Lab 1: Use terraform to setup the docker server and jenkins server for CICD Lab

Launch Ubuntu 20.04 t2.micro EC2 machine.
Note: In the security group, open ports 22, 80, 8080, 9999 and 4243.
Use the EC2 tag "CICDLab-yourname'

Task 1: Install Terraform

After the EC2 server is up & running, SSH into the machine and do the below:
```
sudo hostnamectl set-hostname CICDLab
```
```
bash
```
```
sudo apt update
sudo apt install wget unzip -y
```
```
wget https://releases.hashicorp.com/terraform/1.2.8/terraform_1.2.8_linux_amd64.zip
```
```
unzip terraform_1.2.8_linux_amd64.zip
```
```
ls
sudo mv terraform /usr/local/bin
ls
terraform -v
rm terraform_1.2.8_linux_amd64.zip
```

Task 2: Install AWS CLI and Ansible
```
sudo apt-get install python3-pip -y
sudo pip3 install awscli boto boto3 ansible
```
```
aws configure
```
example:
Access Key ID:
AKIAXMWJXSSHRD27T6SC
Secret Access Key:
H4Vh0U5oenKfmJ/+FEUcbaGbDjcnGAmZvQLX7zTT

If you need to create new credentials, Follow below steps
Go to aws console. On top right corner, click on your name or aws profile id.
when the menu opens, click on Security Credentials
Under AWS IAM Credntials, click on Create access key. If you already have 2 active keys,
you can deactivate and delete the older one so that you can create a new key

Complete aws configure step

do smoke test to check if your credentials are valid
```
aws s3 ls
```
Create hosts inventory file with the necessary permissions
 ```
 sudo mkdir /etc/ansible && sudo touch /etc/ansible/hosts
 sudo chmod 766 /etc/ansible/hosts
```

Task 3: Use terraform to launch 2 servers.

We need 2 additional servers docker-server and jenkins-server
For git-ws, we will use the anchor EC2 (from where we are operating now)
You can use t2.micro for Docker/Jenkins

Create the terraform directory and set up the config files in it
```
mkdir devops-labs && cd devops-labs
```
As a first step, create a key using ssh-keygen.
```
ssh-keygen -t rsa -b 2048
```
This will create id_rsa and id_rsa.pub in /home/ubuntu/.ssh/
Keep the path as /home/ubuntu/.ssh/id_rsa; don't set up any passphrase
Basically just hit 'Enter' key for the 3 questions it asks

Now prepare the terraform config files.
```
vi DevOpsServers.tf
```
Type the below code into DevOpsServers.tf
```
provider "aws" {
  region = var.region
}

resource "aws_key_pair" "mykeypair" {
  key_name   = var.key_name
  public_key = file(var.public_key)
}

# to create 2 EC2 instances
resource "aws_instance" "my-machine" {
  # Launch 2 servers
  for_each = toset(var.my-servers)

  ami                    = var.ami_id
  key_name               = var.key_name
  vpc_security_group_ids = [var.sg_id]
  instance_type          = var.ins_type

  # Read from the list my-servers to name each server
  tags = {
    Name = each.key
  }

  provisioner "local-exec" {
    command = <<-EOT
      echo [${each.key}] >> /etc/ansible/hosts
      echo ${self.public_ip} >> /etc/ansible/hosts
    EOT
  }
}
```

Now, create the variables file with all variables to be used in the main config file.
```
vi variables.tf
```
Add following contents into variables.tf. Please update the keyname and sg_id below.
```
variable "region" {
    default = "us-east-1"
}

# Change the SG ID. You can use the same SG id used for your CICD anchor server
```

```
# Basically the SG should open ports 22, 80, 8080, 9999 and 4243
variable  "sg_id" {
    default = "sg-06dc8863d3ed3d280" # us-east-1
}

# Choose a free tier Ubuntu AMI. You can use below.
variable "ami_id" {
    default = "ami-04505e74c0741db8d" # us-east-1; Ubuntu
}

# We are only using t2.micro for this lab
variable "ins_type" {
    default = "t2.micro"
}

# Replace 'yourname' with your first name
variable key_name {
    default = "yourname-CICDlab-key-pair"
}

variable public_key {
    default = "/home/ubuntu/.ssh/id_rsa.pub"   #Ubuntu OS
}

variable "my-servers" {
  type    = list(string)
  default = ["jenkins-server", "docker-server"]
}
```
Edit the security group id and keyname in variables.tf

