

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:

```
AWS ● Used $7.6 of $50 03:48 ▶ Start Lab

eee_W_3413365@runweb141253:~$ ^V
bash: $'\026': command not found
eee_W_3413365@runweb141253:~$ export aws_access_key_id="ASIA3IKFWBC42RZHM8DB"
eee_W_3413365@runweb141253:~$ export aws_secret_access_key="02Qia0RmbwoPVH00tIN8e18TQhowgHgVmSSG1BMG"
eee_W_3413365@runweb141253:~$ export aws_session_token="IQoJb3JpZ2l1uXZlJEAgACXVzLXd1c3QtMiJIMEYCIQCC0dbO6yM6yJmIYwfdMI
8a2cpUdMpBoD6+syB0vq1x8gIhAMxoiNK1o/Qd547xQ63wCn2VYIxb+TNVNBXv/qYm6rAKqgCCHEQABoMNzcNzc3MTMxNzA1Igxg5swQ56I6Lp+80Mwq
hQLAMF4ac0SSnKZexf2nSrpn6dVHrJgI1iQdeqoRLMmrqDRwnTnHwmhrHxsSUW/kg5td+xMWPDi4ywpZn5d+o837AAun6WvZZiUaBGC9MMODupn71DFyUb
nqOV9mXHdOVHsbWgpPRPD3L5kP9rCEW1hsK2YvN5ONIXeatDytEtD32KEJHvkB5xUPBvypGDuGTdXdr2zhvowXVU7UuKu1o9mEy6Ltd5eHcr8b86gKRW7x
EFLNjSkteNoZuOG0H+0MBS70zmTDKkDgHs3WszI32O1PYGTzPTaTq/gQNNHTo7XwwLI1A/f+rns/Q11kKRnH1Go9dsJ4/yAa67sT2hbJ6rB/FMF1NVIw1f
nSuAY6nAFVzTVHnSGKQmK916tAaV2tcvHj39H17BDESjho8cEUK197unx+i3gyHUHZ6Xk04rn7QQ4bYnbBJg3VS1/wMdnDQ+pFbZ0pzDELEdPnOacj8MGV
5z5F9Dw9Ch/wd3+e56A0FKZVSASmHx+zVkXDuXPgh+OgoABNERmgIOPhsFzANrHzI1kZzxRC1Kk7rkMIzTJAc1JahPJcPb9b0t8="[]
```

This is the AMI_ID:

below

Search our full catalog including 1000s of application and OS images

Recents Quick Start

Amazon Linux macOS Ubuntu Windows Red Hat SUSE Linux

Amazon Machine Image (AMI)

Microsoft Windows Server 2022 Base
ami-0324a83b82023f0b3 (64-bit (x86))
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Microsoft Windows 2022 Datacenter edition. [English]

Architecture AMI ID Username

64-bit (x86) ami-0324a83b82023f0b3 root

Free tier eligible

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance

Launch instance

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+syB0vqlx8glhAMxoiNK1o/Qd547xQ63wCn2VYIhx+TNVNkBXv/qYm6rAKqgCCHEQ
ABoMNzczNzc3MTMxNzA1lgxg5swQ56l6Lp+8OMwqhQLAMF4ac0SSnKZexf2nSrpn6dVHrJgl
1iQdeqoRLMmrqDRwnTnHwmhrHxsSUW/kG5td+xMWPDi4ywpZn5d+o837AAun6WvZZiUaBG
C9MMODupn71DFyUbnqOV9mXHdOVHSbWgpPRPD3L5kP9rCEWIhsK2YvN5ONlxeatDytEtd
32KEJHvkB5xUPBvypGDuGTdXdr2zhvowXVU7UuKu1o9mEy6Ltd5eHcr8b86gKRW7xEfLNJskt
eNoZuOGOH+0MBS70zmTDKkDgHs3Wszl32OIPYGTzPTaTq/gQNNHTo7XwwLIIA/f+rrs/Q1ikK
RnHlGo9dsJ4/yAa67sT2hbj6rB/FMf1NVIwfnSuAY6nAFVzTVHnSGKQmK916tAaV2tcvHj39Hi7
BDESjho8cEUkl97unx+i3gyHUHZ6XkO4rn7QQ4bYnbBJg3VSI/wMdnDQ+pFbZ0pzDELEdPnO
acj8MGV5z5F9Dw9Ch/wd3+e56AOFKZVSASmHx+zVkXDUXPGh+OgoABNERmgIOPHsFzANr
Hzl1kZzxRCIKk7rkMIZtJAc1JahPJcPb9bOt8="
  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 indicated
+ 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 actions are indicated with the following symbols:
+ 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.

