# Advanced DevOps Lab Manual

**TE - IT**

**Odd Semester (V)**

**(2021-22)**

**By:**

**Prof. Ninad V Gaikwad**

**Prof. Vishwayogita Savalkar**

**Email:**

**ninad@mes.ac.in**

**vishwayogita@mes.ac.in**

**Website:**

**https://sites.google.com/a/mes.ac.in/ninad**

# List of Contents

**Experiment 01: Introduction to Cloud 9**

**Aim : To understand the benefits of Cloud Infrastructure and Setup AWS Cloud9 IDE, Launch AWS Cloud9 IDE and Perform Collaboration Demonstration.**

**Prerequisites:**
Account with Amazon AWS (will require **credit / debit** card information)
Opening AWS Free Tier account Video Link: Click Here

**Create new user using Identity and Access Management (IAM):**

Click on Users -> Add User -> type username "devopsuser1" -> check "Programmatic access" box and "AWS Management Console access" -> enter password -> uncheck "Require password reset" box -> click "next" button -> click "Create Group" -> enter name as "devops" -> add permissions for "Cloud9, CodePipeline, CodeCommit, CodeBuild, CodeDeploy, IAMFullAccess, AmazonS3FullAccess" with full access -> click "create group" -> select created group and click on "next" -> review and click on "create user" -> download CSV with login details (Will not be available later)

**Log out of Root user and login as newly created IAM user using details available in CSV file**

**Set up IDE on AWS Cloud9:**

Open cloud9 console:
Services -> Developer tools -> cloud9

Create a new IDE platform:
Create environment -> Give "Name" and "Description" to environment -> click "Next step" -> Keep default settings for Environment type and Instance type (Note: Instance type t2.micro is the only free type)-> Platform can be set to Ubuntu -> click "Next Step" -> Review settings and click "Create environment"

**Creating Maven based Java Web application in Cloud9 (Clone existing repository):**

In cloud9 terminal:

```
sudo apt install maven -y // Install maven if not installed already
maven -version // Check if maven is installed
```

Clone git repository for project files:

```
git clone https://github.com/gaikwadninad89/allinone.git
OR
https://github.com/aws-samples/aws-codedeploy-sample-tomcat
```

Go to allinone/src/main/webapp/ and modify index.html

Clean and build a WAR file:

```
mvn clean package
mvn clean // to delete the generated WAR file
```

**Link the cloud9 project to AWS CodeCommit Repository:**

In CodeCommit Console:

Go to CodeCommit console click on Create Repository button

Enter Name and Description for repository click on create button

Copy repository URL from Clone URL dropdown button and selecting Clone HTTPS

In Cloud9 bash:

Remove GIT repository (Project) linked:

```
git remote -v
git remote remove origin
```

Link CodeCommit repository to git bash on Cloud9:

```
git remote add origin
https://git-codecommit.us-east-1.amazonaws.com/v1/repos/allinone
```

Push changes to CodeCommit repository:

```
git add .
git status
git commit -m "Initial commit on AWS Codecommit"
git push origin main
```

Collaborating with other IAM users:

Click on “Share” button on the top right corner
Enter the name of IAM user in the Invite Members section
Accept the security warning
Login as User2 in other browser
Go to “Shared with you” section in the left menu

## Experiment 02: AWS CodeBuild, CodeDeploy and CodePipeline

**Aim: To Build Your Application using AWS CodeBuild and Deploy on S3 / SEBS using AWS CodePipeline, deploy Sample Application on EC2 instance using AWS CodeDeploy.**

### Create S3 storage (required to store the built war file):

Go to S3 -> click on "Create bucket" -> enter a name for the bucket -> select AWS region (should be as per region of CodeBuild) -> click on "Create bucket"

### Build project using CodeBuild:

Go to Codebuild console -> click on "Create build project" -> Give name to the project -> In source tab select the source provider -> select the repository name and the branch name -> In the environment tab select the Operating System type as ubuntu, Runtime and Image as 5.0 -> Keep default settings for service role -> In Buildspec tab provide buildspec file name -> in artifacts tab select artifact type as "Amazon S3" -> select the bucket created by you -> click on "Create build project"

### Provide putobject permission to service role created

Click on service role name -> go to permissions -> click on "edit policy" -> Add additional permissions -> Choose a service -> select "S3" -> in write permissions select "putobject" -> go to resources and click on "add ARN" -> provide the bucket name and for object select any -> click on save -> review policy and save changes -> click "delete version and save"