Now, execute the terraform config files to launch the servers
```
terraform init
```
```
terraform fmt
```
```
terraform validate
```
```
terraform plan
```
```
terraform apply -auto-approve
```
After the terraform code is executed, check hosts inventory file and ensure below output (sample)
```
sudo vi /etc/ansible/hosts
```
It will show ip addresses of jenkins server and docker server as below.

```
[jenkins-server]
44.202.164.153
[docker-server]
34.203.249.54
```

Now ssh into docker-server & jenkins-server and check they are accessible
```
ssh ubuntu@<Jenkins ip address>
```
Set the hostname
```
sudo hostnamectl set-hostname Jenkins
```
```
```

```
ssh ubuntu@<Docker ip address>
```
```
sudo hostnamectl set-hostname Docker
```

Task 4: Use Ansible to deploy respective packages into each of the 3 servers
```
cd ~
mkdir ansible && cd ansible
```
Download the playbook which will deploy packages into the servers.
```
wget https://devops-e-e.s3.ap-south-1.amazonaws.com/DevOpsSetup.yml
```
Now, run the above playbook to deploy the packages
```
ansible-playbook DevOpsSetup.yml
```
At the end of this step, the docker-server and jenkins-server will be ready for the
Devops labs

Check if Jenkins landing page is appearing:
http://44.202.164.153:8080/ # Use your respective ip address

Check if docker is working
http://34.203.249.54:4243/version # Use your respective ip address
http://44.208.26.120:4243/version

Lab 2: Git operations

Crteate the github repository based on the method shown in the course document
Create an empty repository in your github account. Name = hello-world

After that, lets operate in local Git repository

Initializing the local git repository and committing changes

On the CICD anchor EC2, do the below:
```
cd ~/
```
Check GIT version.
```
git --version
```
If it does not exist, then you can install with below command. Else no need to execute below line:
```
sudo apt install git -y
```
Download the Java code we are going to use in the CICD pipeline
```
wget https://devops-e-e.s3.ap-south-1.amazonaws.com/hello-world-master.zip
```
```
unzip hello-world-master.zip -d hello-world-master
```
```
ls
rm hello-world-master.zip
cd hello-world-master
ls
```
```
git init .
```

```
Check the email and user name configured.
```
git config user.email
git config user.name
```
If you need to change it, you can use below:
```
git config --global user.email "< Add your email >"
git config --global user.name "< Add your Name >"
```
```
git status
git add .
git status
```
```
git log
git commit -m "This is the first commit"
git log
git status
```
```
git remote add origin < Replace your Repository URL >
```
Ex: git remote add origin https://github.com/karthiksen/hello-world.git
```
git remote show origin
```

### Task 3: Pushing the Code into your Remote GitHub Repository

To push code to Github, You need to generate a Personal Access Token (PAT) in github.
Go to your Github homepage. Click on settings in right side top menu. Click on Developer settings on
left side menu bottom. Click on Personal Access Token on left side menu. Click on Generate New Token
Under 'Select Scopes' select all items. Click on 'generate token'. Copy the token

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
```
git push origin master
```
when it asks for password, enter the PAT Token
Token: <enter your PAT>
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 # Use your respective PAT
When you enter the token, the cursor does not move. It's the expected behavior

### Task 4: Creating a Git Branch and Pushing into the Remote Repository

```
git branch dev
git branch
git checkout dev
git branch
```
```
vi index.html
```
Press INSERT and add the below content
```
<html>
<body>
<h1> Hi There! This file is added in dev branch </h1>
</body>
</html>
```
```

Save vi using ESCAPE + :wq!
```
git status
git add index.html
git status
```
```
git commit -m "adding new file index.html in new branch dev"
git log
```
```
git push origin dev
```
when it asks for password, enter PAT Token
Token: <your PAT>
```
git branch
git branch prod
git branch
git checkout prod
git branch
git merge dev
git push origin prod
```
Token: <PAT>
```
git checkout master
git merge prod
git push origin master
```

After this, you have to complete Jenkins setup and Docker setup. You can refer to the course document which gives screenshots

