Experiment 05 and 06: Infrastructure Automation using Terraform

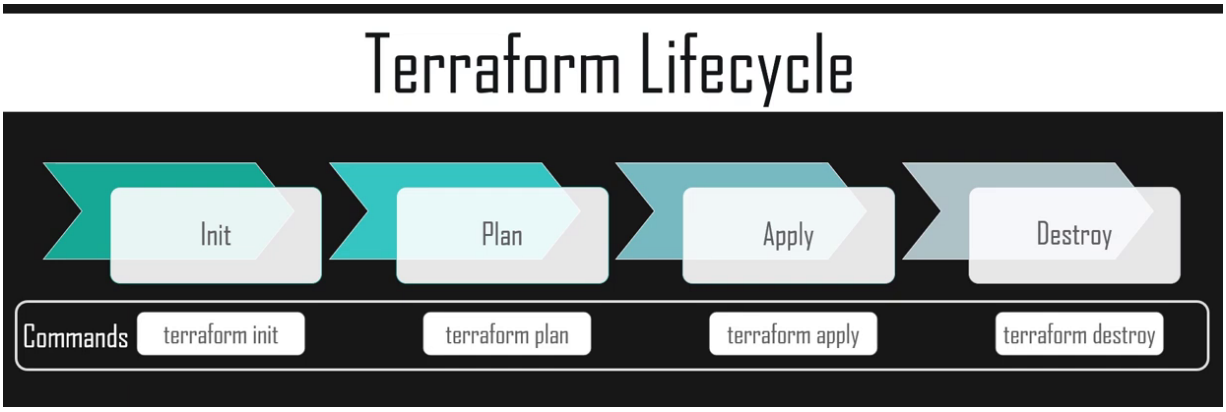
Aim 1 : To understand Terraform lifecycle, core concepts / terminologies and install it on a Linux Machine.
Aim 2 : To Build, change, and destroy AWS / GCP / Microsoft Azure / DigitalOcean infrastructure Using Terraform.

Prerequisites:
An AWS ec2 instance running ubuntu **(Note the region)**
Aws user with "**Programmatic**" access and "**Administrator**" **access**

What is Terraform, its uses:

- Open source Infrastructure as a Code(IaaC) tool developed by Hashicorp
- Server Orchestration tool
- Uses declarative Code language called HashiCode Configuration Language (HCL)

Lifecycle of Terraform:



1. Initialize:
Initializes the working directory which contains all configuration files.

2. Plan:
Used to create an execution plan to reach a desired state of the infrastructure.

3. Apply:

Makes the changes in the infrastructure as defined in the plan and brings the infrastructure to the desired state.

4. Destroy:
Delete all infrastructure resources which are created after the apply phase.

Installing Terraform on ubuntu:

***** Downloading Binary Package Link *****

```
wget https://releases.hashicorp.com/terraform/1.0.7/terraform_1.0.7_linux_amd64.zip
```

```
unzip terraform_1.0.7_linux_amd64.zip // Unzip package
```

```
sudo mv terraform /usr/local/bin/ // Move it to bin directory
```

```
terraform -v // Check terraform version
```

```
mkdir terraform_demo
cd terraform_demo
nano demo.tf
```

Create Terraform Configuration File (demo.tf):

```
provider "aws" {
  region     = "us-west-2"
  access_key = "my-access-key"
  secret_key = "my-secret-key"
}

resource "aws_instance" "terraform_ec2_example" {
  ami           = "ami-0c1a7f89451184c8b" # us-west-2
  instance_type = "t2.micro"
  tags = {
     Name = "Terraform ec2"
  }
}
```

terraform init // Initialize the Terraform directory
terraform plan // View the Plan of execution
terraform apply // Apply changes to infrastructure
terraform destroy // Destroy the defined infrastructure
Change the name of instance to see modifications applied by terraform

Experiment 07 and 08: DevSecOps - Static Application Security Testing using SonarQube

Aim 1: To understand Static Analysis SAST process and learn to integrate Jenkins SAST to SonarQube / GitLab.

Aim 2: Create a Jenkins CICD Pipeline with SonarQube / GitLab Integration to perform a static analysis of the code to detect bugs, code smells, and security vulnerabilities on a sample Web / Java / Python application.

