Adv DevOps Case Study

My case study topic is as follows:-

Infrastructure as Code with Terraform

- Concepts Used: Terraform, AWS S3, and EC2.
- Problem Statement: "Use Terraform to provision an AWS EC2 instance and an S3 bucket. Deploy a sample static website on the S3 bucket using the EC2 instance as the backend server."
- Tasks:
    - Write a Terraform script to create an EC2 instance and an S3 bucket.
    - Deploy the static website on the S3 bucket.
    - Use the EC2 instance to interact with the S3 bucket and log the actions.

Introduction to Infrastructure as Code (IaC) with Terraform

Infrastructure as Code (IaC) allows the automation of infrastructure deployment and management using code, offering efficiency, repeatability, and version control. Terraform is one of the most widely used IaC tools, allowing developers to define and manage infrastructure in cloud environments such as AWS, Google Cloud, and Azure. Terraform uses declarative configuration files that describe the desired state of the infrastructure, enabling consistent provisioning across environments. In this scenario, you are asked to use Terraform to provision two AWS services: an EC2 instance and an S3 bucket, and to deploy a static website on the S3 bucket while using the EC2 instance to interact with the bucket as a backend server.

Significance of Using Terraform in AWS

1. Automation: With Terraform, cloud resources can be created, modified, or destroyed automatically, reducing the need for manual intervention.
2. Version Control: The infrastructure code can be versioned and maintained in repositories, enabling teams to track changes, roll back configurations, and collaborate effectively.
3. Consistency: Infrastructure can be provisioned consistently across different environments, reducing configuration drift and ensuring that development, testing, and production environments are identical.
4. Multi-Cloud Support: Terraform provides the flexibility to manage resources not only on AWS but also across multiple cloud providers with a single tool.

Key Concepts and Services Involved are as follows:-

Terraform

- Significance: Terraform automates the provisioning and management of cloud resources. It allows organizations to treat their infrastructure as code, providing benefits like automation, collaboration, and consistency.

AWS EC2 (Elastic Compute Cloud)

- Significance: In this context, the EC2 instance will serve as the backend server that can interact with the S3 bucket. It will log the actions related to the deployment and management of the static website hosted on S3.
- Applications: EC2 is used for running web applications, backend services, databases, and other computing tasks. In this project, it plays a crucial role in managing and logging the interactions with the S3 bucket.

AWS S3 (Simple Storage Service)

- Significance: The S3 bucket will host the static website. S3 provides high availability and content delivery, making it a suitable platform for website hosting.
- Applications: AWS S3 is widely used for static website hosting, backup storage, big data analytics, and content distribution.

Static Website Hosting on S3

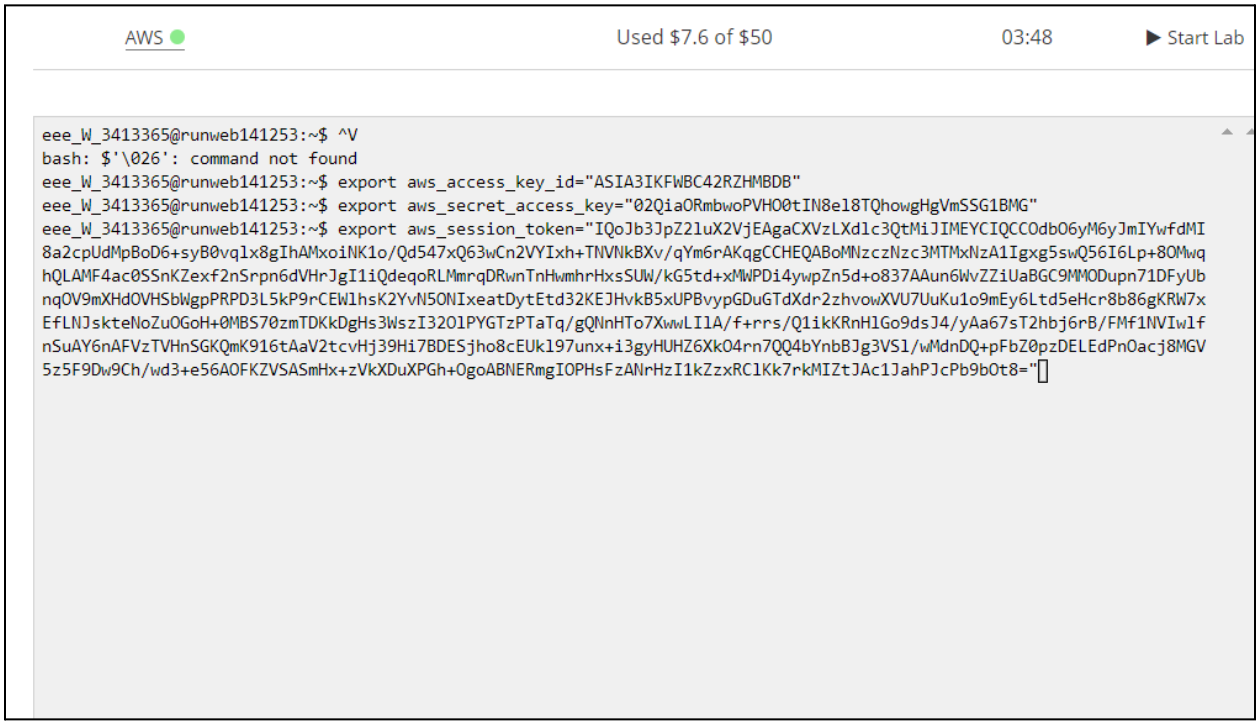
- Significance: Using S3 to host static websites is cost-effective and scalable. By leveraging this, the project eliminates the need for a dedicated web server, offloading the hosting duties to S3.
- Applications: This is useful for content-heavy websites, landing pages, and blogs that do not require server-side processing.

Applications of the Solution are as follows:-

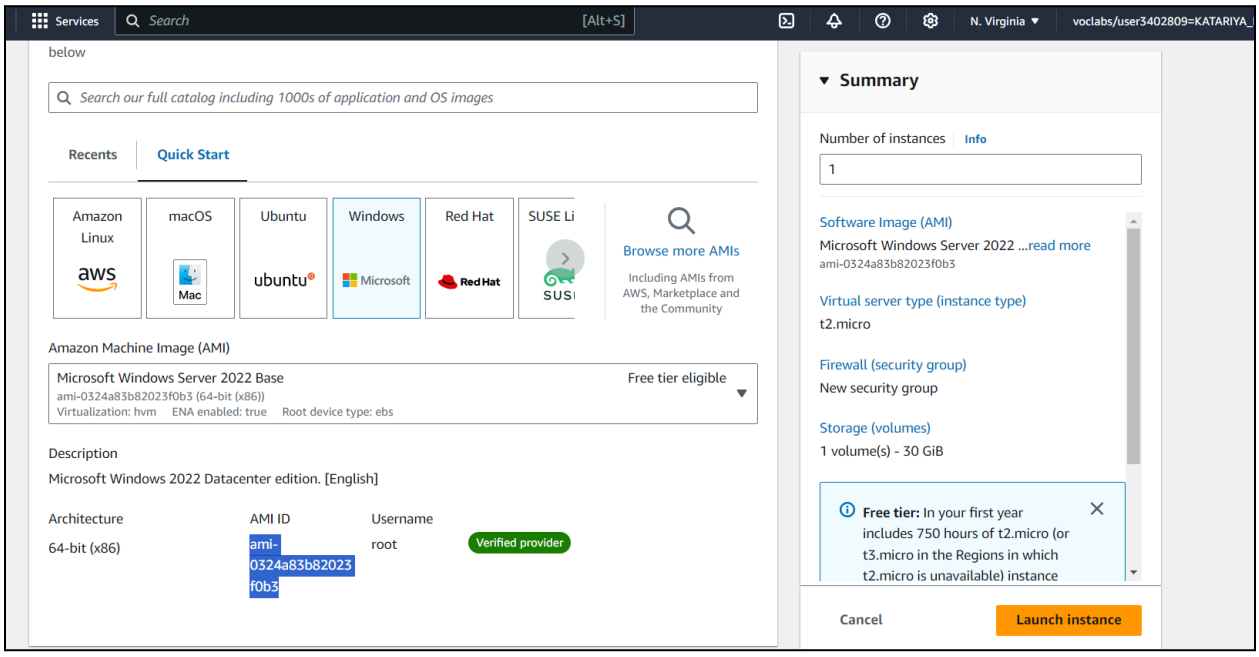
1. Infrastructure Automation: The ability to automate infrastructure provisioning for websites, applications, or backend services significantly reduces deployment time and errors.
2. Scalable Website Hosting: Hosting static websites on S3 provides a scalable and highly available solution for website deployment. This is especially useful for static content, where high performance and low costs are critical.
3. Log Management and Monitoring: Using EC2 to interact with the S3 bucket for logging purposes provides deeper insight into website management. This setup can be extended to gather and analyze logs, track usage, or monitor performance metrics.
4. Collaboration and Versioning: By using Terraform, teams can collaborate effectively with shared infrastructure definitions, enabling version control of infrastructure similar to how they manage application code.

