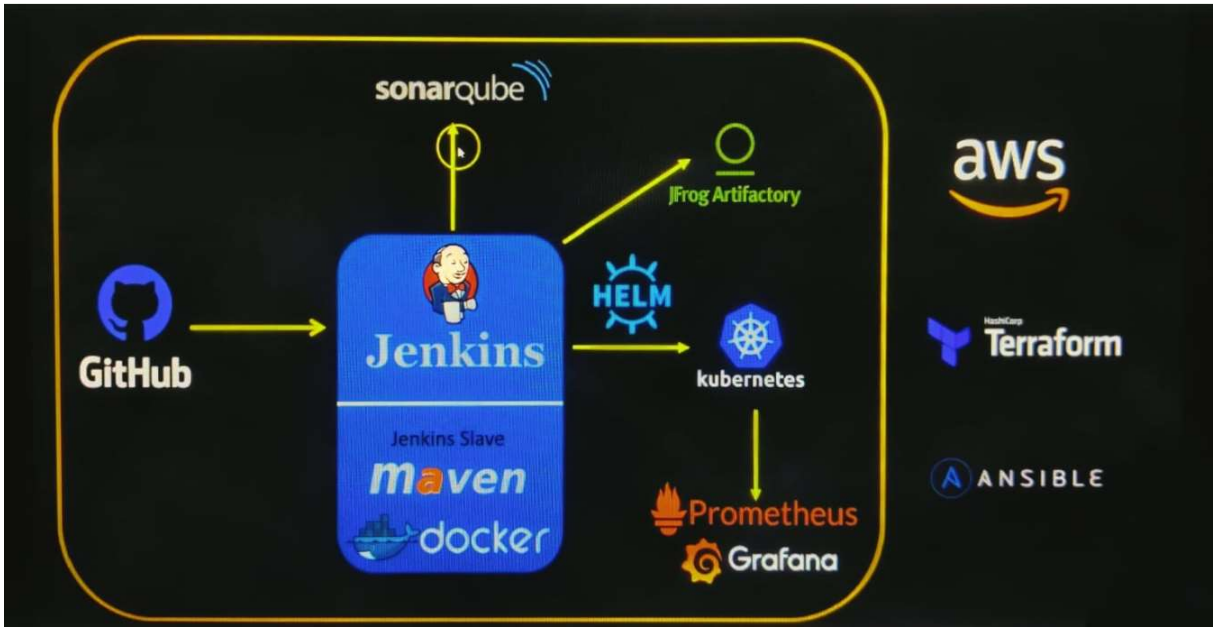
CICD Project Workshop



In this course we are going to create a CI CD pipeline by using various tools.

▼ **Tech stacks used in this project**

GitHub

Jenkins

Terraform

Ansible

Maven

SonarQube

Jfrog

Docker

Kubernetes

Helm Charts

Prometheus

Grafana

▼ **Steps to perform during this CICD pipeline project**

- Set up Terraform
- Provision Jenkins master, build Node and Ansible using Terraform.
- Set up Ansible server.
- Configure Jenkins master and build node using Ansible.
- Create a Jenkins pipeline job
- Create a Jenkins file from scratch.
- Create Multi-branch pipeline

- Enable webhook on GitHub.
- Configuring Sonar Cube and Sonar Scanner.
- Execute Sonar Cube analysis.
- Define rules and gates on Sonar Cube.
- Sonar callback rules.
- Jfrog Artifactory Setup.
- Create a Docker file
- Store Docker Images on Jfrog Artifactory.
- Provisioned Kubernetes cluster using Terraform.
- Create Kubernetes Objects.
- Deploying the Kubernetes objects using Helm.
- Set up Prometheus and Grafana using Helm Charts.
- Monitor Kubernetes Cluster using Prometheus.

