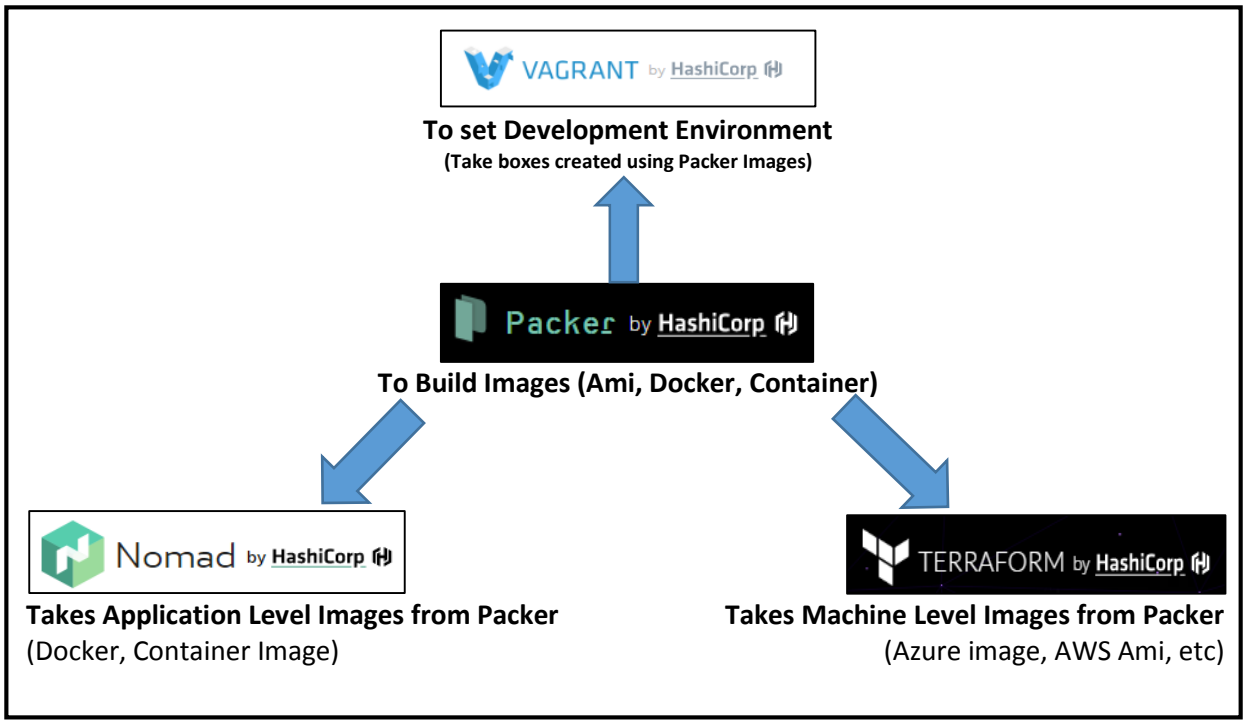
WHITEPAPER

HashiCorp

Vagrant Automated Web Server creation Using Packer and provision using Terraform.

Provision Secure and Run Any Infrastructure for Any Application

Contents



- **Vagrant** is used to create and configure lightweight, reproducible, and portable development environments.
- **Packer** is a tool for creating machine and container images for multiple platforms from a single source configuration.
- **Terraform** provides a common configuration to launch infrastructure — from physical and virtual servers to email and DNS providers.

Requirement: Integrating Vagrant, Packer and Terraform.

Pre-Requisites:

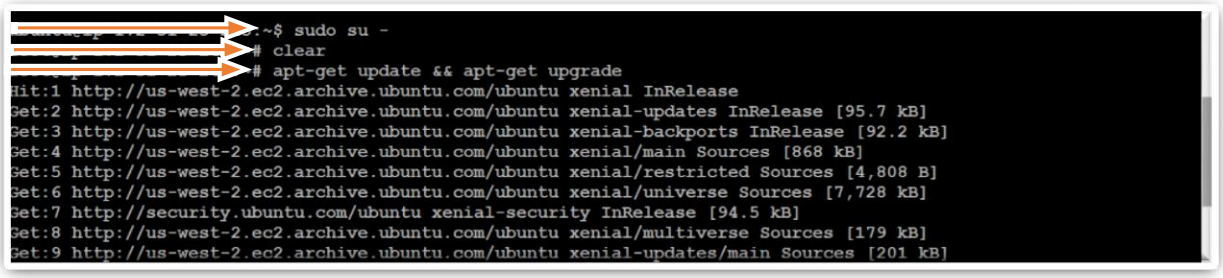
1. Ubuntu Server 16.04/14.04.
2. Min Req- RAM(1gb) and CPU(1core)
3. Public key to access the Ubuntu machine.