Implementation:

This is the configuration:



This is the AMI_ID:



The main.tf file:

```
terraform {
```

```
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.64.0"
    }
  }
}

resource "aws_instance" "myserver" {
    ami = "ami-0324a83b82023f0b3"
    instance_type = "var.instance_type"
    tags = {
      Name="SampleServer"
    }
}
```

The provider.tf file:

```
provider "aws" {
  access_key="ASIA3IKFWBC42RZHMBDB"
  secret_key="02QiaORmbwoPVHO0tIN8el8TQhowgHgVmSSG1BMG"

token="IQoJb3JpZ2luX2VjEAgaCXVzLXdlc3QtMiJIMEYCIQCCOdbO6yM6yJmIYwfdMI8a2cpU
dMpBoD6+syB0vqlx8gIhAMxoiNK1o/Qd547xQ63wCn2VYIxh+TNVNkBXv/qYm6rAKqgCCHEQ
ABoMNzczNzc3MTMxNzA1Igxg5swQ56I6Lp+8OMwqhQLAMF4ac0SSnKZexf2nSrpn6dVHrJgI
1iQdeqoRLMmrqDRwnTnHwmhrHxsSUW/kG5td+xMWPDi4ywpZn5d+o837AAun6WvZZiUaBG
C9MMODupn71DFyUbnqOV9mXHdOVHSbWgpPRPD3L5kP9rCEWlhsK2YvN5ONIxeatDytEtd
32KEJHvkB5xUPBvypGDuGTdXdr2zhvowXVU7UuKu1o9mEy6Ltd5eHcr8b86gKRW7xEfLNJskt
eNoZuOGoH+0MBS70zmTDKkDgHs3WszI32OIPYGTzPTaTq/gQNnHTo7XwwLIIA/f+rrs/Q1ikK
RnHlGo9dsJ4/yAa67sT2hbj6rB/FMf1NVIwlfnSuAY6nAFVzTVHnSGKQmK916tAaV2tcvHj39Hi7
BDESjho8cEUkl97unx+i3gyHUHZ6XkO4rn7QQ4bYnbBJg3VSl/wMdnDQ+pFbZ0pzDELEdPnO
acj8MGV5z5F9Dw9Ch/wd3+e56AOFKZVSASmHx+zVkXDuXPGh+OgoABNERmgIOPHsFzANr
HzI1kZzxRClKk7rkMIZtJAc1JahPJcPb9bOt8="
  region = "us-east-1"
}
```

This is the ss of Terraform init:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

PS C:\Users\272241\FinalAWS> cd .\aws-ec2\
PS C:\Users\272241\FinalAWS\aws-ec2> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.64.0"...
- Installing hashicorp/aws v5.64.0...
- Installed hashicorp/aws v5.64.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Users\272241\FinalAWS\aws-ec2>
```

After running Terraform plan:

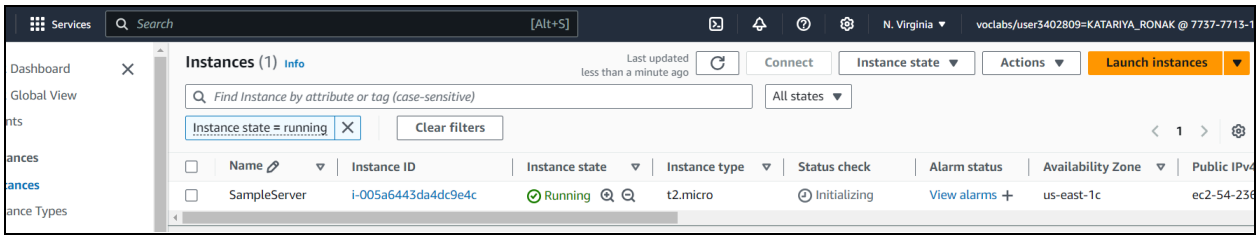
```
PS C:\Users\272241\FinalAWS\aws-ec2> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indic
  + create

Terraform will perform the following actions:

  # aws_instance.myserver will be created
  + resource "aws_instance" "myserver" {
      + ami                                  = "ami-0324a83b82023f0b3"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
```

After Terraform apply:

```
PS C:\Users\272241\FinalAWS\aws-ec2> terraform apply

Terraform used the selected providers to generate the following execution plan. Resource acti
  + create

Terraform will perform the following actions:

  # aws_instance.myserver will be created
  + resource "aws_instance" "myserver" {
      + ami                                  = "ami-0324a83b82023f0b3"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
```

```
      + metadata_options (known after apply)

      + network_interface (known after apply)

      + private_dns_name_options (known after apply)

      + root_block_device (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.myserver: Creating...
aws_instance.myserver: Still creating... [10s elapsed]
aws_instance.myserver: Creation complete after 16s [id=i-005a6443da4dc9e4c]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

After successfully checking it on aws :



S3 bucket creation:

Again take the credentials from the aws and paste it in the aws cli:

Then make a new file named aws-s3:
Then create a main.tf file with the following code:

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.64.0"
    }
     random = {
      source = "hashicorp/random"
      version = "3.6.2"
    }
  }
}
```

```
resource "random_id" "ran_id" {
  byte_length = 8
}
resource "aws_s3_bucket" "demo-bucket" {
  bucket = "my-demo-bucket-${random_id.ran_id.hex}"
}

resource "aws_s3_object" "bucket-data" {
  bucket = aws_s3_bucket.demo-bucket.bucket
  source = "./myfile.txt"
  key = "newfile.txt"

}
```

Then create another file named provider.tf and paste the following code :

```
provider "aws" {
  access_key="ASIA3IKFWBC4ZOZYH36Z"
  secret_key="PCGz8WLykK8K8ECeV7QCAA+AHLpYYF+8KBBGhS6Q"

token="IQoJb3JpZ2luX2VjEA4aCXVzLXdlc3QtMiJHMEUCIQD0YP9lVmHMONLtFlra0JBKj5KR
V4ekOvi7IrcNcwKXHwIgIU/twSs1G2aePplubrKLz7tAV7j3QLLACU3wX6sgUWAqqAIIdxAAGgw
3NzM3NzcxMzE3MDUiDNReKUavH2Cqr/z25yqFAphb3xwo/+Cv0H8/P4CXO0nM+Wn1RSFtm9
VXPxeqico3LkWNyHShigie0WUkjcTXQjnLHVEgfQZs9j08A39T7iAUU4P9uPuR1ZN7gFe6c/Q0
EOx8VdIrMqWjhkrbwNRFmz1yevbJJfZH+iNwz7RSRGe1Tj4FKQmDJexdgr0ngAxCXB8tjVCivF
8g5ZqRqVdMp8LW8ujhO3gX68Or+ARzjBI0JwmBWnp6zrEfDDFk549yLNIIgzT/Nh8zM+QHxBg
BTxOtEPwoLdPrY6Ecs2XLcqsJRrsCdy1bIDy7Y63XR9J9DWtjrdNr9Q5BgvTTmzxSndoASCoY
USBWozBAIHEhz84s455JEzCrjNS4BjqdAZ+GIn07AAWyDqUdvX/e4419ixJOGEtp+wBapt2atttT
3zxPOpeOi2lWvdLLejq5TOKZo61GpOcGK7bb9d9RLbPznBcNHaIrK1gXxm2stsELW3xntEdCb
3UNK8GlDzFsEfcwUpewwzMh1xs8GE+UFIHszuv2xnu4/O2N+ta2K1P0+MaoCApugpd0siHTsp
UfE/4jZipp6VVpzi+I9+w="
  region = "us-east-1"
}
```

Then again do Terraform init:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS        powershell - av

PS C:\Users\272241\FinalAWS\aws-s3> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/random versions matching "3.6.2"...
- Finding hashicorp/aws versions matching "5.64.0"...
- Installing hashicorp/aws v5.64.0...
- Installed hashicorp/aws v5.64.0 (signed by HashiCorp)
- Installing hashicorp/random v3.6.2...
- Installed hashicorp/random v3.6.2 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

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

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

Then perform Terraform plan:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Users\272241\FinalAWS\aws-s3> Terraform plan

Terraform used the selected providers to generate the following execution plan. Resou
  + create

Terraform will perform the following actions:

  # aws_s3_bucket.demo-bucket will be created
  + resource "aws_s3_bucket" "demo-bucket" {
      + acceleration_status         = (known after apply)
      + acl                         = (known after apply)
      + arn                         = (known after apply)
      + bucket                      = (known after apply)
      + bucket_domain_name          = (known after apply)
      + bucket_prefix               = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + object_lock_enabled         = (known after apply)
      + policy                      = (known after apply)
```

