

Name: Hitesh Tanwani
Roll No: 62
Div: D15B

Advanced DevOps-6

Aim

To build, change, and destroy infrastructure on AWS, GCP, Microsoft Azure, or DigitalOcean using Terraform, a tool for automating the management of cloud resources.

Theory

Terraform is an open-source Infrastructure as Code (IaC) tool developed by HashiCorp. It allows developers and system administrators to define, provision, and manage cloud infrastructure using a declarative configuration language called HashiCorp Configuration Language (HCL). Terraform supports multiple cloud providers, including AWS, GCP, Azure, and DigitalOcean, making it a versatile tool for managing cloud environments.

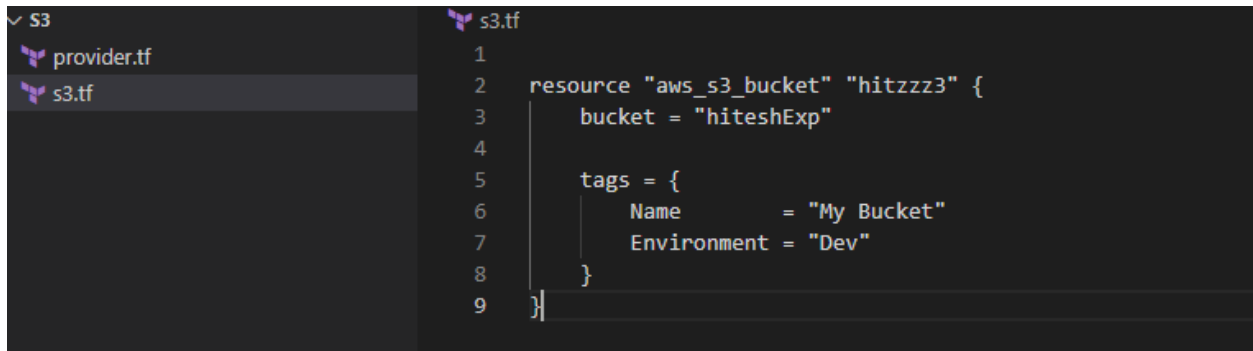
Key Concepts of Terraform

1. **Infrastructure as Code (IaC):**
Terraform enables defining cloud infrastructure through code, allowing for version control, reproducibility, and collaboration.
 2. **State Management:**
Terraform maintains the state of the infrastructure, allowing it to track and manage changes over time.
 3. **Resource Provisioning:**
It can provision various resources like virtual machines, storage, networking, and more across multiple cloud platforms.
 4. **Modular Approach:**
Infrastructure configurations can be broken into reusable modules, making them easy to manage and maintain.
 5. **Lifecycle Management:**
Terraform provides commands to create (apply), update (plan), and destroy (destroy) resources, making it easy to manage the full lifecycle of infrastructure.
-

Benefits of Using Terraform

- **Multi-Cloud Compatibility:**
Supports various cloud providers, offering flexibility and reducing vendor lock-in.
 - **Automation:**
Automates infrastructure provisioning, reducing manual efforts and minimizing errors.
 - **Version Control:**
Changes in infrastructure can be tracked, reviewed, and rolled back using version control systems.
 - **Scalability:**
Easily scales infrastructure up or down according to needs, optimizing resource usage and cost.
-

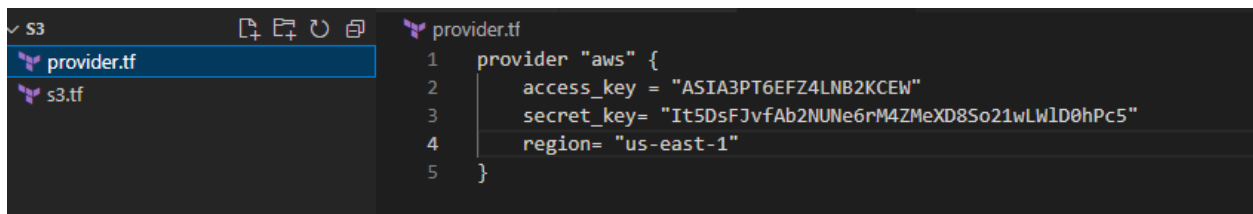
Step 1: Write a Terraform Script in Atom for Creating S3 Bucket on Amazon AWS



