Course-end Project – 1

By

SimpliLearn

Automating Infrastructure using Terraform.

Name	**:**	**Darla Raviteja**
Contact	**:**	**7075411999**
Email	**:**	**raviteja99darla@gmail.com**
GitHub	**:**	**https://github.com/RAVITEJADARLA**

Automating Infrastructure using Terraform.

Problem Statement:

Use Terraform to provision infrastructure

Terraform is a tool that allows you to provision various infrastructure components. Ansible is a platform for managing configurations it means you'll use Terraform to build a virtual machine, for example, and then use Ansible to install the necessary applications on that machine.

Considering the Organizational requirement you are asked to automate the infrastructure using Terraform first and install other required automation tools in it.

Tools required:

- Terraform
- AWS account with security credentials
- Key pair

Expected Deliverables:

- Launch an EC2 instance using Terraform
- Connect to the instance
- Install Jenkins, Java, and Python in the instance

For Source code & Screen shots Please Scroll Down Terraform Script

Terraform Provider:

Provider in Terraform is a Plugin that enables interaction with an API. The Providers are specified in the Terraform configuration code. They tell Terraform which services it needs to interact with.

```
# Providing AWS Provider
provider "aws" {
    region                   = "us-east-1"
}
```

Terraform VPC:

The entire network architecture of any cloud-based on a Virtual Private Cloud (VPC). AWS VPCs offer the required network segregation and enable security by efficient managing aspects like subnets, routing

```
# Creating AWS VPC
resource "aws_vpc" "my_vpc" {
    cidr_block              = "10.10.0.0/16"
    tags                    = {Name = "my_vpc"}
}
```

Terraform Subnet:

A Subnet is a range of IP addresses in VPC. Launch AWS resources, such as Amazon EC2 instance, into our subnets. We can connect a subnet to the internet

```
# Creating AWS Subnet
resource "aws_subnet" "my_subnet" {
    vpc_id                  = aws_vpc.my_vpc.id
    cidr_block              = "10.10.0.0/24"
    availability_zone       = "us-east-1a"
    map_public_ip_on_launch = true
    tags                    = {Name = "my_subnet"}
}
```

Terraform Internet Gateway:

An Internet Gateway (IGW) is a horizontally scaled, redundant, and highly available Amazon VPC component that allows communication between instance in our Amazon VPC & the internet

```
# Creating AWS Internet Gateway
resource "aws_internet_gateway" "my_internet_getaway" {
    vpc_id                  = aws_vpc.my_vpc.id
    tags                    = {Name = "my_internet_getaway"}
}
```

Terraform Route Table:

A Route Table contains a set of rules, called routes, that determine where network traffic from our subnet or gateway is directed.

```
# Creating AWS Route Table
resource "aws_route_table" "my_route_table" {
    vpc_id                  = aws_vpc.my_vpc.id
    route  {
        cidr_block          = "0.0.0.0/0"
        gateway_id          = aws_internet_gateway.my_internet_getaway.id
    }
    tags                    = {Name = "my_rout_table"}
}
```

Terraform Route Table Association:

Route Table Associates a subnet with a route table. The subnet and route table must be in the same VPC. This association causes traffic originating from the subnet to be routed according to the routes in the route table. A route table can be associated with multiple subnets.

```
# Creating AWS Subnet Association Connecting Subnet with Route Table
resource "aws_route_table_association" "my_route_table_association"{
    subnet_id               = aws_subnet.my_subnet.id
    route_table_id          = aws_route_table.my_route_table.id
}
```

Terraform Security Group:

AWS Provides security groups as one of the tools for securing our instance, and need to configure them to meet our security needs.

```
# Creating AWS Security Group
resource "aws_security_group" "my_security_group" {
    vpc_id                  = aws_vpc.my_vpc.id
    tags                    = {Name = "my_security_group"}
    ingress {
        description         = "SSH"
        from_port           = 22
        to_port             = 22
        protocol            = "tcp"
        cidr_blocks         = ["0.0.0.0/0"]
    }
    ingress {
        description         = "http"
        from_port           = 8080
        to_port             = 8080
        protocol            = "tcp"
        cidr_blocks         = ["0.0.0.0/0"]
    }
    ingress {
        description         = "http"
        from_port           = 80
        to_port             = 80
        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"]
    }
}
```