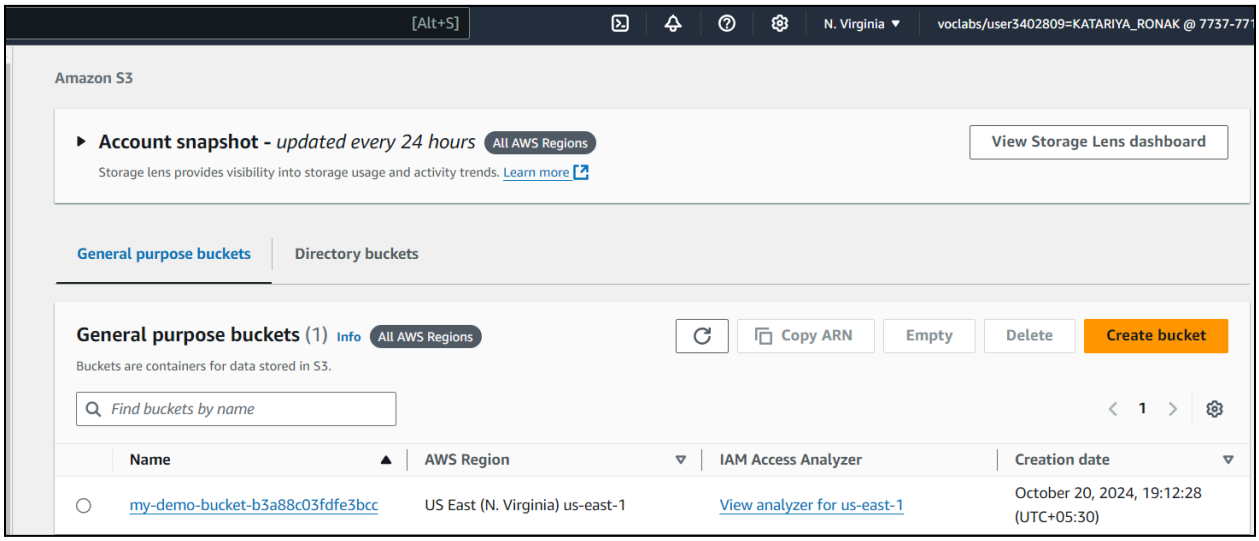
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS                powershell

      + server_side_encryption = (known after apply)
      + source                 = "./myfile.txt"
      + storage_class          = (known after apply)
      + tags_all               = (known after apply)
      + version_id             = (known after apply)
    }

  # random_id.ran_id will be created
  + resource "random_id" "ran_id" {
      + b64_std     = (known after apply)
      + b64_url     = (known after apply)
      + byte_length = 8
      + dec         = (known after apply)
      + hex         = (known after apply)
      + id          = (known after apply)
    }

Plan: 3 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these acti
apply" now.
```

Perform terraform apply :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

PS C:\Users\272241\FinalAWS\aws-s3> Terraform apply
random_id.ran_id: Refreshing state... [id=s6iMA_3-O8w]
aws_s3_bucket.demo-bucket: Refreshing state... [id=my-demo-bucket-b3a88c03fdfe3bcc]

Terraform used the selected providers to generate the following execution plan. Resource actions a
  + create

Terraform will perform the following actions:

  # aws_s3_object.bucket-data will be created
  + resource "aws_s3_object" "bucket-data" {
      + acl                    = (known after apply)
      + arn                    = (known after apply)
      + bucket                 = "my-demo-bucket-b3a88c03fdfe3bcc"
      + bucket_key_enabled     = (known after apply)
      + checksum_crc32         = (known after apply)
      + checksum_crc32c        = (known after apply)
      + checksum_sha1          = (known after apply)
      + checksum_sha256        = (known after apply)
      + content_type           = (known after apply)
      + etag                   = (known after apply)
      + force_destroy          = false
      + id                     = (known after apply)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENT

      + content_type          = (known after apply)
      + etag                  = (known after apply)
      + force_destroy         = false
      + id                    = (known after apply)
      + key                   = "newfile.txt"
      + kms_key_id            = (known after apply)
      + server_side_encryption = (known after apply)
      + source                = "./myfile.txt"
      + storage_class         = (known after apply)
      + tags_all              = (known after apply)
      + version_id            = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_s3_object.bucket-data: Creating...
aws_s3_object.bucket-data: Creation complete after 2s [id=newfile.txt]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Now check AWS to see the bucket created:



Now comes the static hosting part with s3 bucket by using ec2 instance:

Step 1 : create main.tf and write following code
Code -

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.64.0"
    }
     random = {
      source = "hashicorp/random"
      version = "3.6.2"
    }
  }
}

resource "random_id" "ran_id" {
  byte_length = 8
}
resource "aws_s3_bucket" "mywebapp-bucket" {
  bucket = "my-mywebapp-bucket-${random_id.ran_id.hex}"
}

resource "aws_s3_object" "index_html" {
  bucket = aws_s3_bucket.mywebapp-bucket.bucket
  source = "./index.html"
  key = "index.html"
  content_type = "text/html"

}

resource "aws_s3_object" "style_css" {
  bucket = aws_s3_bucket.mywebapp-bucket.bucket
  source = "./styles.css"
  key = "styles.css"
  content_type = "text/css"

}
```

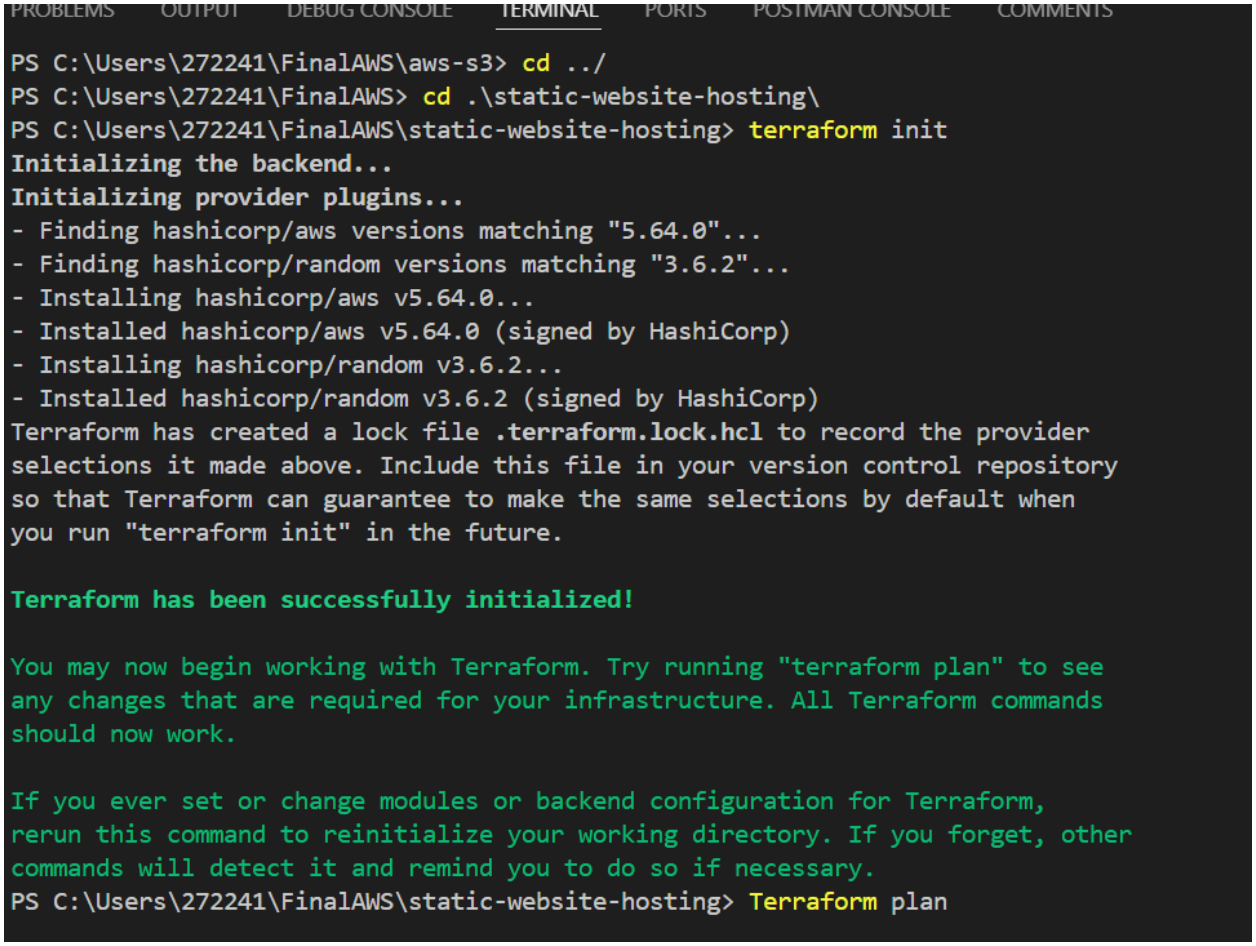
```
resource "aws_s3_bucket_public_access_block" "example" {
  bucket = aws_s3_bucket.mywebapp-bucket.id
  block_public_acls       = false
  block_public_policy     = false
  ignore_public_acls      = false
  restrict_public_buckets = false
}

resource "aws_s3_bucket_policy" "staticwebnew" {
   bucket = aws_s3_bucket.mywebapp-bucket.id
    policy = jsonencode(
       {
    Version = "2012-10-17",
    Statement = [
       {
          Sid = "PublicReadGetObject",
          Effect = "Allow",
          Principal = "*",
          Action = "s3:GetObject",
          Resource = "arn:aws:s3:::${aws_s3_bucket.mywebapp-bucket.id}/*"
       }
    ]
}
    )
}

resource "aws_s3_bucket_website_configuration" "example" {
  bucket = aws_s3_bucket.mywebapp-bucket.id

  index_document {
    suffix = "index.html"
  }
}

output "website_endpoint" {
  value = aws_s3_bucket_website_configuration.example.website_endpoint
```

```
}
```