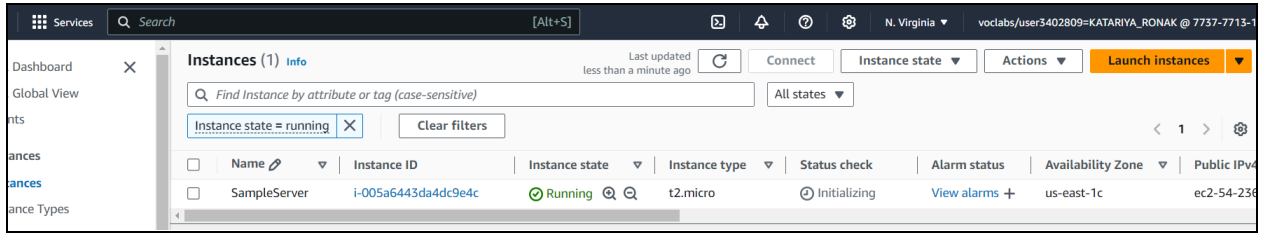
  Enter a value: yes

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:

```

AWS ● Used $7.6 of $50 03:53 ▶ Start Lab

eee_W_3413365@runweb141280:~$ export aws_access_key_id=ASIA3IKFWBC4ZOZYH36Z
eee_W_3413365@runweb141280:~$ export aws_secret_access_key=PCGz8WLykK8K8ECeV7QCAA+AhLpYYF+8KBBGhS6Q
eee_W_3413365@runweb141280:~$ export aws_session_token=IQoJb3JpZ2luX2VjEA4aCXVzLXd1c3QtMiJHMEUCIQD0YP91VmHMONTf1ra0JB
Kj5KRv4ekOvi7IrcNcwKXhWgIU/twSs1G2aePp1ubrKLz7tAV7j3QLLACU3wX6sgUWAqqAII dxAAGgw3NzM3NzcMzE3MDU1dNRKJavH2Cqr/z25yqFA
phb3xwo/+Cv0H8/P4CX00nM+Wn1RSFtm9VXPxeqico3LkWnyHShigie0WUkjctXQjnLHVEgfQZs9j08A39T7iAUU4P9uPuR1ZN7gFe6c/Q0E0x8VdIrMqW
jhkrbwNRFmz1yevbJJfZH+iNwz7RSRGe1Tj4FKQmDJexdgr0ngAxCB8tjVCivF8g5ZqRqVdMp8LW8ujh03gX680r+ARzjBI0JwmBwnp6zrEfDDfK549yL
NIIGzT/Nh8zM+QHxBgBTx0tEPwoLdPrY6Ecs2XLcqsJRrsCdy1bIDy7Y63XR9J9DwtjrdNr9Q5BgvTTmzxSndoASCoYUSBWozBA1HEhz84s455JEzCrjNS
4BjqdAZ+GIn07AAWYdQudvX/e4419ixJOGETp+wBapt2atttT3zxP0pe0i21WvdLLejq5TOKZo61Gp0cGK7bb9d9RLbPznBcNHaiRk1gXxm2stsELW3xnt
EdCb3UNK8G1DzFsEfcwUpewwzMH1xs8GE+UFIHszuv2xnu4/O2N+ta2K1P0+MaoCApugpd0siHTspUFE/4jZipp6VVpzi+I9+w=
eee_W_3413365@runweb141280:~$

