**What is DEVOPS:**

Devops is the practice of operations and development engineers participating together in the entire service life cycle, from design through the development process to production support.

Devops Engineer must understand DevOps life cycle and implement right tools of automation at right place.

DevOps is a concept.. With the main moto to remove barrier between development and Operation tasks. By adopting the process, continuous integration and continuous deployment is achieved using various set of tools.

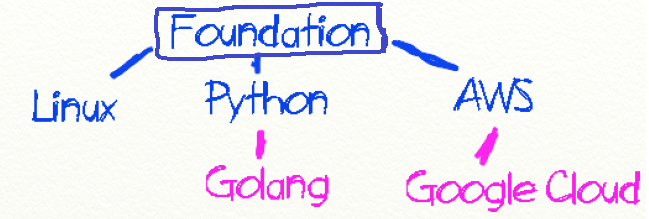
DevOps is a way to deliver software with shared pain and responsibility.

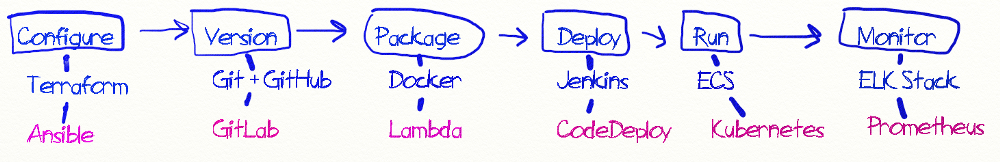
DevOps ultimately means building digital pipelines that take code from a developer’s laptop all the way to revenue generating prod awesomeness!

*It takes many years of experience, combined with a solid understanding of tools, to eventually become a truly effective Senior DevOps practitioner.*

*Goal of Devops Engineer — building a fully automated digital pipeline that takes ideas and turns them into revenue generating pieces of code.*

*The map below represents mine (and probably the majority of folks working in this space) idea of what a competent DevOps Engineer should know. You are meant to traverse this breadth-first, layer by layer. Start (and continue!) with the foundation first. Learn the technologies in blue first (Linux|Python|AWS), then if time permits or job market demands, go after the purple stuff (Golang|Google Cloud).*

****

****

Here, you will see three industry-dominant pillars: operating system, programming language, public cloud

1. Linux: where everything runs. Therefore, Linux is what you must learn and keep learning.

2. Python: the dominant back-end language these days. Easy to get started with, widely used. Bonus: Python is very prevalent in the AI/Machine Learning space, so if you ever want to transition to yet another hot field, you’ll be all set!

**3.** Amazon Web Services: Once again, it is impossible to become a seasoned DevOps professional without a solid understanding of how a public cloud works. And if knowledge of a cloud is what you are after, Amazon Web Services is the dominant player in this space, offering the richest set of tools to work with.

To save the source code, we use repository tools such as GIT, BitBucket, S3 etc

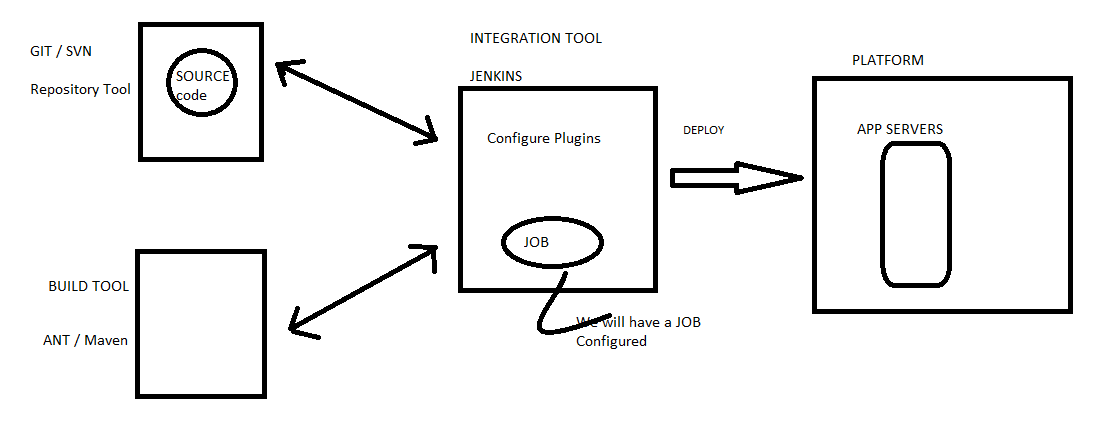
Sourcecode(.ex: java files ) itself is not directly useful . This has to be compiled to Executable code (like .CLASS files, .EXE) .. We need to understand build tools such as Maven, ANT, Gradle .