Steps:

1. Installing and configuring Virtual Box and Vagrant.
2. Installing and configuring Packer.
3. Creating customizable Image using Packer.
4. Installing and configuring Terraform.
5. Use Packer image and deploy/destroy servers in AWS using Terraform.
6. Using Vagrant to create/destroy development environment using the Packer image.

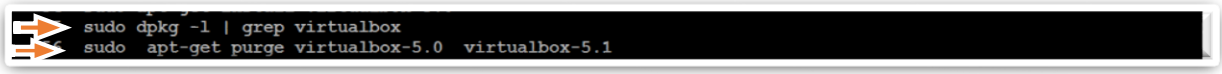
Step 1: Installing and configuring **Virtual Box** and **Vagrant**

- sudo su –
- apt-get update && apt-get upgrade



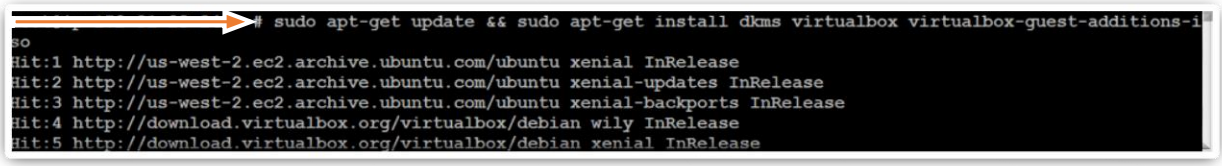
Check if any other versions of virtualbox are pre-installed and remove it.

- sudo dpkg -l | grep virtualbox
- sudo apt-get purge virtualbox-5.0 virtualbox-5.1



Installing Virtual box.

- sudo apt-get update && sudo apt-get install dkms virtualbox virtualbox-guest-additions-iso

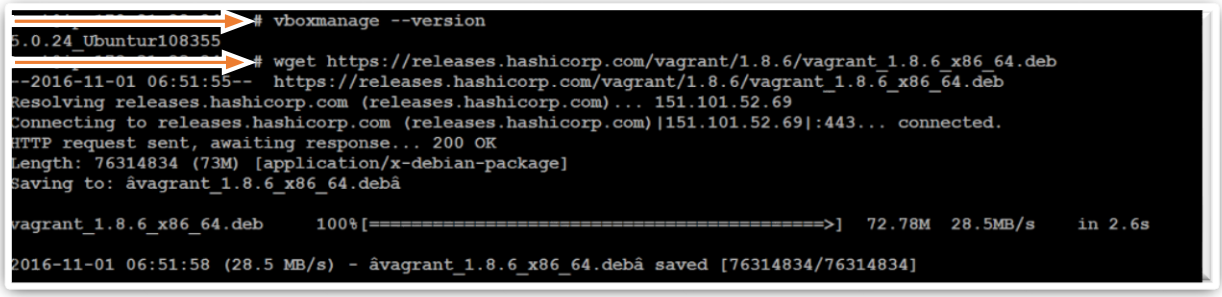


Check whether virtualbox is installed or not.

- vboxmanage –version

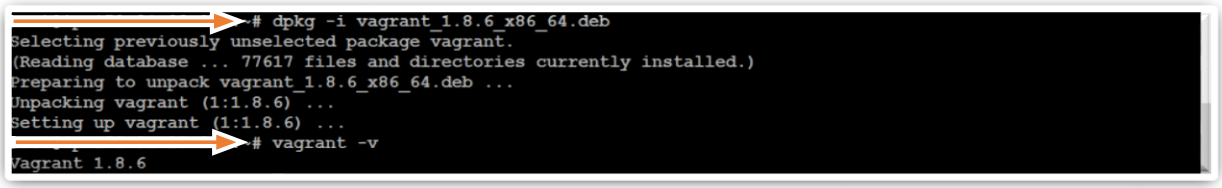
Now, installing Vagrant.