### Deploy application using CodeDeploy:

Go to CodeDeploy console  
Click on "Create application"  
Give name to your application  
Choose "compute platform" as EC2  
Click on "Create application"  
In deployment groups section click on "create deployment group"  
In service role  
Select "deployment type" as "in place"  
Set environment configuration as "Amazon EC2 instances"

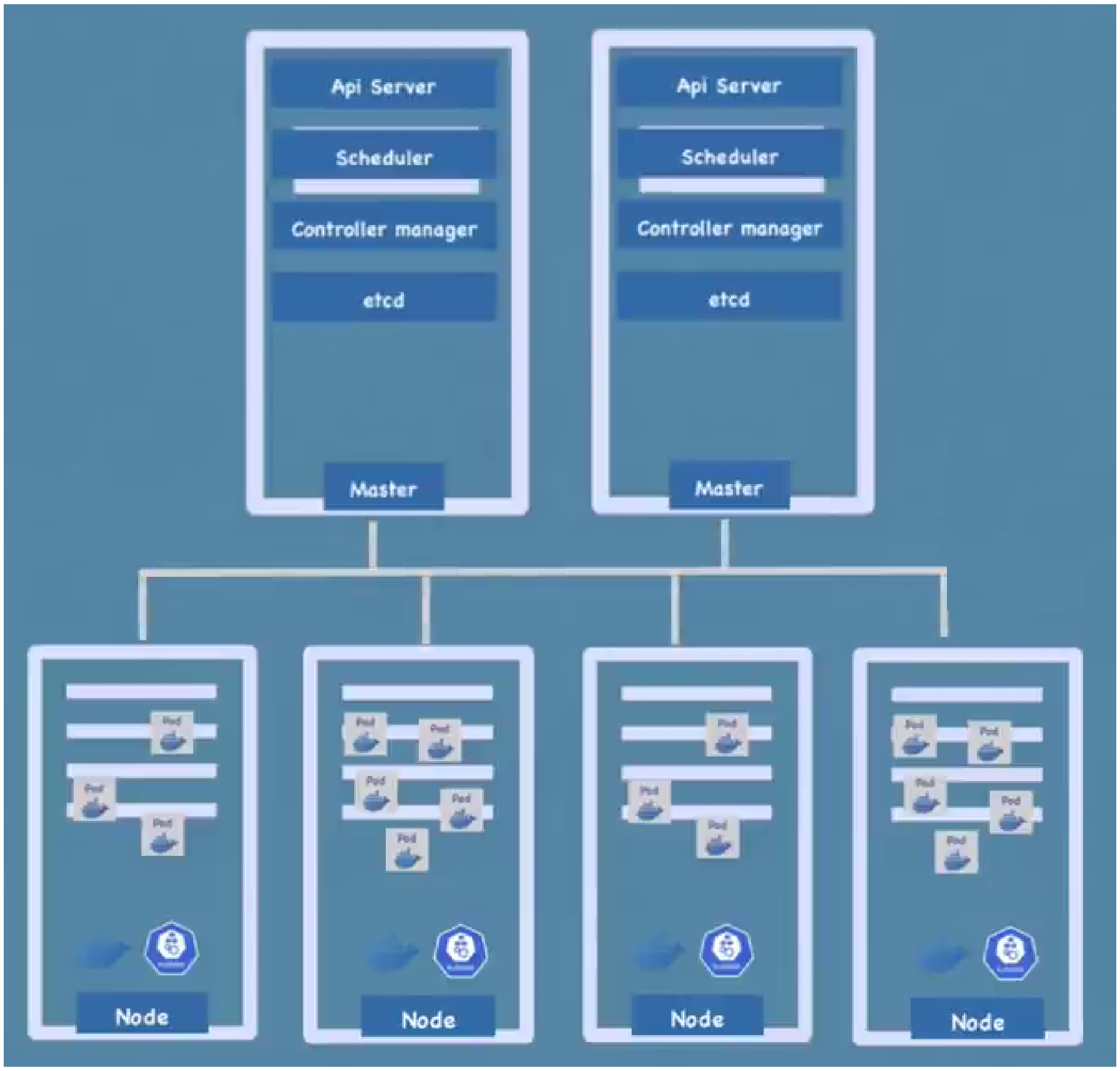
## Experiment 03: Kubernetes (K8s) Cluster Initialization