```

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="IQoJb3JpZ2luX2VjEA4aCXVzLXdlc3QtMiJHMEUCIQD0YP9IVmHMONLtFlra0JBKj5KR
V4ekOvi7IrcNcwKXHwlgIU/twSs1G2aePplubrKLz7tAV7j3QLLACU3wX6sgUWAqqAIIIdxAAGgw
3NzM3NzcMzE3MDUiDNReKUavH2Cqr/z25yqFaphb3xwo/+Cv0H8/P4CXO0nM+Wn1RSFtm9
VXPxeqico3LkWNyHShigie0WUkjcTXQjnLHVEgfQZs9j08A39T7iAUU4P9uPuR1ZN7gFe6c/Q0
EOx8VdlrMqWjhkrbwNRFmz1yevbJfZH+iNwz7RSRGe1Tj4FKQmDJexdgr0ngAxCXB8tjVCivF
8g5ZqRqVdMp8LW8ujhO3gX68Or+ARzjBI0JwmBWnp6zrEfDDFk549yLNlIgzT/Nh8zM+QHxBg
BTxOtEPwoLdPrY6Ecs2XLcqsJRrsCdy1bIDy7Y63XR9J9DWtjrdNr9Q5BgvTTmzxSndoASCoY
USBWozBAIHhEhz84s455JEzCrjNS4BjqdAZ+Gln07AAWyDqUdvX/e4419ixJOGEtp+wBapt2atttT
3zxPOpeOi2IWvdLLejq5TOKZo61GpOcGK7bb9d9RLbPznBcNHalrK1gXxm2stsELW3xntEdCb
3UNK8GIDzFsEfCWUpewwzMh1xs8GE+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 actions
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_3-08w]
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 are indicated with the
+ 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.