Build tools provide the artifacts such as .JAR , .WAR, .EXE files ( also knows as Artifacts) .. These artifacts need to be deployed to required environment ( dev, test, prod)..

So, we need to have knowledge on Deployment and Integration tools such as JENKINS

We shall learn

* GIT - Repository Management Tool
* DOCKER - Containers
* CHEF & ANSIBLE - Config management Tool
* JENKINS - Continuous Integration Tool
* Maven - Build Tool
* KUBERNETES - Orchestration Tools
* PACKER - provisioning Tool
* VAGRANT - Virtulization Tool
* AWS - public cloud (Iaas) which offers infrastructure as a service.

****

**Continuous Integration - Continuous Delivery**

CI / CD - continuous Delivery

CD - Deployment

CT - Continuous Testing.

Build and Release cycle -

1. Source code management - Repository for Version control and Tracking management
2. Compiling, packaging all the dependencies and creating a build . Called artifacts. ( .ear, .war, .jar)
3. These artifacts need to be placed in a location

Sourcecode Mgmt → Build& Test → ReadytoDeploy → Deployment

| ← Continous integration→ | | |

| | |

| ←---------- Continuous Delivery ----------------- →| |

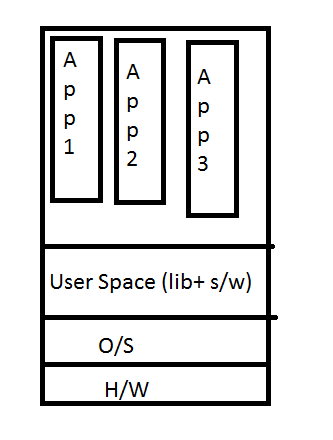
| |

| ← ----------------------------- Continuous Deployment -------------------------- →|

**Docker:**

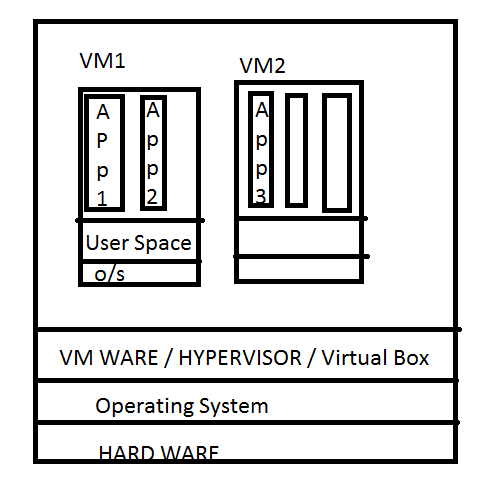
Docker is a container.. Container ensuring isolation. - running applications within its own userspace… so as not to cause impact to the other applications running on the machine.

Scenario 1: On a physical machine structure:

Any machine will have h/w .. o/s on top of it and apps on top of o/s.App3 is consuming lot of CPU process time.. In that time, O/s will get hanged and other apps will get impacted. The data generated by app logs is causing impact to the other applications running on the machine..

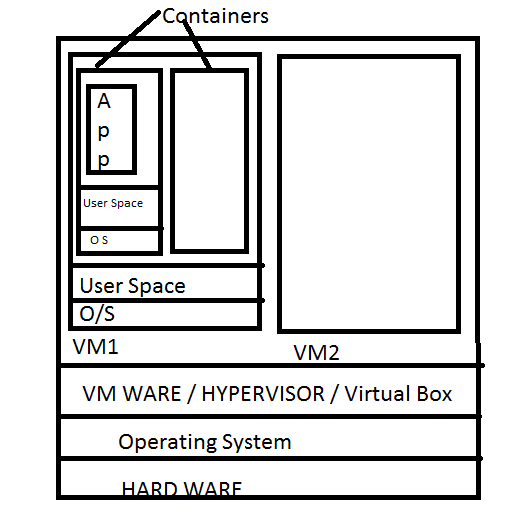
*To overcome this problem:*

On the h/w , we shall have OS and on top of OS we have VMware.. We can create multiple vm’s on it.. ; Each of the vm will have its own OS and will be assigned own set of CPU. If one of the app installed is causing problems, only that VM where the app is installed needs to be restarted.



A better approach would be to use Containers.On the h/w , we have OS and Hypervisor/VMware; On top of it, we have vm1 and vm2 and each of the VM’s will have their own set of containers.