Lab 3: Configure Jenkins

Copy private key to Jenkins server so that we can SSH into docker server from jenkins server
```
ansible jenkins-server -m copy -a "src=/home/ubuntu/.ssh/id_rsa dest=/home/ubuntu/.ssh/id_rsa" -b
```
SSH into the Jenkins server and get the initial password for Jenkins
```
ssh ubuntu@3.93.145.99
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```
afbe8d33e25b4b908c0b9f91546f09e6

Now, go to the browser and enter Jenkins URL
http://3.93.145.99:8080/
http://54.227.66.155:8080/

Under Unlock Jenkins, enter the above Initial password. Continue.
Click on Install suggested Plugins on Customize Jenkins page.

Once the plugins are installed, it gives you the page where you can create new admin user id and password.

Enter user id and password. Save & Continue.
To keep things easier for this training, you may use the same user id and password which you used for Github. Needless to say, in real life, you must keep the user ids and passwords separate.

In next step, on Instance Configuration Page, verify your Jenkins Public IP and Port Number then click on Save and Finish

You will be prompted to the Jenkins Home Page

Click on Manage Jenkins > Manage Plugins

Under Manage Plugins, go to the Available tab and search for Maven. Select the Maven Integration Plugin and Unleash Maven Plugin and click Install without restart

Once the installation is completed, click on Go back to the top page

After going back to the Home Page select Manage Jenkins and go to Global Tool Configuration

Inside Global Tool Configuration, look for Maven, and under Maven installations, click Add Maven. Give the Name as Maven, choose Version as 3.8.4 and Save the configuration

Now you need to make a project for your application build, for that select New Item from the Home Page of Jenkins

Enter an item name as hello-world and select the project as Maven Project and then click OK

After you click on OK, you will be prompted to the configure page inside hello-world project. Go to Source Code Management tab, select Source Code Management as Git You need to provide the GitHub Repository URL and GitHub Account Credentials. In the Credentials field, you have to click Add then click on Jenkins.

Then you will get prompted to Jenkins Credentials Provider page of Jenkins. Under Add Credentials, you can add your GitHub Username, Password and Description. Then click on Add

After returning to the Source Code Management Page, click on Credentials and Choose your GitHub Credentials.

Keep all the other values as default and select Build Tab and inside Goals and options put a line as a clean package and save the configuration by clicking on Save.

'clean package' command clears the target directory and Builds the project and packages the
resulting JAR file into the target directory.

You will get back to Maven project hello-world click on Build Now for building the .war file for your application

You can go to Workspace > dist folder to see that the .war file is created there.
war file will be created in /var/lib/jenkins/workspace/hello-world/target/

Task 2: Installing and Configuring Tomcat for Deploying our Application on Jenkins server

SSH into Jenkins server. Make sure that you are root user and Install Tomcat web server
ssh ubuntu@3.87.66.176 # If you are already in Jenkins, this step not needed
```
sudo apt update
```
```
sudo apt install tomcat9 tomcat9-admin -y
ss -ltn
```
```
sudo systemctl enable tomcat9
```
Now we need to navigate to server.xml to change the Tomcat port number from 8080 to 9999,
as port number 8080 is already being used by Jenkins website.
```
sudo vi /etc/tomcat9/server.xml
```
Change 8080 to 9999 in 1 place. (There are 2 other references in comments)
press esc
:g/8080/s//9999/g hit enter
/9999 hit enter to verify

Now restart the system for the changes to take effect
```
sudo service tomcat9 restart
sudo service tomcat9 status
```
Once the Tomcat service restart is successful, go to your web browser and enter Jenkins Server IP address followed by 9999 port, you can check the Tomcat running on port 9999 on the same machine. http://< Your Jenkins Public IP >:9999
http://184.72.112.155:9999

We need to copy the .war file created in the previous Jenkins build from Jenkins workspace to tomcat webapps directory to serve the web content
```
sudo cp -R /var/lib/jenkins/workspace/hello-world/target/hello-world-war-1.0.0.war /var/lib/tomcat9/webapps
```
Once this is done, go to your browser and enter Jenkins Server Public IP address followed by port 9999 and path Ã¢â‚¬Ëœhello-world-war-1.0.0Ã¢â‚¬â„¢ and you can see tomcat is now serving your web page
(URL: http://< Your Jenkins Public IP >:9999/hello-world-war-1.0.0/)

http://184.72.112.155:9999/hello-world-war-1.0.0/

Stop tomcat9 and remove it. Else it will slow down Jenkins
```
sudo service tomcat9 stop
sudo apt remove tomcat9 -y
```

Lab 4: Using GitWebHook to build your code automatically using Jenkins

Task 1:Configure Git WebHook in Jenkins

Go to Jenkins webpage. Manage Jenkins > Manage Plugins

Go to Available Tab, Search for GitHub Integration. Click on the GitHub Integration Plugin and then on Install without restart

Once the installation is completed, click on Go back to the top page

In your hello-world project, Click on Configure. Go to Build Triggers and enable GitHub hook trigger for GITScm polling. Then Save

Go to your GitHub website, and inside hello-world repository under Settings Tab, then Webhooks. Click on the Add webhook

Fill the details as below.
Payload URL : http://<<jenkins-publicIP>>/github-webhook/

http://184.72.112.155:8080/github-webhook/
Content type: application/json
Click Add webhook

Task 2: Verify the working of WebHook by editing the Source Code

Now change your source code in the hello-world repository by editing hello.txt file.
Make a minor change and commit

As the source code got changed, Jenkins will get triggered by the WebHook and will start building the
new source code. Go to Jenkins and you can see a build is happening.

Observe successful load build in Jenkins page.

Lab 5: Add Docker Machine as Jenkins Slave, build and deploy code in Docker Host as a container

Go to Jenkins' home page and click on the Manage Jenkins option on the left.
Click on Manage Nodes and Clouds option

Click on the option New Node in the next window. Give the node name as docker-slave and then click on the ok button. Select 'permanent agent'

Fill out the details for the node docker-slave as given below. The name should be given as docker-slave, Remote Root Directory as Ëœ/home/ubuntu/, labels to be docker-slave, usage to be given as use this node as much as possible and launch method to be set as Launch agents via SSH. In the host section, give the public IP of the Docker instance.

For Credentials for this Docker node, click on the dropdown button named Add and then click on Jenkins; then in the next window, select kind as SSH username and private key give username as ubuntu, select Enter directly proceed to a private key value below. Click on the Add button once it is done.

You can get the private key as below: Goto your CICD anchor EC2 machine.
```
cd ~/.ssh
cat id_rsa
```
Copy the entire content including first line and last line. Paste it into the space provided for private key

In SSH Credentials, choose newly created ubuntu. Host Key Verification Strategy select Known host file Verification Strategy.

SSH into your Docker Host. Perform the below steps to create a Dockerfile in /home/ubuntu directory.
```
cd ~
```
```
vi Dockerfile
```
enter the below:
```
# Pull Base Image
FROM tomcat:8-jre8