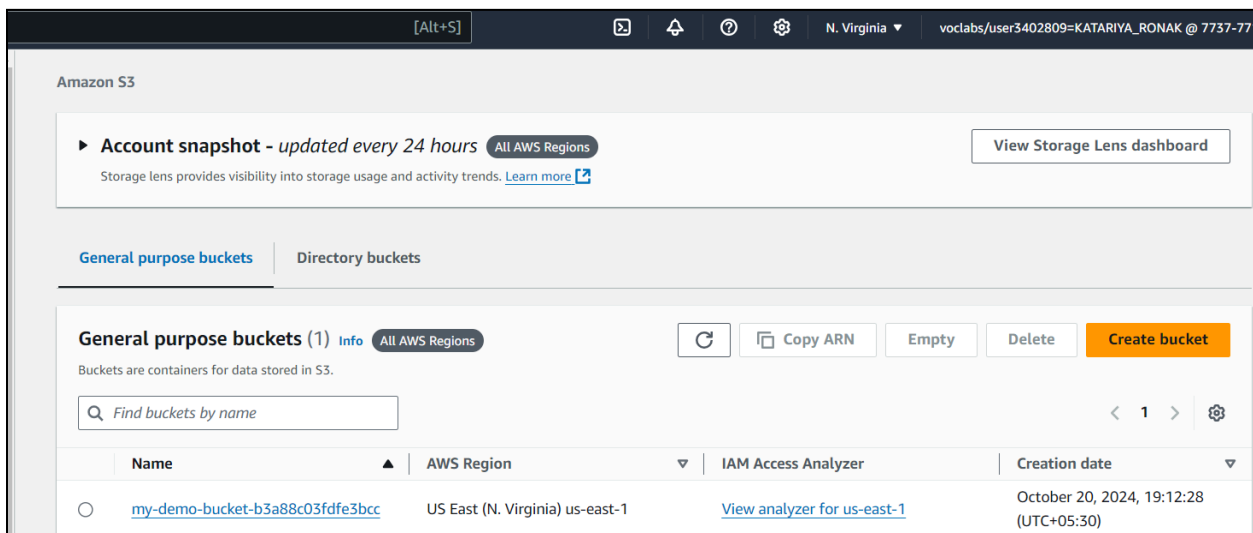
Enter a value: yes

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="IQoJb3JpZ2luX2VjEA4aCXVzLXdlc3QtMiJHMEUCIQD0YP9IVmHMONLtFlra0JBKj5KR  
V4ekOvi7IrcNcwKXHwlgIU/twSs1G2aePplubrKLz7tAV7j3QLLACU3wX6sgUWAqqAII dxAAGgw  
3NzM3NzcxMzE3MDUiDNReKUavH2Cqr/z25yqFAphb3xwo/+Cv0H8/P4CXO0nM+Wn1RSFtm9  
VXPxeqico3LkWNYHShigie0WUkjcTXQjnLHVEgfQZs9j08A39T7iAUU4P9uPuR1ZN7gFe6c/Q0  
EOx8VdlrMqWjhkrbwNRFmz1yevbJJfZH+iNwz7RSRGe1Tj4FKQmDJexdgr0ngAxCXB8tjVCivF  
8g5ZqRqVdMp8LW8ujhO3gX68Or+ARzjBI0JwmBWnp6zrEfDDfk549yLNIlgzT/Nh8zM+QHxBg  
BTxOtEPwoLdPrY6Ecs2XLcqsJRrsCdy1bIDy7Y63XR9J9DWtjrdNr9Q5BgvTTmzxSndoASCoY  
USBWozBAIHEhz84s455JEzCrjNS4BjqdAZ+Gln07AAWyDqUdvX/e4419ixJOGEtp+wBapt2atttT  
3zxPOpeOi2lWvdLLejq5TOKZo61GpOcGK7bb9d9RLbPznBcNHalrK1gXxm2stsELW3xntEdCb  
3UNK8GIDzFsEfcwUpewwzMh1xs8GE+UFIHszuv2xnu4/O2N+ta2K1P0+MaoCApugpd0siHTsp  
UfE/4jZipp6VVpzi+I9+w="
```

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

```
}
```

Step 3: Execute Terraform init command.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE COMMENTS