▼ **Pre-requisites**

- Install below tools on local system
- Visual Studio
- Git
- Terraform
- AWS CLI
- Mobaxterm

Terraform

````
## Prepare Terraform Environment on Windows
As part of this, we should setup
1. Terraform
2. VS Code
3. AWSCLI

### Install Terraform
1. Download terraform the latest version from [here](https://developer.hashicorp.com/terraform/downloads)
2. Setup environment variable
click on start --> search "edit the environment variables" and click on it
Under the advanced tab, chose "Environment variables" --> under the system variables select path variable
and add terraform location in the path variable. system variables --> select path
add new --> terraform_Path
in my system, this path location is C:\Program Files\terraform_1.3.7

1. Run the below command to validate terraform version
   ```sh
   terraform -version
   ```
   the output should be something like below
   ```sh
   Terraform v1.3.7
   on windows_386
   ```

 ### Install Visual Studio code

  Download vs code latest version from [here](https://code.visualstudio.com/download) and install it.

### AWSCLI installation


Download AWSCLI latest version from [here](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html) and insta
````

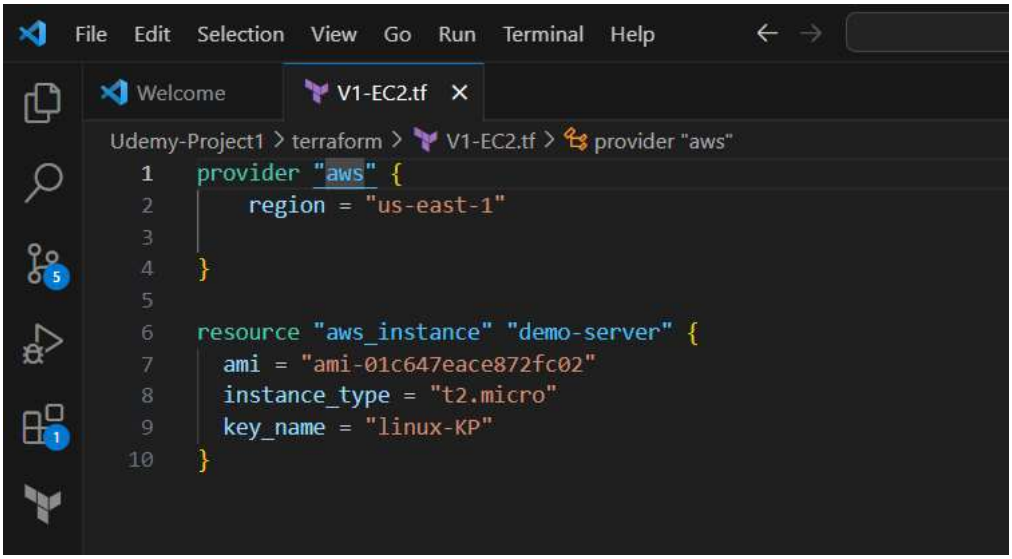
```
or you can run the below command in powershell or the command prompt
```

Terraform-code

1. Create IAM user
2. Login to aws cli

```
namratak@NAMRATAKFTVTCS3 MINGW64 ~/Linux/Udemy-Project1/terraform
$ aws configure
AWS Access Key ID [****************FU46]: AKIAQXVMWBLQZR2E3OHG
AWS Secret Access Key [****************iq1x]: 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
Default region name [us-east-1]: us-east-1
Default output format [None]:
```

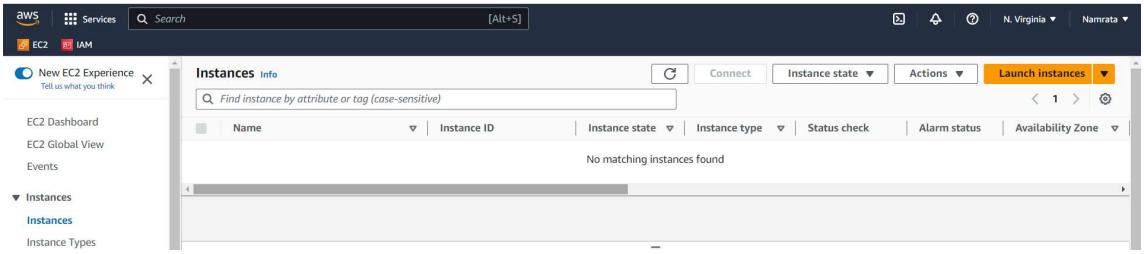
3. Write the First Terraform code



4. **Run terraform command > terrafrom init > terraform validate > terraform plan**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

namratak@NAMRATAKFTVTCS3 MINGW64 ~/Linux/Udemy-Project1/terraform
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.16.0

Terraform has been successfully initialized!