Prerequisites: Two ec2 ubuntu instances
Machine 1(t2.micro): Jenkins
Machine 2(t2.medium): SonarQube

Static Application Security Testing (SAST):

Software artifacts are analyzed to uncover vulnerabilities in:

- Source code, binaries, Config files
- Also known as whitebox testing

Features of SAST:

- Full access to all possible scenarios
- Scaling is easy
- Developer friendly - integration with IDE

Limitations of SAST:

- Source code access required
- Will not uncover issues with operational deployment
- Large number of false positives

SonarQube:

- Open source platform developed by sonarsource

- Implemented in java
- Able to analyze above 20 programming languages

Vulnerabilities detected by SonarQube:

- Bugs: Wrong code which will probably break (Null pointer)
- Code Smells: violation of fundamental programing principles (dead / duplicate code)
- Security Vulnerability: backdoor for attackers (hard-coded passwords, SQl Wildcards)

Step 1: Installing Jenkins (Machine 1):

Perform following steps:

```
sudo apt-get update
```

```
sudo apt-get install openjdk-11-jdk
```

```
sudo apt-get upgrade
```

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
```

```
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
```

```
sudo apt-get update
```

```
sudo apt-get install jenkins
```

Step 2: Install SonarQube container(Machine 2)

Install jdk11:

```
sudo apt upgrade
```

```
sudo apt update
```

```
sudo apt-get install openjdk-11-jdk
```

Install Docker:

```
sudo apt update
```

```
sudo apt-get install docker.io
```

Install SonarQube on Docker:

```
sysctl -w vm.max_map_count=524288
```

```
sysctl -w fs.file-max=131072
```

```
ulimit -n 131072
```

```
ulimit -u 8192
```

```
docker run -d --name sonarqube -p 9000:9000 sonarqube
```

Create empty project in SonarQube:

Click on "Create project" -> "manually" -> give name "sonarqubetest2" -> click setup

Step 3: Configure SonarQube on Jenkins (Machine 1)

Install sonarqube plugin:

Go to "manage jenkins" -> "manage plugins" -> in "available section" search "SonarQube Scanner for Jenkins" -> "install without restart"

Configure sonarqube:

Go to "manage jenkins" -> "Configure System" -> in "SonarQube servers" section click on "Add SonarQube" -> in name type "sonar" -> for url type http://<ip-of-sonarqube>:9000 -> Save

Configure Global tools:

Go to "manage jenkins" -> "Global Tool Configuration" -> in "SonarQube Scanner" section click on "SonarQube Scanner installations" -> Give name "sonar" -> click "install automatically" -> Save

Step 4: Create jenkins Project

Perform following steps:

Create new project -> name it "sonarproject1"

In project configurations go to "Source Code Management" -> select "git" -> type repository url as: "https://github.com/gaikwadninad89/allinone.git"

In "Build" section -> "Add Build Step" -> "Execute SonarQube Scanner" -> in "Analysis properties" paste the following:

```
sonar.projectKey=sonarqubetest
sonar.login=admin
sonar.password=admin
sonar.exclusions=vendor/**, storage/**, resources/**,,
**/*.java
sonar.sources=/var/lib/jenkins/workspace/sonarproject1
```

Save and build the project

Check output on SonarQube Server

Experiment 09 & 10: DevSecOps - Nagios

Aim: To Understand Continuous monitoring and Installation and configuration of Nagios Core, Nagios Plugins and NRPE (Nagios Remote Plugin Executor) on Linux Machine.

Aim: To perform Port, Service monitoring, Windows/Linux server monitoring using Nagios.

Installing Nagios Core: Source Link

Step 1: Security Enhanced linux (Disabled by default) - If you would like to see if it is installed run the following command:

```
sudo dpkg -l selinux*
```

Step 2: Script "InstallNagiosCore.sh"

```
********* FileName: "InstallNagiosCore.sh" ********
#Install the pre-requisite packages:
#===== Ubuntu 20.x =====
sudo apt-get upgrade -y
sudo apt-get update
sudo apt-get install -y autoconf gcc libc6 make wget unzip
apache2 php libapache2-mod-php7.4 libgd-dev