```
PS C:\Users\272241\FinalAWS\aws-s3> cd ../
PS C:\Users\272241\FinalAWS> cd .\static-website-hosting\
PS C:\Users\272241\FinalAWS\static-website-hosting> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.64.0"...
- Finding hashicorp/random versions matching "3.6.2"...
- 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.
PS C:\Users\272241\FinalAWS\static-website-hosting> Terraform plan
```

Step 4: Terraform plan and terraform apply:

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)
```



```
}

# 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)
}
```

```
+ 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" {
```

```
+ 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. Resources to be created:

- + 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=i5gm50_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 :

The screenshot shows the Amazon S3 console interface. The breadcrumb navigation at the top reads 'Amazon S3 > Buckets > my-mywebapp-bucket-8b9826e4eff708dd'. The bucket name 'my-mywebapp-bucket-8b9826e4eff708dd' is displayed with an 'Info' link. Below this is a tabbed interface with 'Properties' selected. The 'Bucket overview' section contains a table with the following data:

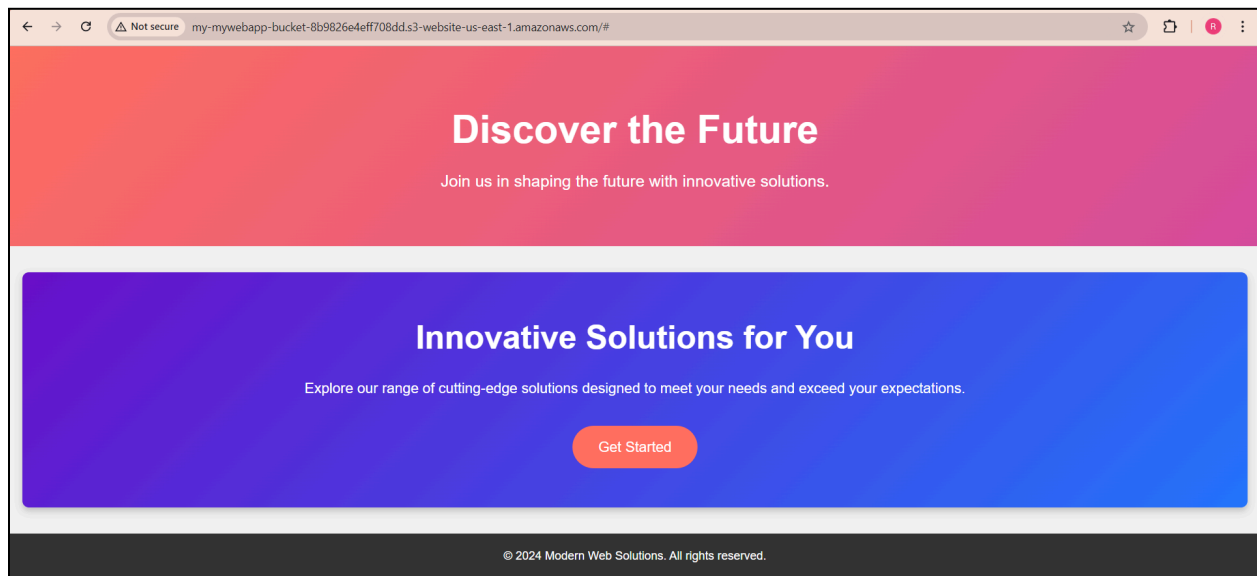
AWS Region US East (N. Virginia) us-east-1	Amazon Resource Name (ARN) arn:aws:s3:::my-mywebapp-bucket-8b9826e4eff708dd	Creation date October 20, 2024, 19:31:36 (UTC+05:30)
---	--	---

The 'Bucket Versioning' section shows 'Bucket Versioning' as 'Disabled' and 'Multi-factor authentication (MFA) delete' as 'Disabled'. Both sections have an 'Edit' button. Descriptive text and 'Learn more' links are provided for both settings.

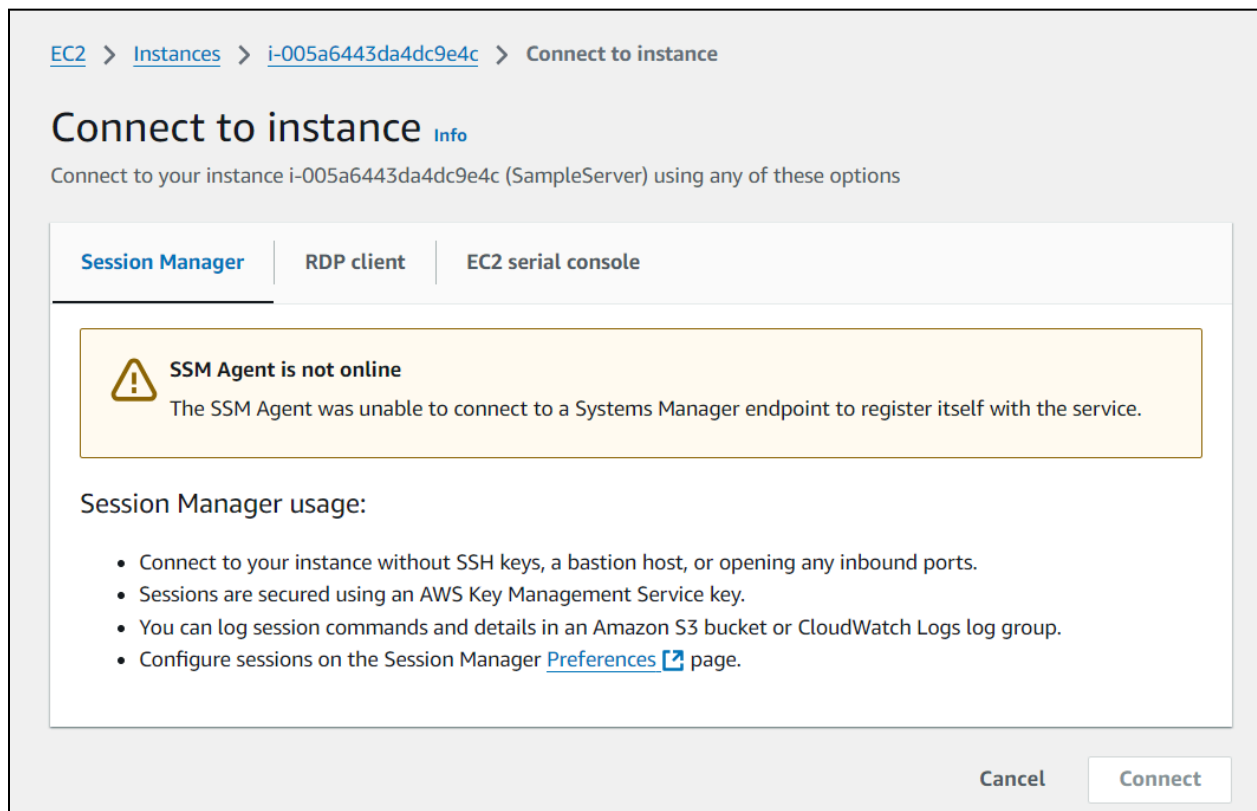
Scroll down :

This screenshot shows the same console page as above, but scrolled down to reveal additional configuration sections. The 'Object Lock' section shows 'Object Lock' as 'Disabled'. The 'Requester pays' section shows 'Requester pays' as 'Disabled'. The 'Static website hosting' section shows 'S3 static website hosting' as 'Enabled' and 'Hosting type' as 'Bucket hosting'. The 'Bucket website endpoint' is displayed as 'http://my-mywebapp-bucket-8b9826e4eff708dd.s3-website-us-east-1.amazonaws.com'. Each section includes an 'Edit' button and a 'Learn more' link.

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:

[Alt+S]

Your preferences have been successfully updated.

[AWS Systems Manager](#) > [Session Manager](#) > Start a session

Step 1
Specify target

Step 2 - optional
Specify session document

Step 3
Review and launch

Specify target

Select an instance to connect to using Session Manager.

Reason

Reason for session – optional
The reason for connecting to the instance. This value is included in the details of the event created by AWS CloudTrail when you start the session.

To perform Practical Exam

This value can have up to 256 characters.

Target instances

Filter instances

Instance na...	Instance ID	Agent version	Instance state	Availability...	Platform
There are no instances which are associated with the required IAM role.					

[Learn how to create and attach the required IAM role to your instances.](#)

Create a new Role:

Services Search [Alt+S]

[IAM](#) > [Roles](#) > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Select trusted entity

Trusted entity type

☒ **AWS service**
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

☐ **AWS account**
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

☐ **Web identity**
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

☐ **SAML 2.0 federation**
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

☐ **Custom trust policy**
Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case
EC2

Select EC2 role for AWS Systems Manager:

Service or use case

EC2

Choose a use case for the specified service.

Use case

- ☐ EC2
Allows EC2 instances to call AWS services on your behalf.
- ☒ **EC2 Role for AWS Systems Manager**
Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.
- ☐ **EC2 Spot Fleet Role**
Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.
- ☐ **EC2 - Spot Fleet Auto Scaling**
Allows Auto Scaling to access and update EC2 spot fleets on your behalf.
- ☐ **EC2 - Spot Fleet Tagging**
Allows EC2 to launch spot instances and attach tags to the launched instances on your behalf.
- ☐ **EC2 - Spot Instances**
Allows EC2 Spot Instances to launch and manage spot instances on your behalf.
- ☐ **EC2 - Spot Fleet**
Allows EC2 Spot Fleet to launch and manage spot fleet instances on your behalf.
- ☐ **EC2 - Scheduled Instances**
Allows EC2 Scheduled Instances to manage instances on your behalf.

Cancel Next

Now add permission :

Services Search [Alt+S] Global voclabs/user3402809=KATARIYA_RONAK @ 7737-7713-1

IAM > Roles > Create role

Step 1
[Select trusted entity](#)

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions Info

Permissions policies (1) Info
The type of role that you selected requires the following policy.

Policy name	Type
AmazonSSMManagedInstanceCore	AWS managed

Set permissions boundary - optional

Set a permissions boundary to control the maximum permissions this role can have. This is not a common setting but can be used to delegate permission management to others. [Learn more about permission boundaries](#)

- ☒ Create role without a permissions boundary
- ☐ Use a permissions boundary to control the maximum role permissions

Cancel Previous Next

Now started with the log actions:

Connect the instance and then start performing the log actions:

[illegible]

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
```