Terraform Outputs:

```
output "get_instance_name" {
    value = aws_instance.instance.tags.Name
}
output "get_public_ip" {
    value = aws_instance.instance.public_ip
}
```

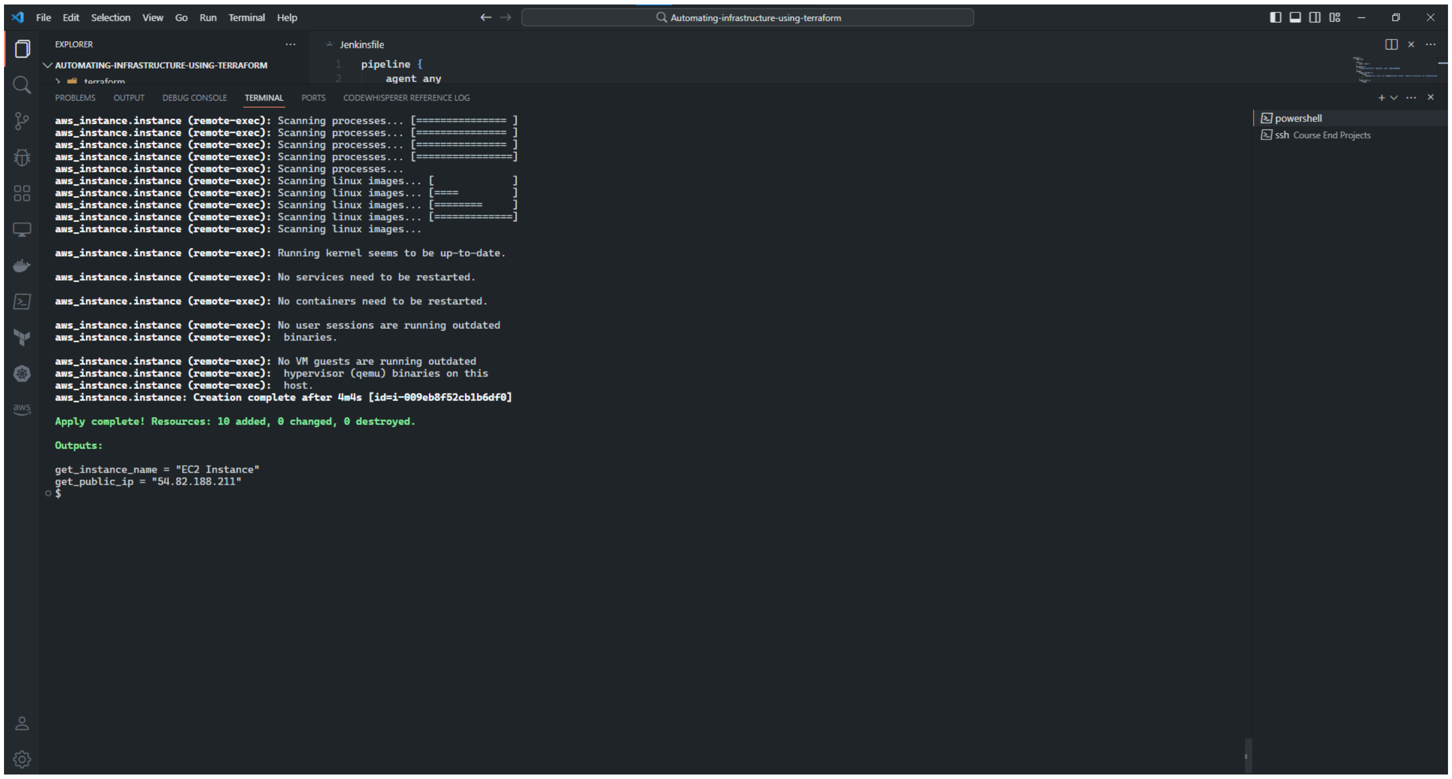
Terraform Creating Key-pair:

```
resource "aws_key_pair" "keypair" {
    key_name        = "project_access_key"
    public_key      = tls_private_key.rsa.public_key_openssh
}
resource "local_file" "instance_key_file" {
    content         = tls_private_key.rsa.private_key_pem
    filename        = "project_access_key.pem"
    file_permission = "0400"
}
resource "tls_private_key" "rsa" {
    algorithm       = "RSA"
    rsa_bits        = 4096
}
```