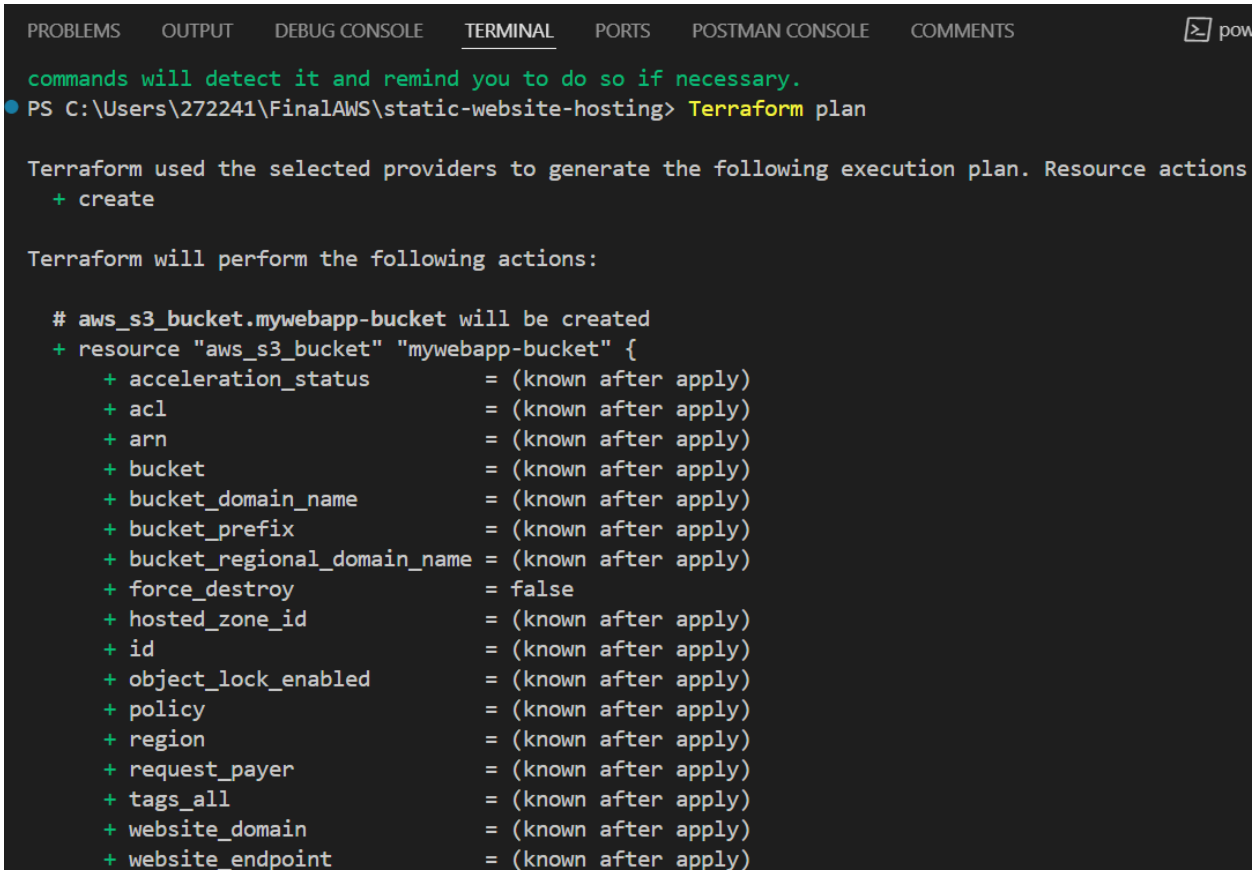
Step 2 : Create Provider.tf and write following code
Code:

```
provider "aws" {
  access_key="ASIA3IKFWBC4ZOZYH36Z"
  secret_key="PCGz8WLykK8K8ECeV7QCAA+AHLpYYF+8KBBGhS6Q"

token="IQoJb3JpZ2luX2VjEA4aCXVzLXdlc3QtMiJHMEUCIQD0YP9lVmHMONLtFlra0JBKj5KR
V4ekOvi7IrcNcwKXHwIgIU/twSs1G2aePplubrKLz7tAV7j3QLLACU3wX6sgUWAqqAIIdxAAGgw
3NzM3NzcxMzE3MDUiDNReKUavH2Cqr/z25yqFAphb3xwo/+Cv0H8/P4CXO0nM+Wn1RSFtm9
VXPxeqico3LkWNyHShigie0WUkjcTXQjnLHVEgfQZs9j08A39T7iAUU4P9uPuR1ZN7gFe6c/Q0
EOx8VdIrMqWjhkrbwNRFmz1yevbJJfZH+iNwz7RSRGe1Tj4FKQmDJexdgr0ngAxCXB8tjVCivF
8g5ZqRqVdMp8LW8ujhO3gX68Or+ARzjBI0JwmBWnp6zrEfDDFk549yLNIIgzT/Nh8zM+QHxBg
BTxOtEPwoLdPrY6Ecs2XLcqsJRrsCdy1bIDy7Y63XR9J9DWtjrdNr9Q5BgvTTmzxSndoASCoY
USBWozBAIHEhz84s455JEzCrjNS4BjqdAZ+GIn07AAWyDqUdvX/e4419ixJOGEtp+wBapt2atttT
3zxPOpeOi2IWvdLLejq5TOKZo61GpOcGK7bb9d9RLbPznBcNHaIrK1gXxm2stsELW3xntEdCb
3UNK8GlDzFsEfcwUpewwzMh1xs8GE+UFIHszuv2xnu4/O2N+ta2K1P0+MaoCApugpd0siHTsp
UfE/4jZipp6VVpzi+I9+w="
  region = "us-east-1"
}
```

Step 3: Execute Terraform init command.