The screenshot shows the Atom editor interface. On the left, a file explorer pane shows a directory named 'S3' containing two files: 'provider.tf' and 's3.tf'. The 's3.tf' file is selected and highlighted. The main editor area displays the content of 's3.tf', which is a Terraform script for creating an S3 bucket. The script is as follows:

```
1
2 resource "aws_s3_bucket" "hitzzz3" {
3     bucket = "hiteshExp"
4
5     tags = {
6         Name      = "My Bucket"
7         Environment = "Dev"
8     }
9 }
```

Create a new `provider.tf` file and write the following contents into it. Save both the files in the same directory `Terraform_Scripts/S3`.



The screenshot shows the Atom editor interface. On the left, a file explorer pane shows a directory named 'S3' containing two files: 'provider.tf' and 's3.tf'. The 'provider.tf' file is selected and highlighted. The main editor area displays the content of 'provider.tf', which is a Terraform script for configuring the AWS provider. The script is as follows:

```
1 provider "aws" {
2     access_key = "ASIA3PT6EFZ4LNB2KCEW"
3     secret_key = "It5DsFJvfAb2NUNE6rM4ZMeXD8So21wLWlD0hPc5"
4     region = "us-east-1"
5 }
```

Step 2: Execute Terraform Init Command

```
C:\>cd Terraform

C:\Terraform>cd ..

C:\>cd "Terraform Scripts"

C:\Terraform Scripts>cd S3

C:\Terraform Scripts\S3>dir
Volume in drive C has no label.
Volume Serial Number is B25F-43A1

Directory of C:\Terraform Scripts\S3

08-08-2024  14:53    <DIR>          .
08-08-2024  14:39    <DIR>          ..
08-08-2024  14:53                0 provider.tf
08-08-2024  14:40                0 s3.tf
                2 File(s)                0 bytes
                2 Dir(s)  190,582,370,304 bytes free

C:\Terraform Scripts\S3>|
```

Execute the command to initialize the resources.

```
PS C:\Terraform Scripts\S3> terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.61.0

Terraform has been successfully initialized!

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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Terraform Scripts\S3> echo hitesh
```

Step 3: Execute Terraform Plan

Execute the command to see the available resources.

```
PS C:\Terraform Scripts\S3> terraform plan

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_s3_bucket.hitzzz3 will be created
+ resource "aws_s3_bucket" "hitzzz3" {
+   acceleration_status      = (known after apply)
+   acl                      = (known after apply)
+   arn                      = (known after apply)
+   bucket                   = "hiteshExp"
+   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                     = {
+     "Environment" = "Dev"
+     "Name"        = "My Bucket"
+   }
+   tags_all              = {
+     "Environment" = "Dev"
+     "Name"        = "My Bucket"
+   }
+   website_domain         = (known after apply)
+   website_endpoint       = (known after apply)
+   cors_rule (known after apply)
+     + "Name" = "My Bucket"
+   }
+   tags_all              = {
+     "Environment" = "Dev"
+     "Name"        = "My Bucket"
+   }
+   website_domain         = (known after apply)
+   website_endpoint       = (known after apply)

+ cors_rule (known after apply)

+ grant (known after apply)

+ lifecycle_rule (known after apply)

+ logging (known after apply)

+ object_lock_configuration (known after apply)

+ replication_configuration (known after apply)

+ server_side_encryption_configuration (known after apply)

+ versioning (known after apply)

+ website (known after apply)
}

Plan: 1 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 if you run
"terraform apply" now.
PS C:\Terraform Scripts\S3> |
```

Activate Windows
Go to Settings to activate Windows.