Terraform AWS-Instance:

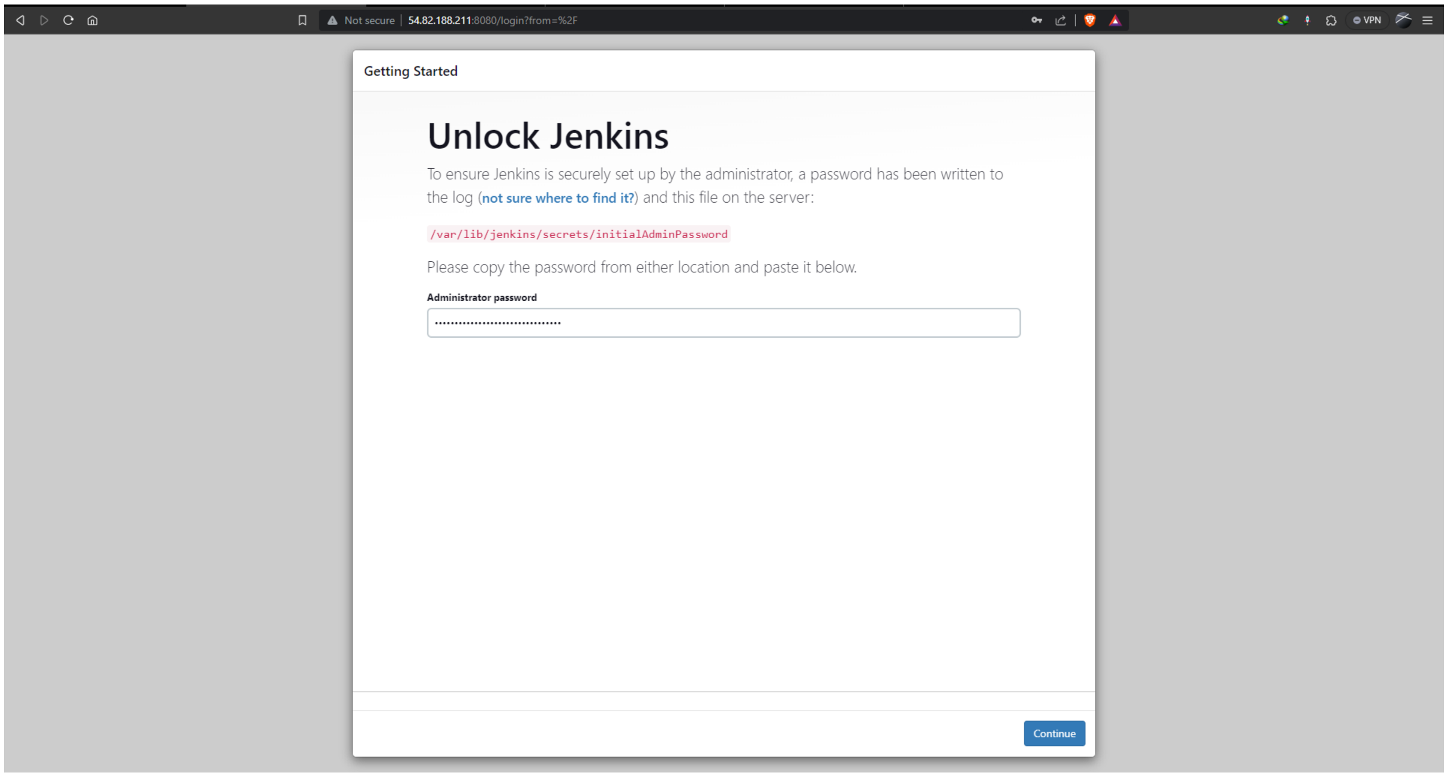
An Amazon EC2 instance is a virtual server in Amazon's Elastic Compute Cloud (EC2) for running applications on the Amazon Web Services (AWS) infrastructure.

```
resource "aws_instance" "instance" {
    ami                         = "ami-0fc5d935ebf8bc3bc"
    instance_type               = "t2.medium"
    associate_public_ip_address = true
    availability_zone           = "us-east-1a"
    subnet_id                   = aws_subnet.my_subnet.id
    vpc_security_group_ids      = [aws_security_group.my_security_group.id]
    tags                        = {Name = "EC2 Instance"}
    key_name                    = "project_access_key"

    connection {
        type                    = "ssh"
        user                    = "ubuntu"
        host                    = aws_instance.instance.public_ip
        private_key             = tls_private_key.rsa.private_key_pem
    }
    provisioner "file" {
        source      = "installation-scripts"
        destination = "/tmp/installation-scripts"
    }
    provisioner "remote-exec" {
        inline = [
            "chmod +x /tmp/installation-scripts/packages.sh",
            "/tmp/installation-scripts/packages.sh",
        ]
    }
}
```



```
pipeline {
    agent any
```

```
aws_instance.instance (remote-exec): Scanning processes... [=============== ]