Step 4: Terraform plan and terraform apply:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
POSTMAN CONSOLE
COMMENTS
commands will detect it and remind you to do so if necessary.
PS C:\Users\272241\FinalAWS\static-website-hosting> Terraform plan
Terraform used the selected providers to generate the following execution plan. Resource actions
+ create
Terraform will perform the following actions:
# aws_s3_bucket.mywebapp-bucket will be created
+ resource "aws_s3_bucket" "mywebapp-bucket" {
+ acceleration_status = (known after apply)
+ acl = (known after apply)
+ arn = (known after apply)
+ bucket = (known after apply)
+ bucket_domain_name = (known after apply)
+ bucket_prefix = (known after apply)
+ bucket_regional_domain_name = (known after apply)
+ force_destroy = false
+ hosted_zone_id = (known after apply)
+ id = (known after apply)
+ object_lock_enabled = (known after apply)
+ policy = (known after apply)
+ region = (known after apply)
+ request_payer = (known after apply)
+ tags_all = (known after apply)
+ website_domain = (known after apply)
+ website_endpoint = (known after apply)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE COMMEN

```
    }

  # aws_s3_bucket_policy.staticwebnew will be created
  + resource "aws_s3_bucket_policy" "staticwebnew" {
      + bucket = (known after apply)
      + id     = (known after apply)
      + policy = (known after apply)
    }

  # aws_s3_bucket_public_access_block.example will be created
  + resource "aws_s3_bucket_public_access_block" "example" {
      + block_public_acls       = false
      + block_public_policy     = false
      + bucket                  = (known after apply)
      + id                      = (known after apply)
      + ignore_public_acls      = false
      + restrict_public_buckets = false
    }

  # aws_s3_bucket_website_configuration.example will be created
  + resource "aws_s3_bucket_website_configuration" "example" {
      + bucket           = (known after apply)
      + id               = (known after apply)
      + routing_rules    = (known after apply)
      + website_domain   = (known after apply)
      + website_endpoint = (known after apply)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE COMMENTS

```
      + index_document {
          + suffix = "index.html"
        }

      + routing_rule (known after apply)
    }

  # aws_s3_object.index_html will be created
  + resource "aws_s3_object" "index_html" {
      + acl                    = (known after apply)
      + arn                    = (known after apply)
      + bucket                 = (known after apply)
      + bucket_key_enabled     = (known after apply)
      + checksum_crc32         = (known after apply)
      + checksum_crc32c        = (known after apply)
      + checksum_sha1          = (known after apply)
      + checksum_sha256        = (known after apply)
      + content_type           = "text/html"
      + etag                   = (known after apply)
      + force_destroy          = false
      + id                     = (known after apply)
      + key                    = "index.html"
      + kms_key_id             = (known after apply)
      + server_side_encryption = (known after apply)
      + source                 = "./index.html"
      + storage_class          = (known after apply)
      + tags_all               = (known after apply)
      + version_id             = (known after apply)
    }

  # aws_s3_object.style_css will be created
  + resource "aws_s3_object" "style_css" {
```

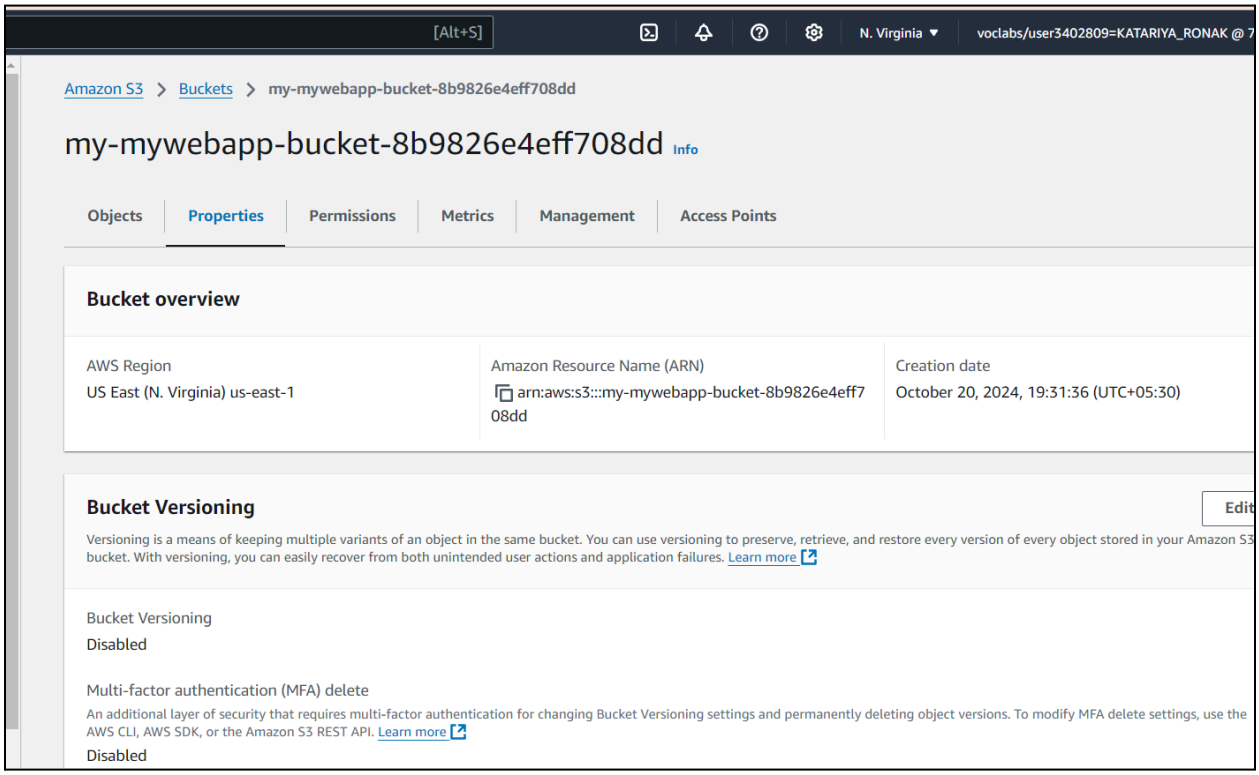
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE COMMENTS

```
      + checksum_crc32c         = (known after apply)
      + checksum_sha1           = (known after apply)
      + checksum_sha256         = (known after apply)
      + content_type            = "text/css"
      + etag                    = (known after apply)
      + force_destroy           = false
      + id                      = (known after apply)
      + key                     = "styles.css"
      + kms_key_id              = (known after apply)
      + server_side_encryption = (known after apply)
      + source                  = "./styles.css"
      + storage_class           = (known after apply)
      + tags_all                = (known after apply)
      + version_id              = (known after apply)
    }

  # random_id.ran_id will be created
  + resource "random_id" "ran_id" {
      + b64_std     = (known after apply)
      + b64_url     = (known after apply)
      + byte_length = 8
      + dec         = (known after apply)
      + hex         = (known after apply)
      + id          = (known after apply)
    }

Plan: 7 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + website_endpoint = (known after apply)
```

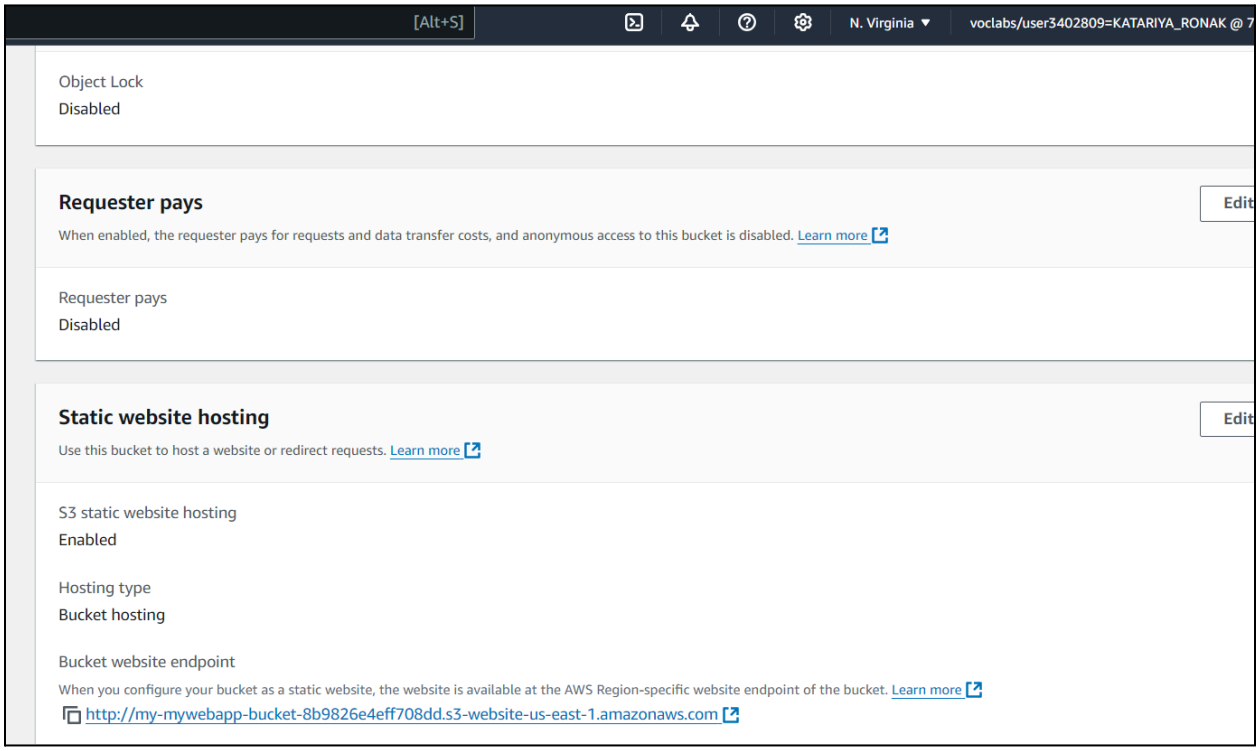
```
apply" now.
PS C:\Users\272241\FinalAWS\static-website-hosting> Terraform apply