Each of the containers will have the OS library files that are specifically needed for the appication. .. Assume the base OS as LINUX and your app runs on Ubantu, so the container will have OS libraries of Ubantu and when the app has problem, only the container is impacted and it can be restarted. It doesn’t impact other applications. And thus isolation is achieved.



* Main aim of Docker is to run applications independently in its own userspace.

i.e Isolation of applications

* OS that comes with container is very light weighted.. It only contains the o/s req libraries needed to run the applications.
* Docker as of now supports LINUX based apps.. But sooner it will be for windows too..
* Docker comes in two flavours..
  + Community edition
  + Enterprise Edition
* On AWS linux AMI, docker comes preinstalled.

*Additional Read:*

Docker relies on Linux Kernal features such as namespaces and cgroups, to ensure resource isolation and to package an application along with its dependencies. This packaging enables the application to run as expected across different linux operating systems. Its this portability that has created interest among developers.

Docker means people refer to DOCKER Engine. Docker Engine runs and orchestrates containers. The same way as Hypervisor technology that runs virtual machines, the docker engine is the core container runtime that runs the containers.

Shipping an application bundled with all the dependencies/libraries in an image without OS. -- This is the idea behind Docker. This is what container is.

If virtual machines are Hardware virtualization then containers are OS virtualization. We don’t need the real OS in our container to install our application.

If Java application is hosted inside container, it will use all the java libraries and config files from Container Data, but for compute resources its relied on the host OS kernel.

Containers are like other processes that run in an operating system, but it is isolated. It’s processes, files, libraries , configurations are contained within the boundaries of the container.

Containers have their own process tree and networking also. Every container will have an IP address and port on which the application inside the container is running.

VM has its own OS and container does not.

When you install Docker engine you get two componenets.. Docker client ; Docker Engine

Broadly there are two areas where we operate in docker engine.

Docker Images

Docker Containers

Docker Images are stopped state of containers. Image contains enough of an operating system(OS) as well as all the code to run whatever application it’s designed for. Images are built and distributed like software.

Once we have an image pulled locally on your docker host, we can use the docker run command to launch a container from it.

**LAB ACTIVITY :** Install Docker and explore Docker commands.

Spin a Amazon EC2 instance that has DOCKER and explore the following:

TO install Docker

*yum install docker*

To check Docker version

*docker --version*

To start Docker service

*service docker start*

To check the service of Docker

*service docker status*

To display the docker images that are currently available in the instance.

*docker images*

Docker pull command is used to pull the docker images. It first checks the local yum repository and if not found, pulls from Docker hub

*docker pull httpd*

Docker ps command is used to view the running containers.

*docker ps*

Docker run command is used to run the docker container.. or in other words spin/start up the container

*docker run httpd*

Note: The above command will log you into the running container, but you cannot do anything further on it. If you exit, container will also stop. In order to overcome this problem, we run docker in detached mode using the following command.

*docker run -d httpd* - To run a container in detached mode..

To see the history of previously launched containers irrespective of state, we use

*docker ps -a*

To enter into a running container, we use docker exec -it <name/id of container> /bin/bash

*docker exec -it 00109c33285a /bin/bash*

To stop a running container, we use docker stop <name / id of container>

*docker stop 00109c33285a*

To remove the container item from the history (displayed on the use of *docker ps-a* command)

*docker rm httpd*

To remove the container image

*docker rmi httpd*

**port forwarding..**

which port of base machine is mapped to which port number on container.

-p 81:80

first port number is host port number; second port number is the port number on the container; we use -p for port forwarding

In the below example, any request to port 81 on base machine will be forwarded to port 80 on the container.

*docker run -d -p 81:80 --name="myhttpdcont" httpd*

**Tip:** On ubuntu machines, index.html page is placed in htdocs. To install we use apt-get instead of yum ; instead of ***vi*** we use ***vim***

**Lab Activity - Install of SQL Container:**

From Docker hub, we have the command to run SQL container.

docker run --name some-mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql:tag

*docker run --name MySQLDB -e MYSQL\_ROOT\_PASSWORD=12345 -d -p 3306:3306 mysql*

*mysql -h 818cfc2a2762 -u root -p12345*

With the name as *MySQLDB*  and password as 12345, we run the following command to start mysql container.

( Note: when we use run , it automatically performs pull and then performs run operation

Image name from which the container is created is given as the last argument)

During the start of sql container, it expects root password to be set .. we are setting this password as an environmental variable, which is passed as an argument ( -e)

**Linking of Containers:**

When there is a dependency of applications running on different containers, we need to link the containers. WordPress application has a dependency on sql .. So, when we spin up the wordpress container, we link the dependency of sqldb.