#Downloading the Source:
cd /tmp
wget -O nagioscore.tar.gz
https://github.com/NagiosEnterprises/nagioscore/archive/nagio
s-4.4.6.tar.gz
tar xzf nagioscore.tar.gz

#Compile:
cd /tmp/nagioscore-nagios-4.4.6/
sudo ./configure --with-httpd-conf=/etc/apache2/sites-enabled
sudo make all

#Create User And Group
#This creates the nagios user and group. The www-data user is
also added to the nagios group.
sudo make install-groups-users
sudo usermod -a -G nagios www-data
```

```
#Install Binaries
#This step installs the binary files, CGIs, and HTML files.
sudo make install

#Install Service / Daemon
#This installs the service or daemon files and also
configures them to start on boot.
sudo make install-daemoninit

#Install Command Mode
#This installs and configures the external command file.
sudo make install-commandmode

#Install Configuration Files
#This installs the *SAMPLE* configuration files. These are
required as Nagios needs some configuration files to allow it
to start.
sudo make install-config

#Install Apache Config Files
#This installs the Apache web server configuration files and
configures Apache settings.
sudo make install-webconf
sudo a2enmod rewrite
sudo a2enmod cgi

#Configure Firewall
#You need to allow port 80 inbound traffic on the local
firewall so you can reach the Nagios Core web interface.
sudo ufw allow Apache
sudo ufw reload

#Create nagiosadmin User Account
#You'll need to create an Apache user account to be able to
log into Nagios.
#The following command will create a user account called
nagiosadmin and you will be prompted to provide a password
for the account.
sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users
nagiosadmin

#Start Apache Web Server
#===== Ubuntu 15.x / 16.x / 17.x / 18.x / 20.x =====
#Need to restart it because it is already running.
sudo systemctl restart apache2.service
```

```
#Start Service / Daemon
#This command starts Nagios Core.
#===== Ubuntu 15.x / 16.x / 17.x / 18.x / 20.x =====
sudo systemctl start nagios.service

#Installing The Nagios Plugins
#Prerequisites
#Make sure that you have the following packages installed.
sudo apt-get install -y autoconf gcc libc6 libmcrypt-dev make
libssl-dev wget bc gawk dc build-essential snmp
libnet-snmp-perl gettext

#Downloading The Source
cd /tmp
wget --no-check-certificate -O nagios-plugins.tar.gz
https://github.com/nagios-plugins/nagios-plugins/archive/rele
ase-2.3.3.tar.gz
tar zxf nagios-plugins.tar.gz

#Compile + Install
cd /tmp/nagios-plugins-release-2.3.3/
sudo ./tools/setup
sudo ./configure
sudo make
sudo make install

#Test Plugins
#Point your web browser to the ip address or FQDN of your
Nagios Core server, for example:
#http://10.25.5.143/nagios
#http://core-013.domain.local/nagios

#Service / Daemon Commands
#===== Ubuntu 15.x / 16.x / 17.x / 18.x / 20.x =====
sudo systemctl start nagios.service
sudo systemctl stop nagios.service
sudo systemctl restart nagios.service
sudo systemctl status nagios.service
```

Step 3: Run Script

```
sh InstallNagiosCore.sh
```

Installing Nagios Plugins: Source Link

Step 1: Script "InstallNagiosPlugins.sh"

```
#**** Installing Plugins for Nagios ****
#*** Ubuntu ***
#*** Prerequisites - Common ***
#These are the common set of packages required for compiling
most of the plugins. SNMP and required modules are included
here; they are one of the most common types of network
monitoring.
sudo apt-get update
sudo apt-get install -y autoconf gcc libc6 libmcrypt-dev make
libssl-dev wget bc gawk dc build-essential snmp
libnet-snmp-perl gettext

#Prerequisites - check_pgsql
#This is required for the check_pgsql plugin.
sudo apt-get install -y libpqxx3-dev

#Prerequisites - check_dbi
#This is required for the check_dbi plugin.
sudo apt-get install -y libdbi-dev

#Prerequisites - check_radius
#This is required for the check_radius plugin.
#Ubuntu 14.x / 15.x / 16.x
sudo apt-get install -y libfreeradius-client-dev

#Prerequisites - check_ldap
#This is required for the check_ldap plugin.
sudo apt-get install -y libldap2-dev