Terraform used the selected providers to generate the following execution plan. Resou
  + create

Terraform will perform the following actions:

  # aws_s3_bucket.mywebapp-bucket will be created
  + resource "aws_s3_bucket" "mywebapp-bucket" {
      + acceleration_status         = (known after apply)
      + acl                         = (known after apply)
      + arn                         = (known after apply)
      + bucket                      = (known after apply)
      + bucket_domain_name          = (known after apply)
      + bucket_prefix               = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + object_lock_enabled         = (known after apply)
      + policy                      = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags_all                    = (known after apply)
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)
```

```
Changes to Outputs:
  + website_endpoint = (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

random_id.ran_id: Creating...
random_id.ran_id: Creation complete after 0s [id=i5gm5O_3CN0]
aws_s3_bucket.mywebapp-bucket: Creating...
aws_s3_bucket.mywebapp-bucket: Creation complete after 5s [id=my-mywebapp-bucket-8b9826e4eff708dd]
aws_s3_object.style_css: Creating...
aws_s3_object.index_html: Creating...
aws_s3_bucket_public_access_block.example: Creating...
aws_s3_bucket_website_configuration.example: Creating...
aws_s3_bucket_policy.staticwebnew: Creating...
aws_s3_bucket_public_access_block.example: Creation complete after 1s [id=my-mywebapp-bucket-8b9826e4eff708dd]
aws_s3_object.index_html: Creation complete after 1s [id=index.html]
aws_s3_bucket_website_configuration.example: Creation complete after 1s [id=my-mywebapp-bucket-8b9826e4eff708dd]
aws_s3_object.style_css: Creation complete after 1s [id=styles.css]
aws_s3_bucket_policy.staticwebnew: Creation complete after 1s [id=my-mywebapp-bucket-8b9826e4eff708dd]

Apply complete! Resources: 7 added, 0 changed, 0 destroyed.

Outputs:

website_endpoint = "my-mywebapp-bucket-8b9826e4eff708dd.s3-website-us-east-1.amazonaws.com"
PS C:\Users\272241\FinalAWS\static-website-hosting>
```

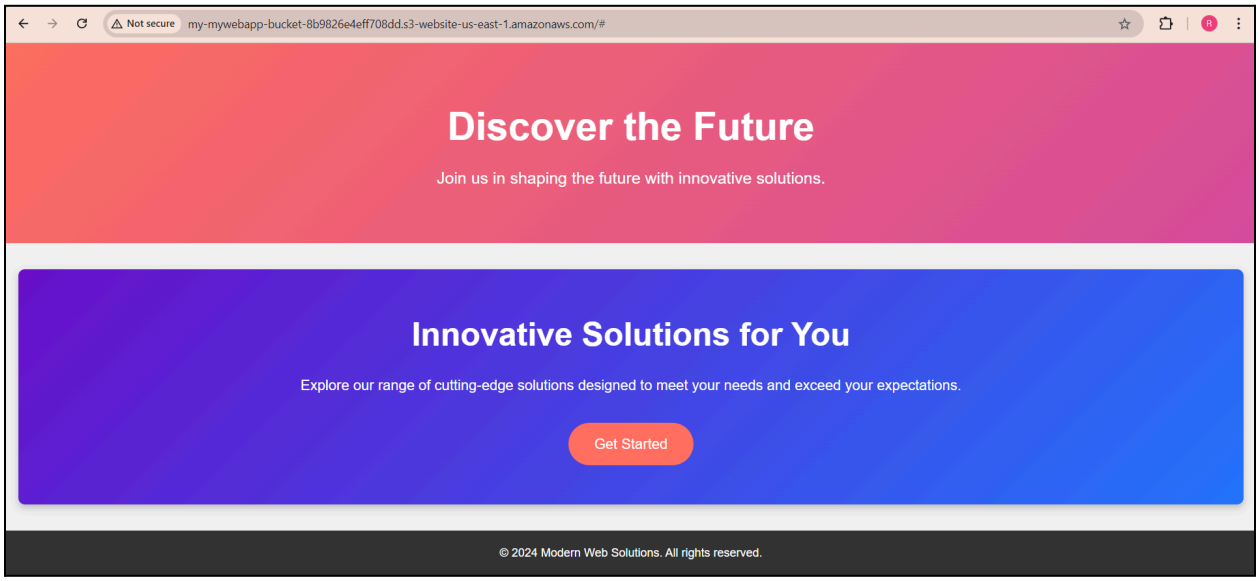
After this go to the bucket properties :



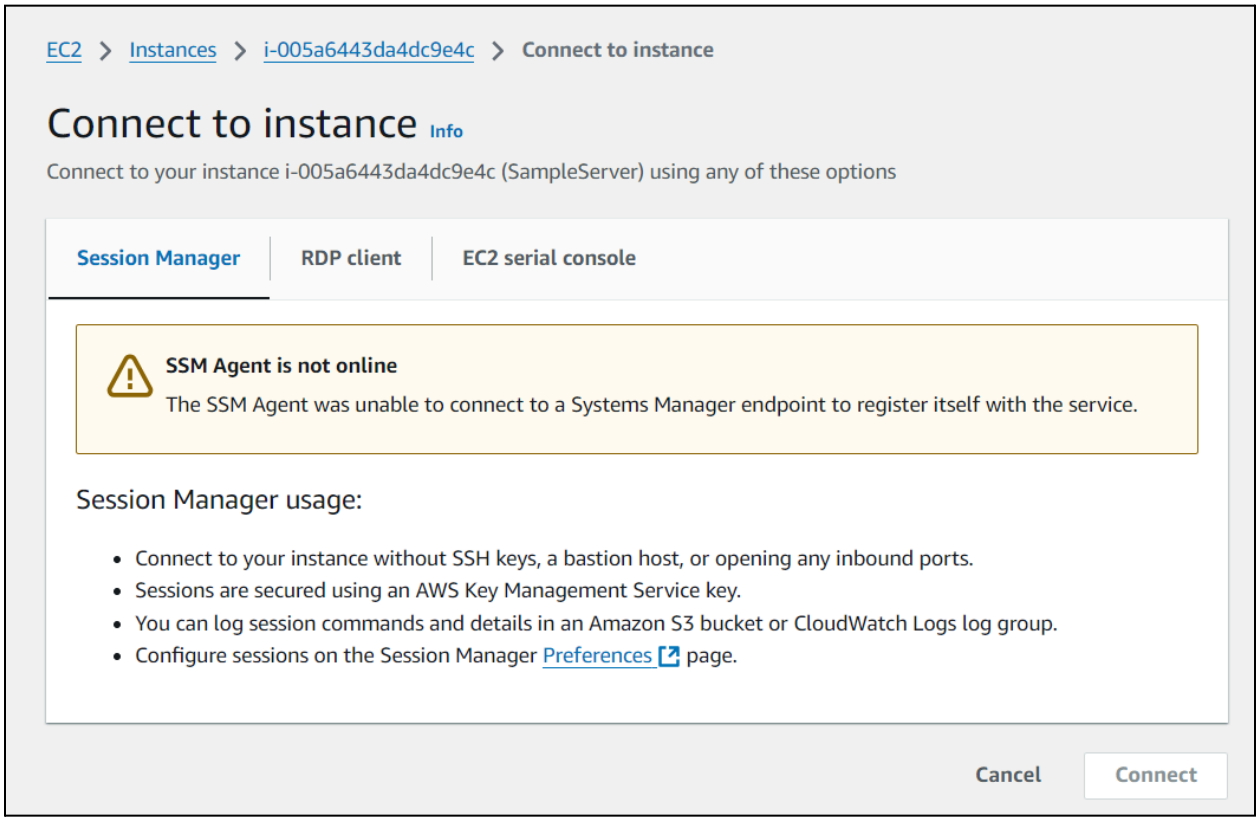
Scroll down :



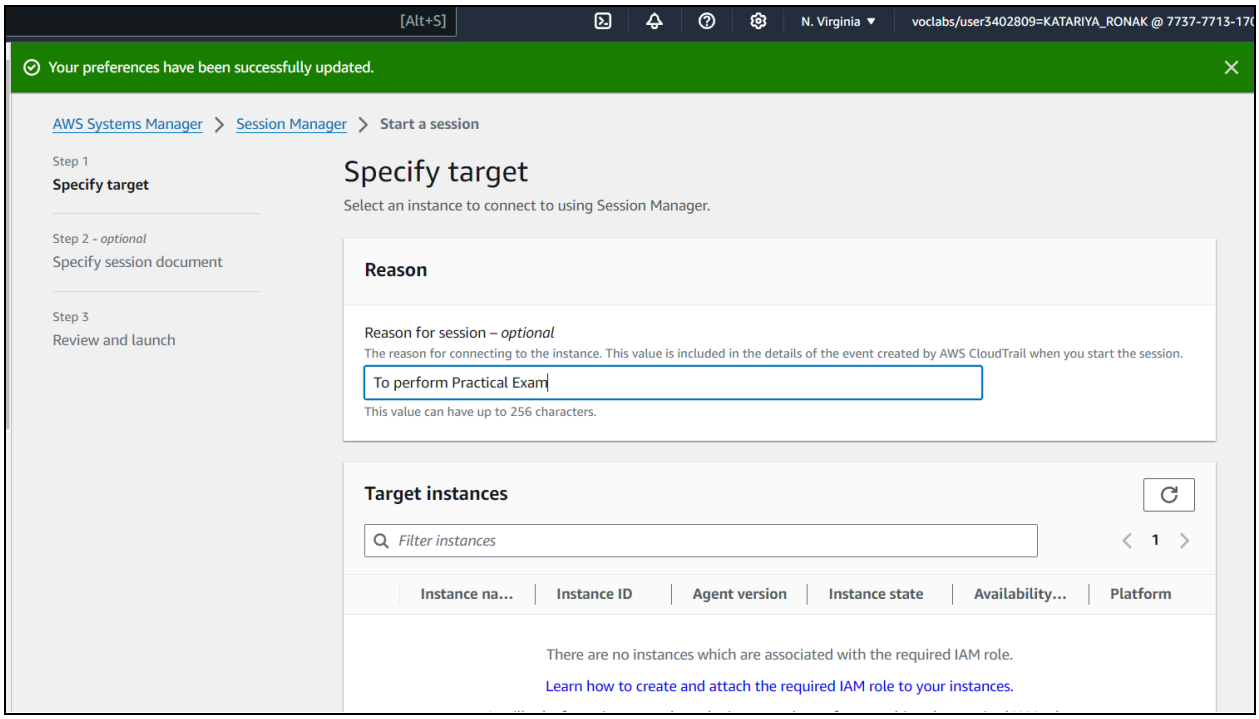
Click on the url displayed there:



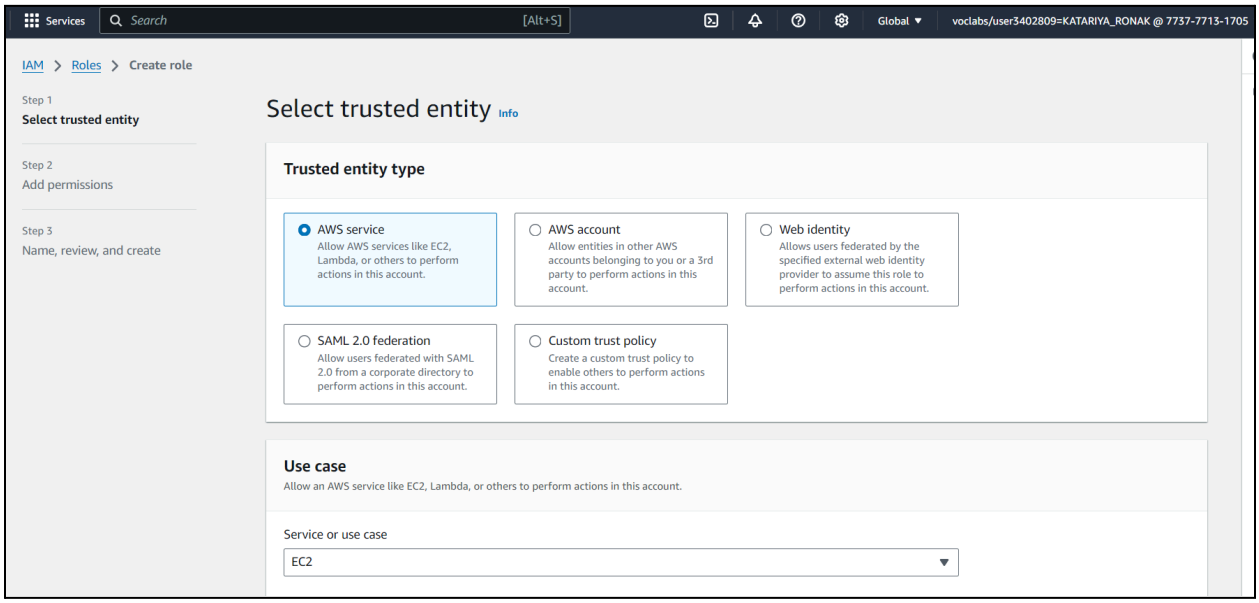
Now comes an important step;



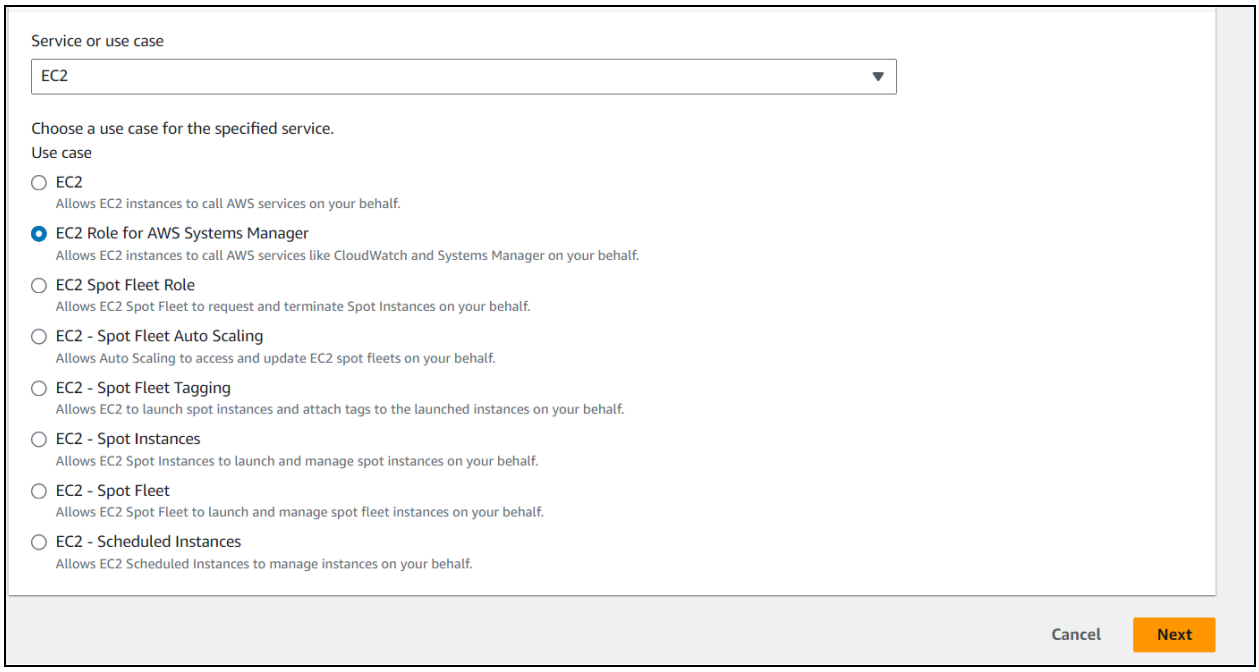
So we can see that the ec2 instance cant be connected, so for that we will have to do the following:



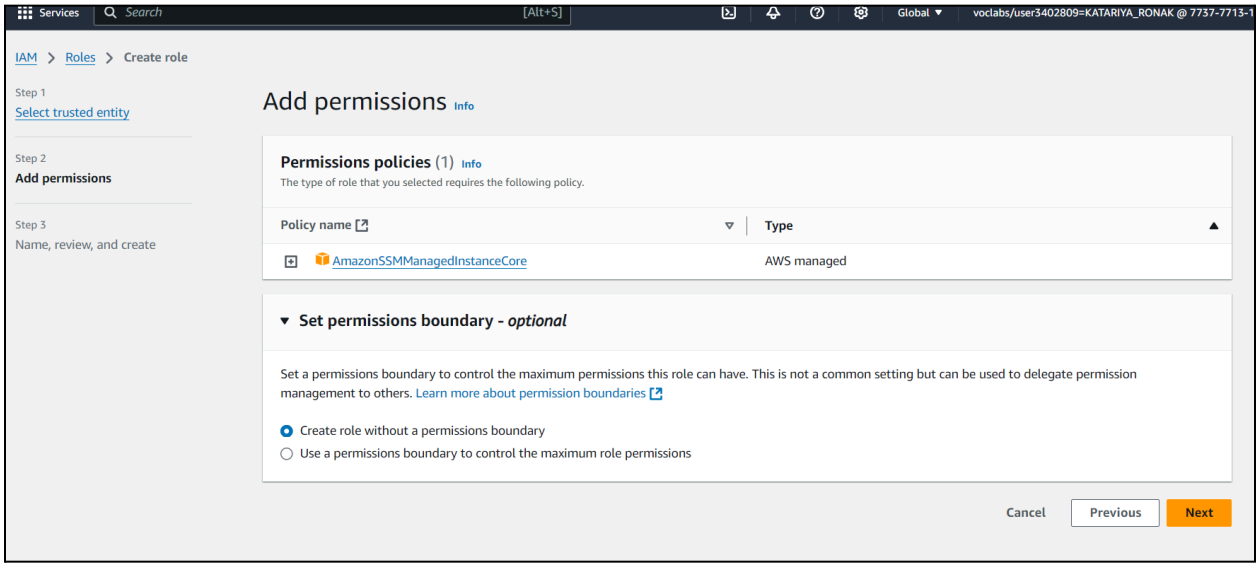
Create a new Role:



Select EC2 role for AWS Systems Manager:



Now add permission :



Now started with the log actions:

Connect the instance and then start performing the log actions:

```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Wed Oct 23 04:57:57 2024 from 18.237.140.163
```

List all objects in the S3 bucket:

```
[ec2-user@ip-172-31-23-146 ~]$ aws s3 ls s3://my-static-website-bucket-unique123/
2024-10-22 15:07:47       2834 index.html
```

Log the action into a file:

aws s3 ls s3://your-bucket-name/ > s3_logs.txt
This will create a file named s3_logs.txt with the output of the aws s3 ls command.

echo "Test log file" > testfile.txt
aws s3 cp testfile.txt s3://your-bucket-name/
aws s3 cp testfile.txt s3://your-bucket-name/ > upload_log.txt

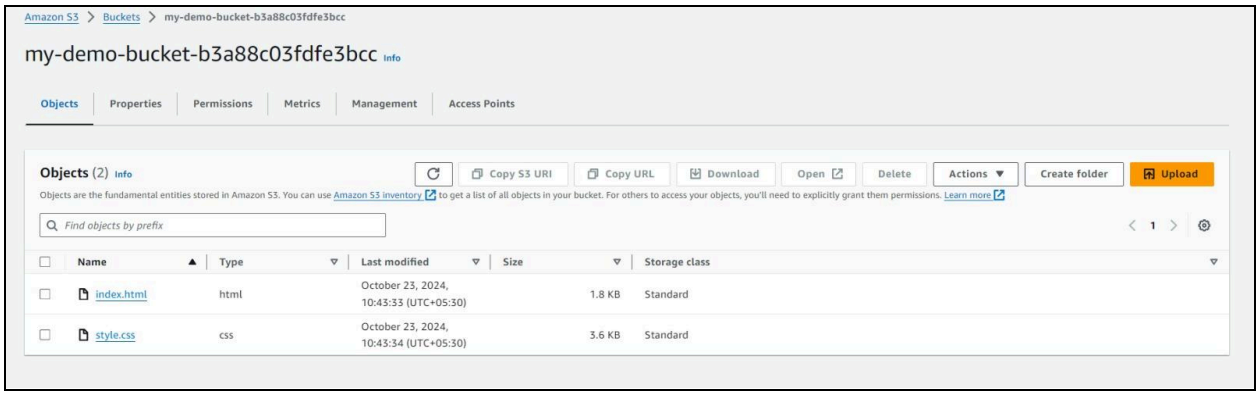
View logs:
Run this command:
cat s3_logs.txt
cat upload_log.txt

```
[ec2-user@ip-172-31-23-146 ~]$ aws s3 cp textfile.txt s3://my-static-website-bucket-unique123/
upload: ./textfile.txt to s3://my-static-website-bucket-unique123/textfile.txt
[ec2-user@ip-172-31-23-146 ~]$ aws s3 cp textfile.txt s3://my-static-website-bucket-unique123/ > upload_log.txt
```

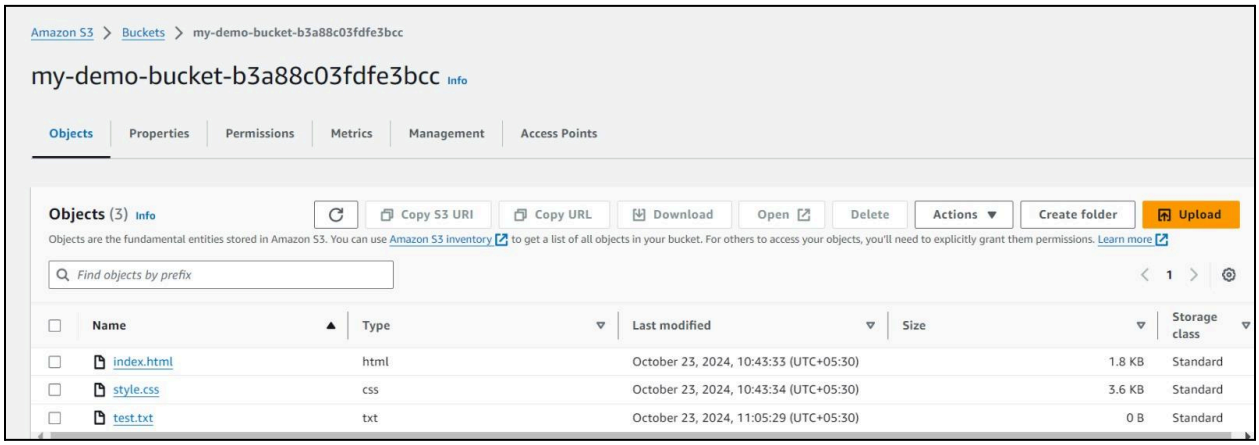
```
[ec2-user@ip-172-31-23-146 ~]$ cat upload_log.txt
upload: ./textfile.txt to s3://my-static-website-bucket-unique123/textfile.txt
[ec2-user@ip-172-31-23-146 ~]$
```

```
Last login: Wed Oct 23 04:57:57 2024 from 18.237.140.163
[ec2-user@ip-172-31-23-146 ~]$ echo "Test log file" > textfile.txt
[ec2-user@ip-172-31-23-146 ~]$ aws s3 cp testfile.txt s3://my-static-website-bucket-unique123/

The user-provided path testfile.txt does not exist.
```

The bucket before the upload using log actions:



The bucket after the upload using log actions:



Conclusion:

In this case study, we successfully utilized Infrastructure as Code (IaC) principles with Terraform to automate the provisioning of AWS resources. Specifically, we created an EC2 instance and an S3 bucket. The S3 bucket was used to host a static website, while the EC2 instance served as the backend server, interacting with the S3 bucket to log actions.

The process involved:

- Writing Terraform code to define and provision AWS resources.
- Assigning an IAM role to the EC2 instance with appropriate policies to allow interaction with AWS Systems Manager and S3.
- Deploying a static website on the S3 bucket and enabling public access via bucket policies.
- Ensuring proper logging and session management through the EC2 instance using AWS Systems Manager.

This case study demonstrates the power and flexibility of Terraform for managing AWS resources efficiently and securely. By automating resource creation and management, we minimized manual configuration, improved scalability, and ensured a repeatable and consistent infrastructure setup.