*docker run --name mywordpress --link MySQLDB:mysql -p 8080:80 -d wordpress*

15- Nov-2018

Remove all docker containers.

docker rm -f $(docker ps -aq)

Remove all docker images

docker rmi $(docker images)

Lab - Use of DOCKER COPY command

docker cp /root/index.html my\_httpd:/usr/local/apache2/htdocs

You can have your configuration files handy on the server and push to the container using copy command.

You can also copy configuration files while building the container.

We can also copy configs from running container to your local machine

docker cp my\_httpd:/usr/local/apache2/htdocs /root/index.html

Volumes:

To create a volume:

*docker volume create my\_vol*

docker run -v my\_vol:/root /usr/local/apache2/htdocs

docker run -v my\_vol:/usr/local/apache2/htdocs/ -d -p 81:80 httpd

We are mapping /creating softlink/symlink for the content in the specified path (/usr/local/apache2/htdocs) to my\_vol

My\_vol → /usr/local/apache2/htdocs

Changes done on my\_vol will reflect in the htdocs and vice-versa.

Where is my\_vol located??

df -hk ( to list all the mounts)

User docker inspect command to find the location of my\_vol

[root@ip-172-31-30-133 ~]# docker inspect my\_vol

[

{

"CreatedAt": "2018-11-15T04:33:12Z",

"Driver": "local",

"Labels": {},

"Mountpoint": "/var/lib/docker/volumes/my\_vol/\_data",

"Name": "my\_vol",

"Options": {},

"Scope": "local"

}

]

[root@ip-172-31-30-133 ~]#

Multiple containers could be mapped to a single volume and configuration changes done on the location is reflected on the container.

This could be configuration or logs.

We use volumes to avoid explicitly logging into containers.

Try with one more opton of --mount /root/my\_container\_httpdconfig

I.e .creating a volume on the desired mount point.

Docker Compose:

Place all the required configuration files in a yml file and spin all the required containers in a single shot using this yaml file.

16-NOV-2018

Building Images:

Create a file name Dockerfile in a directory under root. ( filename is case sensitive.. We have to use exactly Dockerfile )

Docker uses DSL .. Domain specific language.. Docker is developed in Golang..

In the Dockerfile, the first command starts with FROM

vi Dockerfile

FROM ubantu ( image building it from ubantu)

.. use FROM scartch if you want to build from scratch

MAINTAINER .. owner of the Docker image..

RUN .. after RUN we will give the command that need to be executed during the build of image

MAINTAINER prasad\_vadali

RUN apt-get update -y

RUN apt-get install apache2 -y

Note: On ubantu, once the service is installed, it gets started automatically

(Note for self: HOw to find and delete swarm file???).. Temp file created on unexpected loss of connectivity during file creation.

Docker build is the command to build image..

docker build -t my\_apache\_vadali:1.0 .

144 cd docker\_projects/

145 docker build -t my\_apache\_vadali:1.0 .

146 docker images

147 docker run -d --name my\_cont my\_apache\_vadali:1.0

148 docker rm -f $(docker ps -aq)

149 docker run -d -it --name my\_cont my\_apache\_vadali:1.0

150 docker ps

151 docker exec -it my\_cont /bin/bash

-t stands for tag.

In run command, the last argument you pass should be the image name.

Explore Docker file commands..

In real time, your containers should be scalable.. Docker swarm is the cluster management of docker containers. We will have multiple machines..

On one of the machines, we initiate swarm by using swarm init command. It becomes the leader. All the other machines will become as workers.

By default, leader will also be a manager.

Docker playground is used for development of swarms.. We can create multiple docker instances.

[https://labs.play-with.docker.com](https://labls.play-with.docker.com)

$ docker swarm init --advertise-addr 192.168.0.18

Swarm initialized: current node (xzxxuacvyn68q3z8ym2ipnzar) is now a manager.

To add a worker to this swarm, run the following command:

docker swarm join --token SWMTKN-1-4bfxzvx2bogq7zuhr0xh1iybatyjjuqll537oqdiipeeqwenx4-avdnve8jd4wo50wmulpm0pvqi 192.168.0.18:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

[node1] (local) root@192.168.0.18 ~

$ docker swarm join-token manager

To add a manager to this swarm, run the following command:

docker swarm join --token SWMTKN-1-4bfxzvx2bogq7zuhr0xh1iybatyjjuqll537oqdiipeeqwenx4-9wgfz2wwtkv5zb9dl6st6cfm8 192.168.0.18:2377

[node1] (local) root@192.168.0.18 ~

To add a worker to this swarm, run the following command:

docker swarm join --token SWMTKN-1-4n3eycdab6qrqpk6klo89kj70grf00wi922vyj4dsobsa435lm-efzn6ww3qfh6d8z56ac4fz9cs 192.168.0.8:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

Docker node ls command is used to list the containers inside the swarm.