aws_instance.instance (remote-exec): Scanning processes... [=============== ]
aws_instance.instance (remote-exec): Scanning processes... [=============== ]
aws_instance.instance (remote-exec): Scanning processes... [================]
aws_instance.instance (remote-exec): Scanning processes...
aws_instance.instance (remote-exec): Scanning linux images... [             ]
aws_instance.instance (remote-exec): Scanning linux images... [====         ]
aws_instance.instance (remote-exec): Scanning linux images... [========     ]
aws_instance.instance (remote-exec): Scanning linux images... [=============]
aws_instance.instance (remote-exec): Scanning linux images...

aws_instance.instance (remote-exec): Running kernel seems to be up-to-date.

aws_instance.instance (remote-exec): No services need to be restarted.

aws_instance.instance (remote-exec): No containers need to be restarted.

aws_instance.instance (remote-exec): No user sessions are running outdated
aws_instance.instance (remote-exec):  binaries.

aws_instance.instance (remote-exec): No VM guests are running outdated
aws_instance.instance (remote-exec):  hypervisor (qemu) binaries on this
aws_instance.instance (remote-exec):  host.
aws_instance.instance: Creation complete after 4m4s [id=i-009eb8f52cb1b6df0]

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.

Outputs:

get_instance_name = "EC2 Instance"
get_public_ip = "54.82.188.211"
$
```