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

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

namratak@NAMRATAKFTVTCS3 MINGW64 ~/Linux/Udemy-Project1/terraform
$ terraform validate
Success! The configuration is valid.


namratak@NAMRATAKFTVTCS3 MINGW64 ~/Linux/Udemy-Project1/terraform
$ 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_instance.demo-server will be created
  + resource "aws_instance" "demo-server" {
      + ami                                  = "ami-01c647eace872fc02"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
```

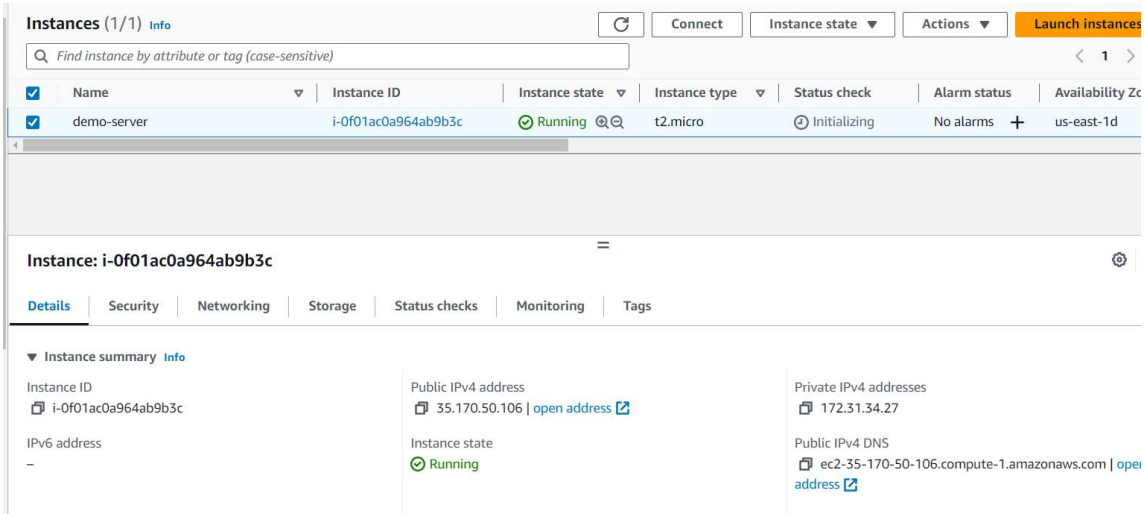
5. **Before running the 'terraform apply' command check AWS console**



6. **Run terraform 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.demo-server: Creating...
aws_instance.demo-server: Still creating... [10s elapsed]
aws_instance.demo-server: Still creating... [20s elapsed]
aws_instance.demo-server: Creation complete after 26s [id=i-0f01ac0a964ab9b3c]

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

7. **EC2 instance created**



8. **To destroy created infrastructure run 'terraform destroy'**

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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.demo-server: Destroying... [id=i-0f01ac0a964ab9b3c]
aws_instance.demo-server: Still destroying... [id=i-0f01ac0a964ab9b3c, 10s elapsed]
aws_instance.demo-server: Still destroying... [id=i-0f01ac0a964ab9b3c, 20s elapsed]
aws_instance.demo-server: Still destroying... [id=i-0f01ac0a964ab9b3c, 30s elapsed]
aws_instance.demo-server: Still destroying... [id=i-0f01ac0a964ab9b3c, 40s elapsed]
aws_instance.demo-server: Destruction complete after 42s

Destroy complete! Resources: 1 destroyed.
```

==========

Terraform with Ansible

This document discusses using Terraform and Ansible to provision infrastructure and configure Jenkins master and slave servers.
The document includes Terraform code for setting up EC2 instances, a VPC, and other resources. It also mentions the process of converting one instance to an AWS instance and using Ansible playbooks to configure the Jenkins servers.
Ansible server is going to manage two different systems and through Ansible playbooks we are going to convert one server as a Jenkins master and another one as a Jenkins slave.
Screenshots of Terraform commands and the created instances are provided.