[node1] (local) root@192.168.0.18 ~

$ docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSIONxzxxuacvyn68q3z8ym2ipnzar \* node1 Ready Active Leader 18.06.1-cegtvj9nlpzpcw77fpjaf4d4tjk node2 Ready Active 18.06.1-ce

tvjf3guxvf1khpcgz4mn3pkc8 node3 Ready Active 18.06.1-ce

zykhgpbtzhw90r7ef07b0w6g6 node4 Ready Active Reachable 18.06.1-ce

wlqlgi7k7mrtnt775a86ugrtq node5 Ready Active Reachable 18.06.1-ce

Docker service ls … command to list the running services in the swarm.

Docker service create ---name my\_httpd -p 80:80 httpd

1 docker swarm init

2 docker swarm init --advertise-addr 192.168.0.18

3 docker swarm join-token manager

4 clear

5 docker node ls

6 clear

7 docker node ls

8 docker serivce ls

9 docker service ls

10 Docker service create ---name my\_httpd -p 80:80 httpd

11 docker servic create --name my\_httpd -p 80:80 httpd

12 docker service create --name my\_httpd -p 80:80 httpd

13 docker rm my\_httpd

14 docker service rm my\_httpd

Docker service my

docker service create --name my\_httpd\_swarm -p 80:80 --replicas=10 httpd

Here port forwarding is for the service..

Portainer is a docker monitoring tool that runs on

$ docker volume create portainer\_data  
$ docker run -d -p 9000:9000 --name portainer --restart always -v /var/run/docker.sock:/var/run/docker.sock -v portainer\_data:/data portainer/portainer

On a ubantu machine, install tomcat, start service and deply sample application on

/var/lib/tomcat8/webapps

Wget https://www.oracle.com/webfolder/technetwork/tutorials/obe/fmw/wls/12c/03-DeployApps/files/benefits.war

/usr/local/tomcat/webapps

apt-get docker -y

2 apt-get install docker

3 service docker start

4 service docker status

5 apt-get install docker.io

6 docker run -d -p 8383:8080 --name my\_tomcat tomcat:8.0

7 docker ps

8 mkdir my\_apps

9 cd my\_apps/

10 wget https://www.oracle.com/webfolder/technetwork/tutorials/obe/fmw/wls/12c/03-DeployApps/files/benefits.war

11 ls -lart

12 docker ps

13 docker exec -it my\_tomcat /bin/bash

14 docker cp /root/myapps/benefits.war mytomcat:/usr/local/tomcat/webapps

15 docker cp /root/my\_apps/benefits.war mytomcat:/usr/local/tomcat/webapps

16 docker exec -it my\_tomcat /bin/bash

17 docker images

18 docker cp /root/my\_apps/benefits.war tomcat:/usr/local/tomcat/webapps

19 docker cp /root/my\_apps/benefits.war tomcat:usr/local/tomcat/webapps

20 docker ps

21 docker cp /root/my\_apps/benefits.war my\_tomcat:usr/local/tomcat/webapps

Kubernetes is a tool .. containers cluster management tool.

Containers offering same functionality are grouped together as Pots.

KUBERNETES SETUP on UBUNTU

Install docker

1) apt-get update

2) apt install docker.io

Install kubernetes

Add kubenetes to apt repos

3) curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

4) echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' | sudo tee /etc/apt/sources.list.d/kubernetes.list

Install kubeadm , kubectl, kubelet

5) apt-get update

6) apt-get install kubelet kubeadm kubectl

Initialize kubernetes cluster with the below bootstrap command.

7) kubeadm init --pod-network-cidr=172.31.0.0/16 --apiserver-advertise-address=172.31.41.53

8) mkdir -p $HOME/.kube

9) sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

10) sudo chown $(id -u):$(id -g) $HOME/.kube/config

11) kubectl get nodes

you should see that Master Node is listed as not ready.

Because the cluster does not have a Container Networking Interface (CNI).

Let’s deploy a Calico CNI for the Master Node with the following command:

12) kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

13) kubectl get pods --all-namespaces

wait till the status is RUNNING .

and then again run

14) kubectl get nodes

this time it should be showing as READY.

Add Slave Node to the Kubernetes Cluster

join any number of machines by running the following on each node

as root:

create other machines with docker and kubernetes installed ( perform step 1 to step 6 only on the other machines ) and run the below join command

15) kubeadm join --token xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

on master node run the below command and check if the worker node is listed here.