Not secure | 54.82.188.211:8080/login?from=%2F

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

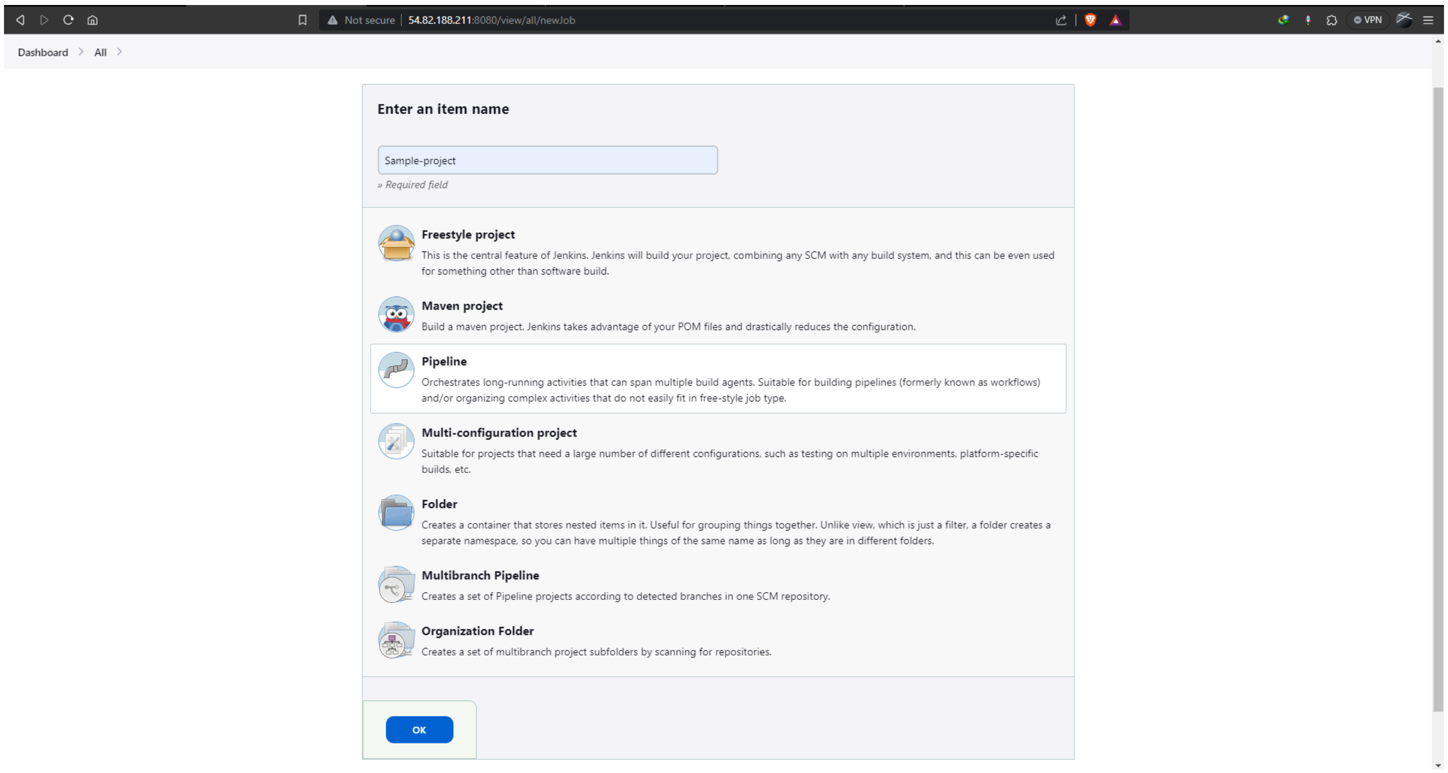
`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