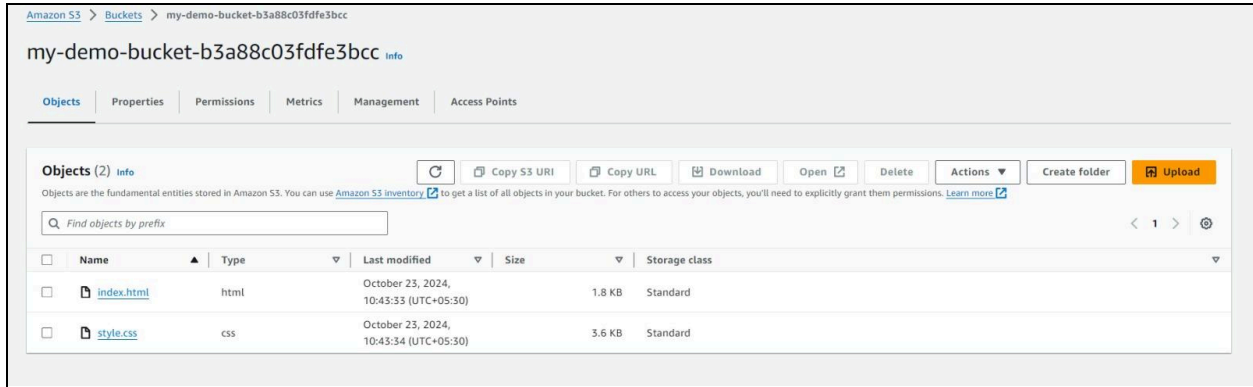
```
[ec2-user@ip-172-31-23-146 ~]$ aws s3 cp textfile.txt s3://my-static-website-bucket-unique123/
upload: ./textfile.txt to s3://my-static-website-bucket-unique123/textfile.txt
[ec2-user@ip-172-31-23-146 ~]$ aws s3 cp textfile.txt s3://my-static-website-bucket-unique123/ > upload_log.txt
```

```
[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 ~]$
```