16) kubectl get nodes

Deployement on the cluster

17) kubectl create deployment myhttpd --image=httpd

Create a service to run the deployment

18) kubectl create service nodeport myhttpd --tcp=80:80

list the services

19) kubectl get svc

try accesing the service in the browser with the public ip of the node and the assigned port number

apiVersion: v1

kind: Pod

metadata:

name: my-pod-demo

namespace: pod-example

spec:

containers:

- name: myhttpd-container

image: httpd

ports:

- containerPort: 80

Deployment

create a deploymeent

Update a deployment

rolling updates

Roll back to a previous version

pause/ Resume a deployment

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: httpd-deployment**

**spec:**

**replicas: 3**

**template:**

**metadata:**

**labels:**

**app: myhttpd**

**spec:**

**containers:**

**- name: myhttpd\_container**

**image: httpd**

**ports:**

**- containerPort: 80**

kubectl create –f < file name >

kubectls get deployments

kubectl get rs

kubectl get pods

kubectl get pods –show-labels

kubectl rollout status deployment/<deployment name>

kubectl expose deployment < deploymentname> --type=NodePort

kubectl get service

kubectl describe service < service name>

kubectl service <servicename> --url

kubectl set image deployment/<deploymentname > < old image name=new image name>

kubectl rollout

kubectl get pods

kubectl rollout history < deployment name>

kubectls rollout undo < deployment name > --to-revision=3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

On execution of script, environment is built.. Infrastructure as Code

Terraform provided as zip distribution and is compatible with all platforms

--TERRAFORM

Download terraform from https://www.terraform.io/downloads.html   
mkdir terraform  
unzip the .zip file  
export PATH=/e/terraform/:$PATH  
echo $PATH  
  
or on windows set in environment variables  
  
then you should be able to kcik the command terraform from any location.  
  
terraform version  
  
create an aws instance using terraform   
  
get the IAM access keys from AWS and keep it ready.  
  
mkdir my\_project  
cd my\_project  
vi create\_inst.tf  
==============================================================  
provider "aws" {  
 access\_key = "xxxxxxxxxxxxxxxxxxxxxxxxx"  
 secret\_key = "xxxxxxxxxxxxxxxxx"  
 region = "ap-south-1"  
}  
  
resource "aws\_instance" "terraform\_example" {  
 ami = "ami-xxxxxxxxxxxxxxxx"  
 instance\_type = "t2.micro"  
}  
  
==================================================================   
terraform init

terraform apply   
  
 will create an instance on your Mumbai regiuon in AWS.  
 We can check the same on AWS console.  
  
  
Variables  
  
Use variable to hide secrets  
  
Provider.tf   
========================================  
  
provider "aws" {  
 access\_key = "${var.AWS\_ACCESS\_KEY}"  
 secret\_key = "${var.AWS\_SECRET\_KEY}"  
 region = "${var.AWS\_REGION}"  
}  
vars.tf   
  
variable “AWS\_ACCESS\_KEY” {}  
variable “AWS\_SECRET\_KEY” {}  
variable “AWS\_REGION” {  
 default = “ap-south-1”  
}  
  
  
  
  
Terraform.tfvars  
  
AWS\_ACCESS\_KEY = ””  
AWS\_SECRET\_KEY = ””  
AWS\_REGION = “”  
  
  
  
Instance .tf   
resource "aws\_instance" "terraform\_example" {  
 ami = "ami-xxxxxxxxxxxxxxxx"  
 instance\_type = "t2.micro"  
}  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
Software Provisioning   
  
Instance.tf  
  
resource "aws\_key\_pair" "mykey" {  
 key\_name = "mykey"  
 public\_key = "${file("${var.PATH\_TO\_PUB\_KEY}")}"  
}  
  
resource "aws\_instance" "terraform\_example" {  
 ami = "ami-0912f71e06545ad88"  
 instance\_type = "t2.micro"  
 key\_name = "${aws\_key\_pair.mykey.key\_name}"  
   
   
 provisioner "file" {  
 source = "script,sh"  
 destination = "/tmp/script.sh"  
 }  
   
 connection {  
 user = "${var.INSTANCE\_USERNAME}"  
 private\_key = "${file("${var.PATH\_TO\_PVT\_KEY}")}"  
 }  
}

