSH:

Step 5: Set up and Mount Volume

1. Check disks:

lsblk

You’ll see:

pgsql

CopyEdit

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

xvda 202:0 0 8G 0 disk /

xvdf 202:80 0 1G 0 disk

1. Create a file system on the new volume:

bash

CopyEdit

sudo mkfs -t xfs /dev/xvdf

1. Create a mount point:

bash

CopyEdit

sudo mkdir /ust

1. Mount the volume:

bash

CopyEdit

sudo mount /dev/xvdf /ust

1. Verify with lsblk:

bash

CopyEdit

lsblk

1. Write a file into mounted volume:

bash

CopyEdit

cd /ust

echo "hii" | sudo tee file.txt

cat file.txt

You should see:

nginx

CopyEdit

hii

✅ PART 2: Detach and Reattach Volume to 2nd Instance

Step 6: Launch Second Instance

1. Launch a new EC2 instance: Name it Instance-2.
   * Use same subnet/zone (e.g., us-east-1a).
   * Allow SSH.
   * Launch.

Step 7: Detach Volume from First Instance

1. Go to EC2 → Volumes → Select your volume.
2. Click Actions → Detach Volume.
3. Wait for state to be "available".

Step 8: Attach Volume to Second Instance

1. Select the detached volume → Actions → Attach Volume.
2. Select Instance-2 and device as /dev/xvdf (or similar).
3. Click Attach.

Step 9: Connect to Instance-2

1. SSH or EC2 Connect:

bash

CopyEdit

ssh -i your-key.pem ec2-user@<instance-2-ip>

1. Check volume:

bash

CopyEdit

lsblk

You should see:

nginx

CopyEdit

xvda 8G

xvdf 1G

Step 10: Mount the Volume and View File

1. Create mount point:

bash

CopyEdit

sudo mkdir /ust

1. Mount the volume:

bash

CopyEdit

sudo mount /dev/xvdf /ust

1. List contents:

bash

CopyEdit

ls /ust

You should see:

CopyEdit

file.txt

1. View file contents:

bash

CopyEdit

cat /ust/file.txt

Output:

nginx

CopyEdit

hii

✅ Summary of Commands Used:

bash

CopyEdit

# On Instance-1:

lsblk

sudo mkfs -t xfs /dev/xvdf

sudo mkdir /ust

sudo mount /dev/xvdf /ust

cd /ust

echo "hii" | sudo tee file.txt

cat file.txt

# On Instance-2:

lsblk

sudo mkdir /ust

sudo mount /dev/xvdf /ust

ls /ust

cat /ust/file.txt

Let me know if you want this in a bash script or a PDF!

**DAYXZ**