# Maintainer
MAINTAINER "CloudThat"

# Copy the war file to images tomcat path
ADD hello-world-war-1.0.0.war /usr/local/tomcat/webapps/
```

Go to your Jenkins Home page, click on drop-down hello-world project, select Configure on the left tab. In General Tab, check Restrict where this project can be run and enter Label Expression as docker-slave

Go to Post Steps Tab, select Run only if the build succeeds then click on Add post-build step select
Execute shell from the drop-down and type the following commands in the shell and Save

execute shell commands in Jenkins:
```
cd ~
cp -f /home/ubuntu/workspace/hello-world/target/hello-world-war-1.0.0.war .
sudo docker container rm -f yourname-helloworld-container
sudo docker build -t helloworld-image .
```

```
sudo docker run -d -p 8080:8080 --name yourname-helloworld-container helloworld-image
```
Note - you may replace 'yourname' with your actual first name (line 3 and 5).

Now you can build your hello-world project by clicking on Build Now or by making a small change in Github files.

Once the loadbuild is successful, to access the tomcat server page, you can use below:

Use http:// < Your Docker Host Public IP >:8080/hello-world-war-1.0.0/ in your browser to see the website
http://<docker ip address>:8080/hello-world-war-1.0.0/
http://3.95.192.77:8080/hello-world-war-1.0.0/

Clean up the Instances

We can now terminate all the 3 instances.