Vars.tf  
  
variable AWS\_ACCESS\_KEY {}  
variable AWS\_SECRET\_KEY {}  
variable AWS\_REGION {  
 default = "ap-south-1"  
}   
  
variable PATH\_TO\_PVT\_KEY {  
 default = "mykey"  
}  
  
variable PATH\_TO\_PUB\_KEY {  
 default = "mykey.pub  
  
variable INSTANCE\_USERNAME {  
 default = "ec2-user"  
}  
  
---------------------------------------------------------------------------------------------------------------------------------------------

Example from class:

Instance.tf  
---------------  
resource "aws\_instance" "terraform\_example" {

ami = "ami-01663519d47852113"

instance\_type = "t2.micro"

key\_name = "key\_pair"

security\_groups = ["my\_sg"]

provisioner "file" {

source = "script.sh"

destination = "/tmp/script.sh"

}

provisioner "remote-exec" {

inline = [

"chmod +x /tmp/script.sh",

"/tmp/script.sh"

]

}

connection {

user = "${var.INSTANCE\_USERNAME}"

private\_key = "${file("${var.PATH\_TO\_PVT\_KEY}")}"

}

}

Vars.tf

--------------------------------------------------------

variable "AWS\_ACCESS\_KEY" {}

variable "AWS\_SECRET\_KEY" {}

variable "AWS\_REGION" {

default = "ap-south-1"

}

variable PATH\_TO\_PVT\_KEY {

default = "key\_pair.pem"

}

variable INSTANCE\_USERNAME {

default = "ubuntu"

}

Provider.tf

----------------------------------------------------------

provider "aws" {

access\_key = "${var.AWS\_ACCESS\_KEY}"

secret\_key = "${var.AWS\_SECRET\_KEY}"

region = "${var.AWS\_REGION}"

}

Terraform.tfvars

------------------------------------------------------------------

AWS\_ACCESS\_KEY = "AKIAJSLFFBH5OYIQ6N3Q"

AWS\_SECRET\_KEY = "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"

Script.sh

----------------------------------------------------------------------

#!/bin/bash

sudo rm -rf /var/www/html/index.html

sudo touch /var/www/html/index.html

sudo chmod 777 /var/www/html/index.html

sudo echo 'This is application version.1' >> /var/www/html/index.html

End of class example  
------------------------------------------------------------------------------------------------------------------------------------------------------------

Windows Server Provisioning   
  
  
  
resource "aws\_key\_pair" "mykey" {  
 key\_name = "mykey"  
 public\_key = "${file("${var.PATH\_TO\_PUB\_KEY}")}"  
}  
  
resource "aws\_instance" "terraform\_example" {  
 ami = "ami-0912f71e06545ad88"  
 instance\_type = "t2.micro"  
 key\_name = "${aws\_key\_pair.mykey.key\_name}"  
 userdata = <<EOF  
 <powershell>  
 net user ${var.INSTANCE\_USERNAME} ${var.INSTANCE\_PASSWORD} /add  
 net localgroup administrators ${var.INSTANCE\_USERNAME} /add  
  
 winrm quickconfig –q  
 winrm set winrm/config/winrs ‘@{MaxMemoryPerShellMB=”300”}’  
 winrm set winrm/config ‘@{MaxTimeoutMS=”180000”}’  
 winrm set winrm/config/service ‘@{AllowUnencrypted=”true”}’  
 winrm set winrm/config/service/auth ‘@{Basic=”true”}’  
  
 netsh advfirewall firewall add rule name=”WinRM 5985” protocol=TCP dir=in localport=5985 action=allow  
netsh advfirewall firewall add rule name=”WinRM 5986” protocol=TCP dir=in localport=5986 action=allow  
net stop winrm  
sc.exe config winrm start=auto  
net start winrm   
</powershell>  
EOF  
  
  
  
  
  
  
  
   
 provisioner "file" {  
 source = "testing.txt"  
 destination = "C:/testing.txt"  
 }  
   
 connection {  
 type = “winrm”  
 user = "${var.INSTANCE\_USERNAME}"  
 password = "${var.INSTANCE\_PASSWORD" }  
}  
  
  
Vars.tf  
variable AWS\_ACCESS\_KEY {}  
variable AWS\_SECRET\_KEY {}  
variable AWS\_REGION {  
 default = "ap-south-1"  
}   
  
variable PATH\_TO\_PVT\_KEY {  
 default = "mykey"  
}  
  
variable PATH\_TO\_PUB\_KEY {  
 default = "mykey.pub"  
}  
  
variable INSTANCE\_USERNAME {  
 default = "ec2-user"  
}  
variable INSTANCE\_PASSWORD { }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PACKER EXAMPLE:  
  