••••••••••••••••••••••••••••••••

Continue


Not secure | 54.82.188.211:8080/view/all/newJob

Dashboard > All >

Enter an item name

Sample-project

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

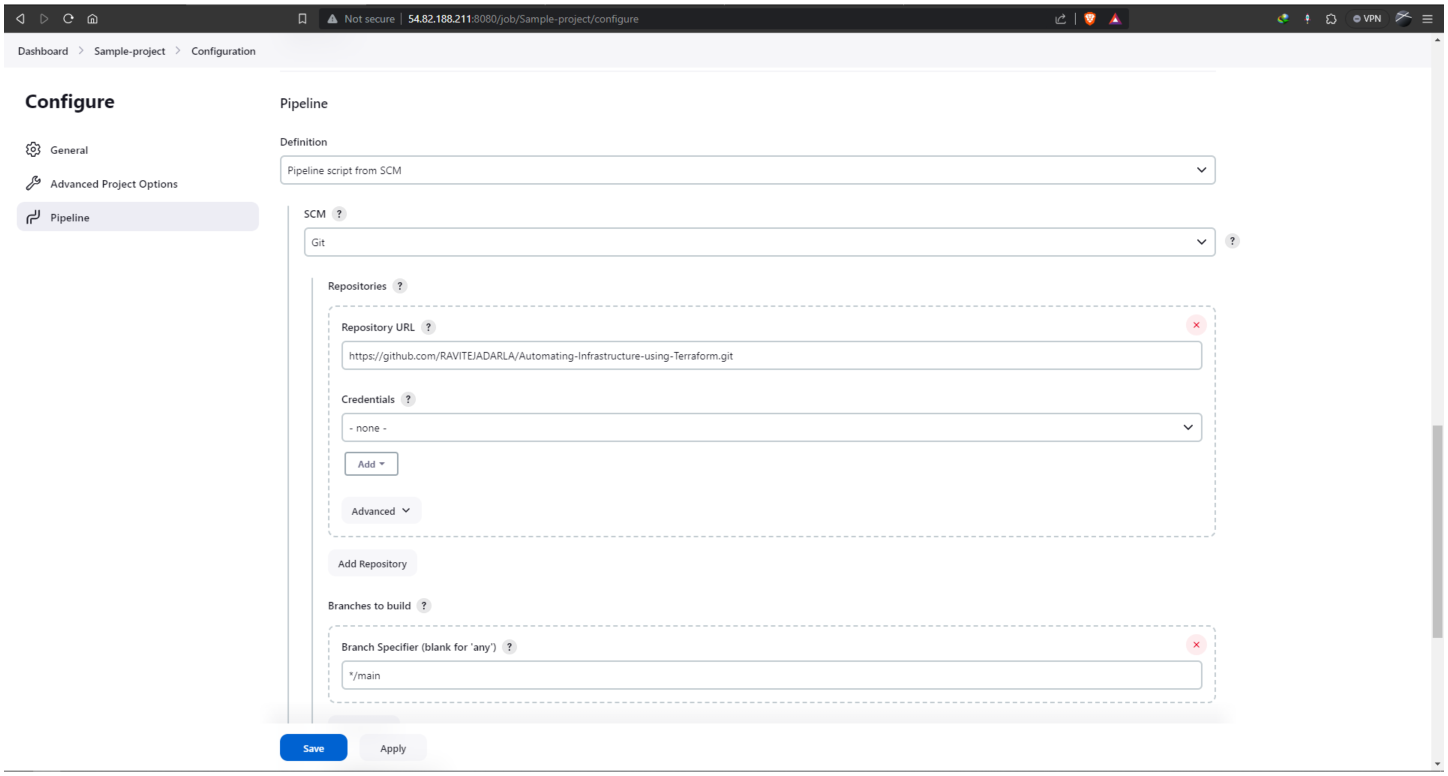
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

Organization Folder
Creates a set of multibranch project subfolders by scanning for repositories.