I have written a Terraform manifest file to create three EC2 instances, by using 'for each block'.

I need to convert one of these instances as a AWS instance. Then this

Features

- Setup 3 EC2 instances through Terraform
- Provision Jenkins-master, Jenkins-slave and Ansible
- Setup Ansible Server
- Configure Jenkins master using Ansible

We need to run the same script multiple times to create multiple instances. Rather than this, I am going to use one more parameter called 'for each'.

Write TF script to provision infrastructure V2-EC2-with-vpc-for-each.

▼ **Terraform code-with-VPC-for-each**

```
provider "aws" {
region = "us-east-1"

}

resource "aws_instance" "demo-server" {
ami = "ami-053b0d53c279acc90"
instance_type = "t2.micro"
key_name = "linux-KP"
//security_groups = ["demo-sg"]
vpc_security_group_ids = [aws_security_group.demo-sg.id]
subnet_id = aws_subnet.Nam-public-subnet-01.id

for_each = toset(["jenkins-master", "jenikns-slave", "ansible"])
tags = {
Name = "${each.key}"
}
}

resource "aws_security_group" "demo-sg" {
name        = "demo-sg"
description = "SSH Access"
vpc_id = aws_vpc.Nam-vpc.id

ingress {
description      = "Shh access"
from_port        = 22
to_port          = 22
protocol         = "tcp"
cidr_blocks      = ["0.0.0.0/0"]
}

egress {
from_port        = 0
to_port          = 0
protocol         = "-1"
cidr_blocks      = ["0.0.0.0/0"]
ipv6_cidr_blocks = ["::/0"]
}

tags = {
Name = "ssh-port"

}
}

resource "aws_vpc" "Nam-vpc" {
cidr_block = "10.1.0.0/16"
tags = {
Name = "Nam-vpc"
}

}

resource "aws_subnet" "Nam-public-subnet-01" {
vpc_id = aws_vpc.Nam-vpc.id
cidr_block = "10.1.1.0/24"
map_public_ip_on_launch = "true"
availability_zone = "us-east-1a"
tags = {
Name = "Nam-public-subent-01"
}
}
```

```
resource "aws_subnet" "Nam-public-subnet-02" {
vpc_id = aws_vpc.Nam-vpc.id
cidr_block = "10.1.2.0/24"
map_public_ip_on_launch = "true"
availability_zone = "us-east-1b"
tags = {
Name = "Nam-public-subent-02"
}
}

resource "aws_internet_gateway" "Nam-igw" {
vpc_id = aws_vpc.Nam-vpc.id
tags = {
Name = "Nam-igw"
}
}

resource "aws_route_table" "Nam-public-rt" {
vpc_id = aws_vpc.Nam-vpc.id
route {
cidr_block = "0.0.0.0/0"
gateway_id = aws_internet_gateway.Nam-igw.id
}
}

resource "aws_route_table_association" "Nam-rta-public-subnet-01" {
subnet_id = aws_subnet.Nam-public-subnet-01.id
route_table_id = aws_route_table.Nam-public-rt.id

}

resource "aws_route_table_association" "Nam-rta-public-subnet-02" {
subnet_id = aws_subnet.Nam-public-subnet-02.id
route_table_id = aws_route_table.Nam-public-rt.id

}
```

`terraform init`

```
namratak@NAMRATAKFTVTCS3 MINGW64 ~/Linux/Workshop-devops2/Terraform (main)
$ ls
V4-EC2-With_VPC_for_each.tf  terraform.tfstate  terraform.tfstate.backup

namratak@NAMRATAKFTVTCS3 MINGW64 ~/Linux/Workshop-devops2/Terraform (main)
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.17.0

Terraform has been successfully initialized!

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

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

`terraform validate`