Step 4: Execute Terraform Apply

Execute the command to apply the configuration, which will automatically create an S3 bucket based on our configuration.

```
PS C:\terraform_scripts\s3> terraform apply

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

Terraform will perform the following actions:

# aws_s3_bucket.sujal will be created
+ resource "aws_s3_bucket" "sujal" {
+   acceleration_status      = (known after apply)
+   acl                      = (known after apply)
+   arn                      = (known after apply)
+   bucket                   = "sujal-63"
+   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)
+   replication_configuration = (known after apply)
+   server_side_encryption_configuration = (known after apply)
+   versioning                = (known after apply)
+   website                   = (known after apply)
}

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

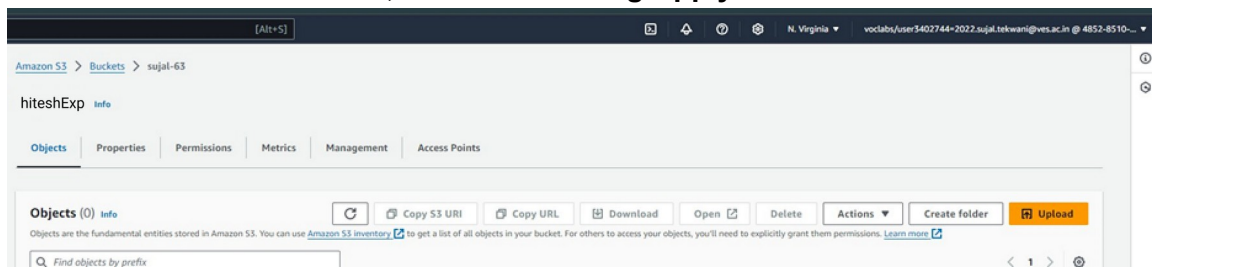
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_s3_bucket.sujal: Creating...
aws_s3_bucket.sujal: Creation complete after 7s [id=sujal-63]

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

AWS S3 Bucket Dashboard, Before Executing Apply Command:



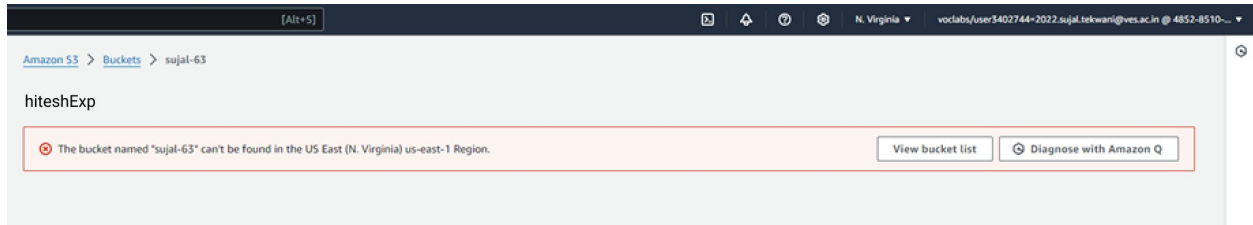
Step 5: Execute Terraform Destroy

Execute the command to delete the configuration, which will automatically delete an EC2 instance.

```
PS C:\terraform_scripts\s3> terraform destroy
```

```
Destroy complete! Resources: 1 destroyed.
```

AWS S3 Bucket Dashboard, After Executing Apply Command:



Conclusion

Using Terraform to build, change, and destroy infrastructure across AWS, GCP, Microsoft Azure, and DigitalOcean showcases the power and flexibility of Infrastructure as Code (IaC). Terraform's ability to automate infrastructure provisioning, manage state, and support multiple cloud platforms streamlines operations, reduces human error, and enhances scalability.

By employing Terraform, organizations can efficiently manage their cloud environments, enforce best practices, and achieve consistent infrastructure management across diverse cloud providers, ultimately accelerating their cloud adoption and operational agility.