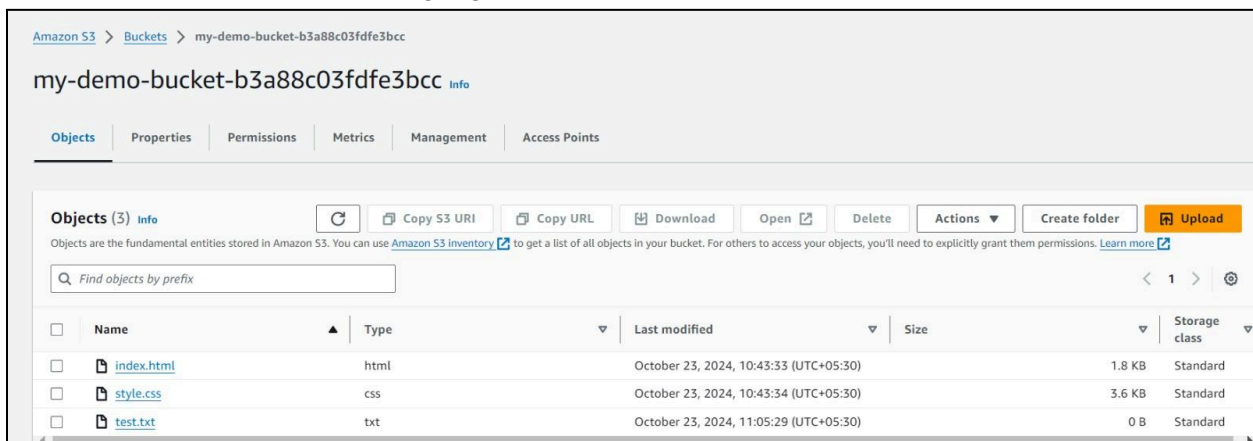
```
Last login: Wed Oct 23 04:57:57 2024 from 18.237.140.163
[ec2-user@ip-172-31-23-146 ~]$ echo "Test log file" > testfile.txt
[ec2-user@ip-172-31-23-146 ~]$ aws s3 cp testfile.txt s3://my-static-website-bucket-unique123/

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.