- wget https://releases.hashicorp.com/vagrant/1.8.6/vagrant_1.8.6_x86_64.deb



Depackage, Install and check whether Vagrant is installed or not.

- dpkg -i vagrant_1.8.6_x86_64.deb
- vagrant -v



Step 2: Installing and configuring **Packer**

Create a Packer directory and go inside it.

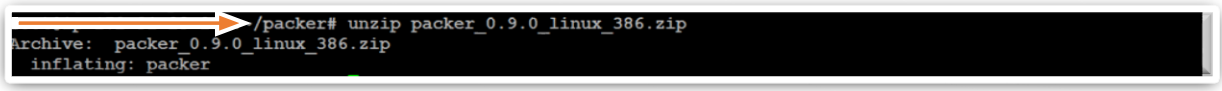
- mkdir packer
- cd packer

Download the Packer installation Zip file.

- wget https://releases.hashicorp.com/packer/0.9.0/packer_0.9.0_linux_386.zip
- apt install unzip

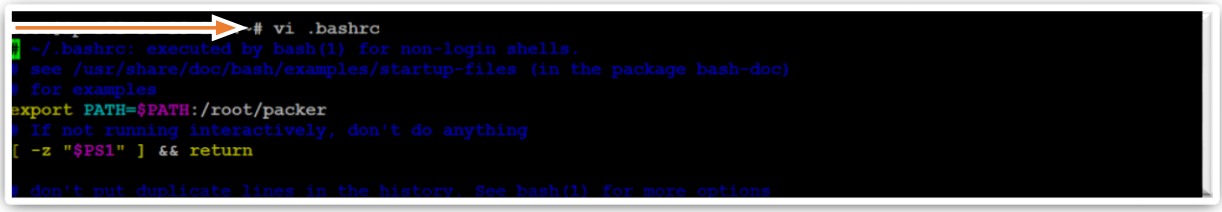
Unzip the Packer file.

- unzip packer_0.9.0_linux_386.zip



Set the packer path in .bashrc

- vi .bashrc
  - `export PATH=$PATH:/root/packer`

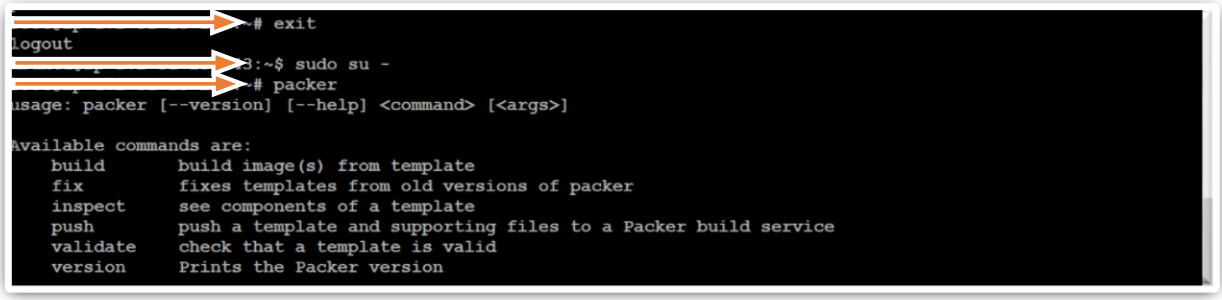


Logout of the system and again login using sudo user.

- exit
- sudo su –

Check whether Packer is installed or not.

- packer



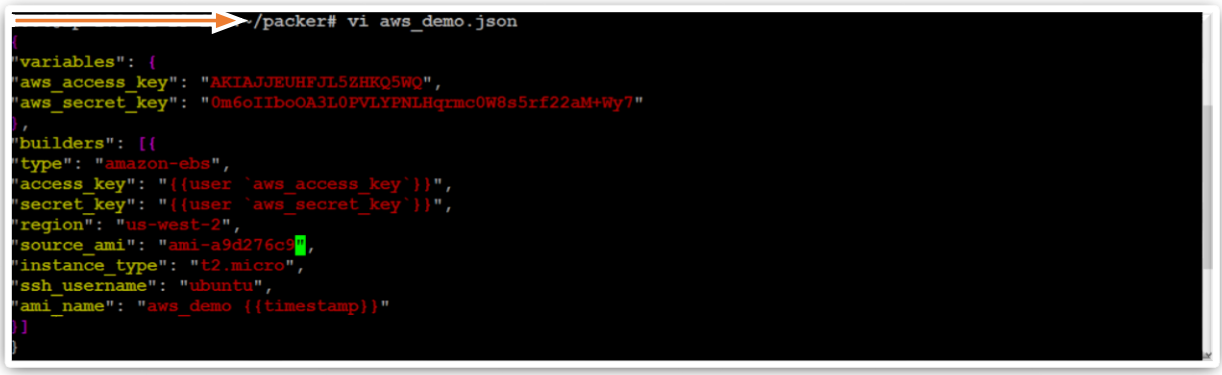
Step 3: Creating customizable Image using **Packer**.