#Prerequisites - check_mysql check_mysql_query
#This is required for the check_mysql and check_mysql_query
plugins.
sudo apt-get install -y libmysqlclient-dev

#Prerequisites - check_dig check_dns
#This is required for the check_dig and check_dns plugins.
sudo apt-get install -y dnsutils

#Prerequisites - check_disk_smb
#This is required for the check_disk_smb plugin.
sudo apt-get install -y smbclient

#Prerequisites - check_game
```

```
#This is required for the check_game plugin.
#This package comes from the EPEL repository (EPEL was
enabled in the "Prerequisites - Common" section).
sudo apt-get install -y qstat

#Prerequisites - check_fping
#This is required for the check_fping plugin.
#This package comes from the EPEL repository (EPEL was
enabled in the "Prerequisites - Common" section).
sudo apt-get install -y fping

#Prerequisites - check_mailq
#This is required for the check_mailq plugin.
sudo apt-get install -y qmail-tools

#Prerequisites - check_flexm
#The check_flexm plugin requires lmstat from Globetrotter
Software to monitor flexlm licenses. This is a commercial
product,  you will need to contact them for instructions on
how to install lmstat on your OS.
#Downloading the Source
cd /tmp
wget --no-check-certificate -O nagios-plugins.tar.gz
https://github.com/nagios-plugins/nagios-plugins/archive/rele
ase-2.3.3.tar.gz
tar zxf nagios-plugins.tar.gz

#Compile + Install
cd /tmp/nagios-plugins-release-2.3.3/
sudo ./tools/setup
sudo ./configure
sudo make
sudo make install

#Plugin Installation Location
echo "The plugins will now be located in
/usr/local/nagios/libexec/."
cd /usr/local/nagios/libexec
ls
```

Step 2: Run Script

```
sh InstallNagiosPlugins.sh
```

Installing NRPE: Source Link

Step 1: Script "InstallNRPE.sh"

```
#Installing The NRPE Agent
#Download the Linux NRPE agent to the /tmp directory on the
Linux server you wish to monitor.
cd /tmp
wget
https://assets.nagios.com/downloads/nagiosxi/agents/linux-nrp
e-agent.tar.gz

#Unpack the installation archive you just downloaded:
tar xzf linux-nrpe-agent.tar.gz

#Enter the newly created agent subdirectory:
cd linux-nrpe-agent

#Run the wrapper script as root
sudo ./fullinstall
```

Step 2: Run Script

```
sh InstallNRPE.sh
```

Experiment 11 & 12: Serverless Computing

Aim 1: To understand AWS Lambda, its workflow, various functions and create your first Lambda functions using Python / Java / Nodejs.
Aim 2: To create a Lambda function which will log "An image has been added" once you add an object to a specific bucket in S3.

Prerequisites: An AWS account

Step 1: Create an S3 bucket

Go to S3 -> click "Create bucket" -> Give it a name and save it

Step 2: Create a lambda function to display message on image upload

Go to lambda -> click "create function" -> select "Use a blueprint" -> select "s3-get-object-python" blueprint -> click "Configure"

Give a function name "lambdafunct1" -> select "Create a new role from AWS policy templates" -> enter a role name -> make sure it has "Amazon S3 object read-only permissions" permission

In the S3 trigger section select the bucket name that you created -> in the event type select "PUT" -> in suffix you can optionally add ".jpg" -> click "create function"

Code which looks something like this will be generated:

```
import json
import urllib.parse
import boto3

print('Loading function')

s3 = boto3.client('s3')


def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))
```

```
    # Get the object from the event and show its content type
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'],
encoding='utf-8')
    try:
        response = s3.get_object(Bucket=bucket, Key=key)
        print("CONTENT TYPE: " + response['ContentType'])
        return response['ContentType']
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. Make sure they exist and your
bucket is in the same region as this function.'.format(key, bucket))
        raise e
```

Below the highlighted code add a line:
print("An image has been added")

Click on "Deploy"

Step 3: Test the uploading of image

Go to S3 bucket -> upload a ".jpg" file to the bucket

Go to lambda function -> go to the "Monitor" tab -> there will be a dot on all the graphs indicating the duration of running of your lambda function -> click on "View logs in CloudWatch" to view the print statement