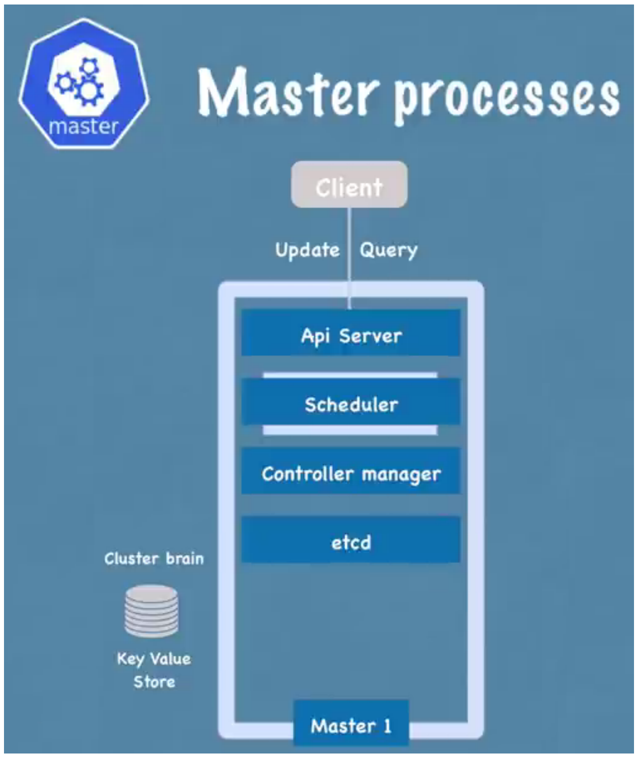
**Aim: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines / Cloud Platforms..**

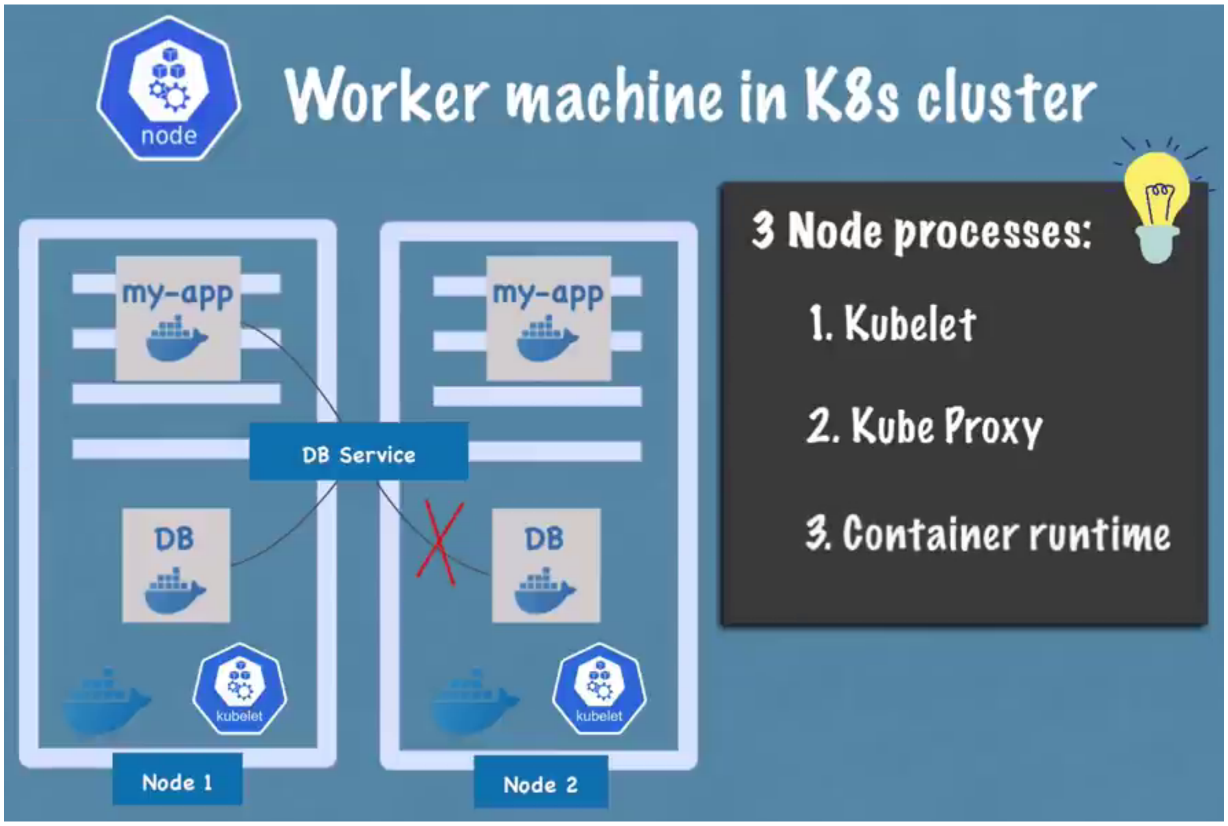
**Prerequisites:**
Two amazon ec2 / Virtualbox ubuntu instances:

- Master (2 core - CPU, 4 GB Memory)
- Worker

**K8s Architecture:**



| Master Node: |
|---|
|  master<br>Master processes<br>Client<br>Update<br>Query<br>Api Server<br>Scheduler<br>Controller manager<br>etcd<br>Cluster brain<br>Key Value Store<br>Master 1 |
| Processes of Master Node:<br>● API Server<br>○ Acts as Cluster Gateway<br>○ Interaction using UI, CLI & API<br>○ Validates and forwards requests<br>● Scheduler<br>○ Scheduling new Pods based on node usage<br>● Controller Manager<br>○ Detects changes in cluster<br>○ In case of pod failure directs scheduler which in turn directs respective worker kubelet to spin the pod<br>● etcd<br>○ Key Value store<br>○ Cluster states and changes recorded<br>■ Is the cluster healthy<br>■ What resources are available<br>■ Did the cluster state change<br>○ Does not store application data |
| |

| Worker Node: |
|---|
|   node<br>Worker machine in K8s cluster<br>my-app<br>my-app<br>DB Service<br>DB<br>DB<br>kubelet<br>kubelet<br>Node 1<br>Node 2 3 Node processes: 1. Kubelet 2. Kube Proxy 3. Container runtime |
| Processes of Master Node:<br>● Kubelet<br>○ Interacting with master node<br>○ Managing the pods on worker<br>● Kube Proxy<br>○ Communication between pods and services<br>○ Intelligent service forwarding<br>● Container Runtime<br>○ Support intended container technology |
| |
| **Installing Kubernetes (Windows):** |
| Install Docker Desktop for windows with Kubernetes:<br>Visit this link for video instructions |
| |
| **Installing Kubernetes (Linux): Both Master and Worker Nodes** |

Step 1: Create at least two AWS EC2 Ubuntu Instances
Log in via ssh:

```
ssh -i key-file-name.pem ubuntu@ip-of-aws-server
```

Step 2: Install "apt-transport-https" package

| |
|---|
| `sudo su` // to execute all commands as super user |
| `sudo apt-get update` // update apt repository |
| `sudo apt-get install apt-transport-https` // The APT transport allows the use of repositories accessed via the HTTP Secure protocol (HTTPS), also referred to as HTTP over Transport Layer Security (TLS). |

Step 3: Install Docker

| |
|---|
| `apt install docker.io -y` // Install docker engine |
| `docker --version` // check docker installation |
| `systemctl start docker` // start docker engine |
| `systemctl enable docker` // enable docker service |

Step 4: Add signing key for K8s installation

| |
|---|
| `sudo apt-get install curl` // used in command lines or scripts to transfer data |
| `sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg \| sudo apt-key add` // Since you are downloading Kubernetes from a non-standard repository, it is essential to ensure that the software is authentic. This is done by adding a signing key. |

Step 5: Add repository download location in apt

```
sudo nano /etc/apt/sources.list.d/kubernetes.list // add
kubernetes repository to apt sources list
```

```
*** Add to the file and save ***
deb http://apt.kubernetes.io/ kubernetes-xenial main
```

```
sudo apt-get update // will update apt repository to include
K8s source
```

Step 6: Install K8s components

```
sudo apt-get install -y kubelet kubeadm kubectl
kubernetes-cni
```

```
kubeadm version
```

Step 7: Configure CGroup

```
sudo swapoff -a //turn swap area off
```

```
*** In the fstab file, comment out the swap entry (by adding
a leading # character): Eg. “#/swap.img none swap sw 0 0sudo”
***
nano /etc/fstab
```

```
*** Should say that it already exists ***
sudo mkdir /etc/docker
```

```
*** Configure Docker daemon to add systemd cgroup drivers***
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
  "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
```

```
sudo systemctl restart docker
```

**Initialize K8s Cluster: (Only on Master)**

Steps:

```
kubeadm init // Will not work on t2 micro (1 cpu + 2 GB RAM)
kubeadm init --ignore-preflight-errors=all // Ignore warnings
*** Copy the join token ***
```

```
kubeadm reset // Delete cluster
```

```
*** Directory for the cluster with configurations ***
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
OR
export KUBECONFIG=/etc/kubernetes/admin.conf
```

```
*** Create Pod Network to allow communication between
different nodes in the Cluster
https://kubernetes.io/docs/concepts/cluster-administration/ad
dons/ ***
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Docum
entation/kube-flannel.yml
```