 {  
 "variables": {  
 "aws\_access\_key": "AKIAJAAG47HITSVYQJPQ",  
 "aws\_secret\_key": "7V9UgtgWoa1qyTebgWCDluKb0v87+jz42CXQ0Zaf"  
 },  
 "builders": [{  
 "type": "amazon-ebs",  
 "access\_key": "{{user `aws\_access\_key`}}",  
 "secret\_key": "{{user `aws\_secret\_key`}}",  
 "region": "ap-south-1",  
 "source\_ami\_filter": {  
 "filters": {  
 "virtualization-type": "hvm",  
 "name": "ubuntu/images/\*ubuntu-xenial-16.04-amd64-server-\*",  
 "root-device-type": "ebs"  
 },  
 "owners": ["099720109477"],  
 "most\_recent": true  
 },  
 "instance\_type": "t2.micro",  
 "ssh\_username": "ubuntu",  
 "ami\_name": "packer-example {{timestamp}}"  
 }],  
 "provisioners": [{  
 "type": "shell",  
 "inline": [  
 "sudo apt-get update",  
 "sudo sleep 30",   
 "sudo apt-get install apache2 -y"  
 ]  
 }]  
}

Command to initialize repository:

git init <repository\_name>

Git init demo1

Where-ever .git is available, it is your repository

Add a file ..

Use command: git status

-- this shall list all the new files added to the location as untracked items.

And if any files are changed after the previous commit, these will be shown as modified.

Git add <Filename> … to add the file to the repository

To add all the files.. We use git add .

. indicates add all files.

git add test.txt

git commit - m “New file added”

For Remote repositories..

Registering on github. .. You can create your own repositories or use existing repositories.

To configure a remote repository:

git remote add origin https://github.com/vadaliprasad/gitdemo\_project.git

To push the content from current working directory to remote repository, we use

git push -u origin master

Note - Before push, please ensure that the content in the working directory(local repository) is committed.

In order to use pull command to get the repository content, you have to be in a repository

git init pull\_demo

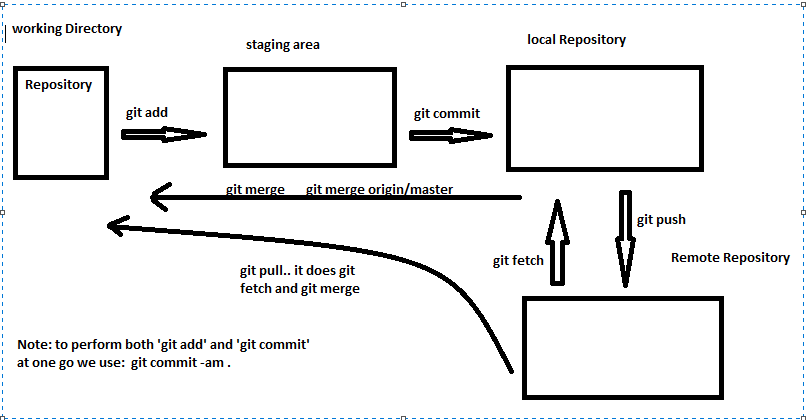
cd pull\_demo

git pull https://github.com/vadaliprasad/gitdemo\_project.git

TO see the transactions done with the repository, we use : git log

git remote add origin https://github.com/vadaliprasad/gitdemo\_project.git

205 git push -u origin master



Blame gives you the detail on who modified and what was modified.

prasad.vadali@HYD-LT-PrasadV MINGW64 ~/Desktop/AWS/git\_projects/first\_git\_repo (master)

$ git blame text2.txt

739f7b40 (Prasad Vadali 2018-11-26 10:06:47 +0530 1) This is second file

Once the file is created, it is created in working area.

git add <filename> will add it to the staging.

To remove from staging we use git rm --cached <filename>

git log --oneline

This will display the log in one liner..

Git reset <transaction\commit\_id> will reset the working directory to previous commit

Git reset head will reset to the last commit.

Git commit -am

Will only add modifications to existing files and do the commit

It doesn’t do anything on the untracked files.

Untracked files cannot be committed using git commit -am command.

HEAD always points to the last commit

Files shown in green are in staging area. .

Git rm will remove the file from the working area and add the deletion action to the staging area..

Deleted file will be displayed in green .. indicating the file ( the change) is already available in staging.

Bash rm will only remove the file from the working area , but will not track/add the deletion action to the staging area. The change is only

The behaviour is same for all bash related commands.

Branches:

Git branch … lists all the available branches.

\*(asterick) on the branch name indicates current branch

Git branch <branchname> will create a new branch.

In order to switch between branches, we use .. git checkout <branch\_name>