Not secure | 54.82.188.211:8080/job/Sample-project/configure

Dashboard > Sample-project > Configuration

Configure

Pipeline

Definition: Pipeline script from SCM

SCM: Git

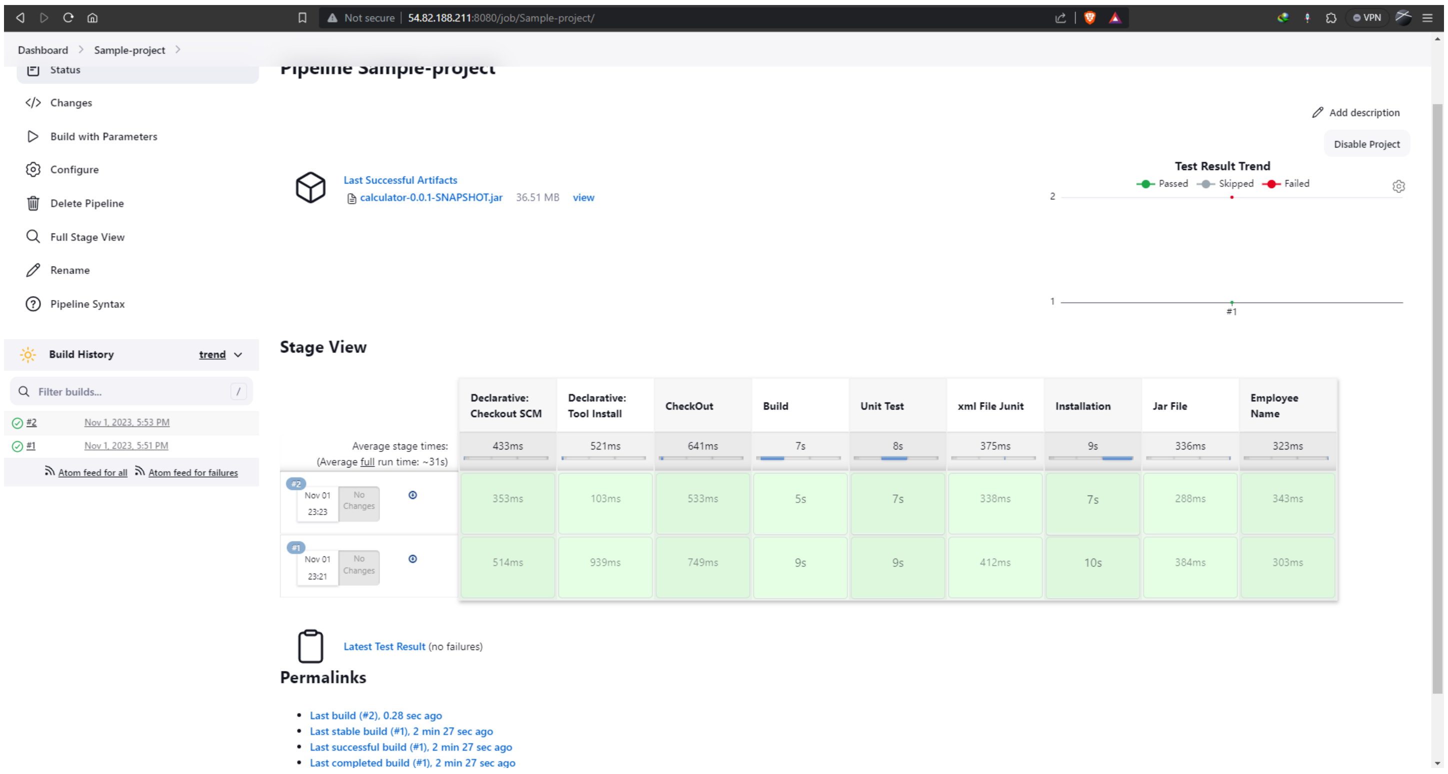
Repositories

Repository URL: https://github.com/RAVITEJADARLA/Automating-Infrastructure-using-Terraform.git

Credentials: - none -

Branches to build

Branch Specifier (blank for 'any'): */main



Not secure | 54.82.188.211:8080/job/Sample-project/

Dashboard > Sample-project >

Pipeline Sample-project

- Status
- Changes
- Build with Parameters
- Configure
- Delete Pipeline
- Full Stage View
- Rename
- Pipeline Syntax

Build History trend

Filter builds...

- #2 Nov 1, 2023, 5:53 PM
- #1 Nov 1, 2023, 5:51 PM

Atom feed for all Atom feed for failures

Add description

Disable Project

Last Successful Artifacts
calculator-0.0.1-SNAPSHOT.jar 36.51 MB view

Test Result Trend
Passed Skipped Failed

Stage View

	Declarative: Checkout SCM	Declarative: Tool Install	CheckOut	Build	Unit Test	xml File Junit	Installation	Jar File	Employee Name
Average stage times: (Average full run time: ~31s)	433ms	521ms	641ms	7s	8s	375ms	9s	336ms	323ms
#2 Nov 01 23:23 No Changes	353ms	103ms	533ms	5s	7s	338ms	7s	288ms	343ms
#1 Nov 01 23:21 No Changes	514ms	939ms	749ms	9s	9s	412ms	10s	384ms	303ms

Latest Test Result (no failures)

Permalinks

- Last build (#2), 0.28 sec ago
- Last stable build (#1), 2 min 27 sec ago
- Last successful build (#1), 2 min 27 sec ago
- Last completed build (#1), 2 min 27 sec ago

Thank You