**Join Workers in K8s Cluster: (Only on Workers)**

```
*** Sample token ***
kubeadm join 172.31.12.223:6443 --token s4dsnx.4eh87eiu1r48i556
--discovery-token-ca-cert-hash
sha256:f2febcbe061454d9bda30c0852279b4fbea6f4f6fd1543315d1228ed
9f0a31ca
```

**Check joined nodes in cluster (Only on Master)**

```
kubectl get nodes
```

**Experiment 04: Running applications on Kubernetes (K8s) Cluster**

**Aim: To install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy your First Kubernetes Application.**

**Install Kubectl (Windows):**

Go to this URL -> Download the binary file -> save it in a folder named "kubectl" in C: drive

OR

Run following commands in windows Powershell:

```
Get-ExecutionPolicy -List // Displays Execution Policies
```

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser // Change current user policy to run remote scripts
```

```
Install-script -Name 'install-kubectl' -Scope CurrentUser -Force // download script to install kubectl
```

```
install-kubectl.ps1 c:\kubectl // download and install kubectl in c: drive
```

```
kubectl version --short // get version of kubectl for client and server
```

```
kubectl get nodes // Lists nodes in the cluster
```

```
kubectl drain NodeToBeRemoved // Drain all traffic to node
```

```
kubectl delete node NodeToBeRemoved // Remove a node from the cluster
```

Create a new Deployment:

What is a Deployment?

A deployment is a type of Kubernetes object that ensures there's always a specified number of pods running based on a defined template, even if the pod crashes during the cluster's lifetime. The above deployment will create a pod with one container from the Docker registry's Nginx Docker Image.

| |
|---|
| kubectl create deployment nginx --image=nginx // Creates a deployment based on nginx image |
| kubectl delete deployment nginx // Delete a Deployment |
| kubectl get deployments // List all Deployments<br>kubectl get deploy |
| |
| Create a Service: |
| What is a Service?<br>Services are another type of Kubernetes object that expose cluster internal services to clients, both internal and external. They are also capable of load balancing requests to multiple pods, and are an integral component in Kubernetes, frequently interacting with other components. |
| kubectl expose deploy nginx --port 80 --target-port 80 --type NodePort // Creates a service with random port assigned for traffic |
| kubectl delete service nginx // Deleting a Service |
| kubectl get services // Lists running services |
| |
| Verify running service:<br>Visit: http://worker_ip:service_port |
| |
| docker ps // list running containers |
| |
| Scaling Services: |
| kubectl get pods // get running pods |
| kubectl scale --current-replicas=1 --replicas=2 deployment/nginx |
| kubectl get pods |
| |
| kubectl describe deployment/nginx // Describe Deployment |

**Experiment 05 and 06: Infrastructure Automation using Terraform**

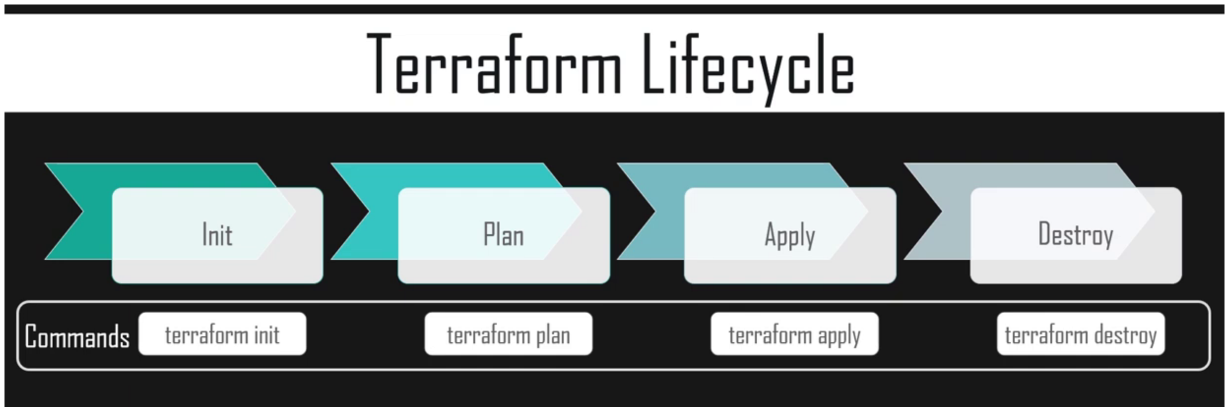
**Aim 1 : To understand Terraform lifecycle, core concepts / terminologies and install it on a Linux Machine.**
**Aim 2 : To Build, change, and destroy AWS / GCP / Microsoft Azure / DigitalOcean infrastructure Using Terraform.**

**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