Create a json file and paste the below script.

- vi aws_demo.json

```
{
"variables": {
"aws_access_key": "AKIAJJEUHFJL5ZHKQ5WQ",
"aws_secret_key": "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"
},
"builders": [{
"type": "amazon-ebs",
"access_key": "{{user `aws_access_key`}}",
"secret_key": "{{user `aws_secret_key`}}",
"region": "us-west-2",
"source_ami": "ami-a9d276c9",
"instance_type": "t2.micro",
"ssh_username": "ubuntu",
"ami_name": "aws_demo {{timestamp}}"
}]
}
```

- Here, we have to provide our own AWS access key and secret key.
- Region will be the aws-region where you want to create the image.
- Source-ami is the default machine image from where we are replicating our customizable image.
- Instance-type is the type of machine with different configuration of ram and cpu.
- Ssh-username is the machine name through which we can ssh it.
- Ami-name is the name which you want to give to ua image.

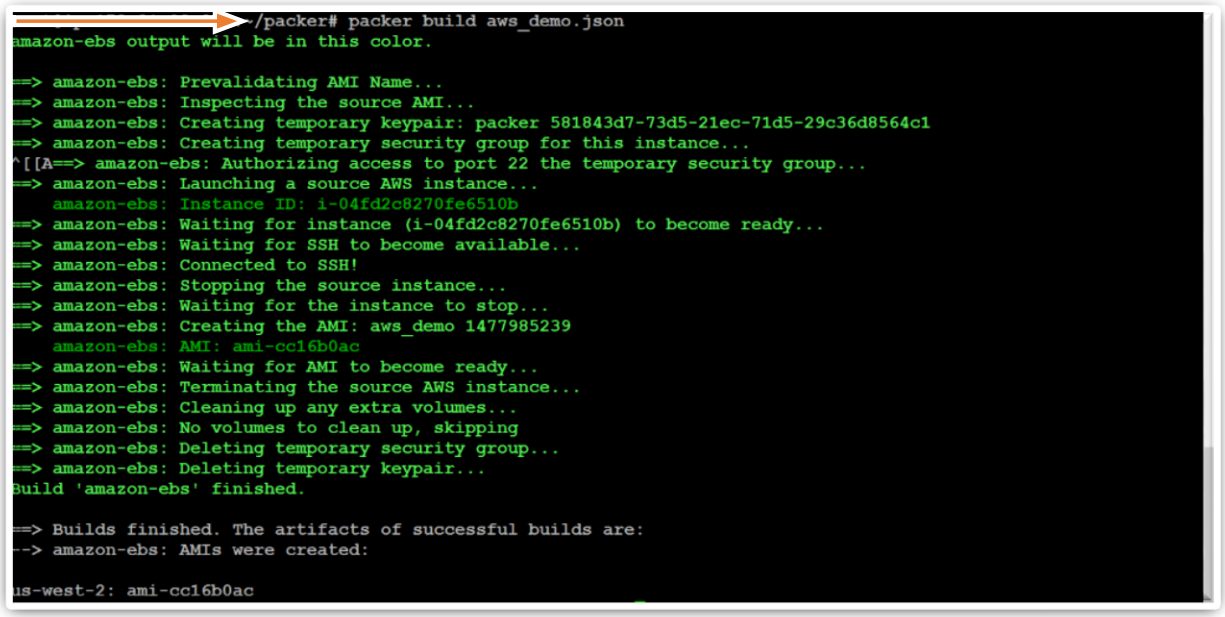


Validate the json file.

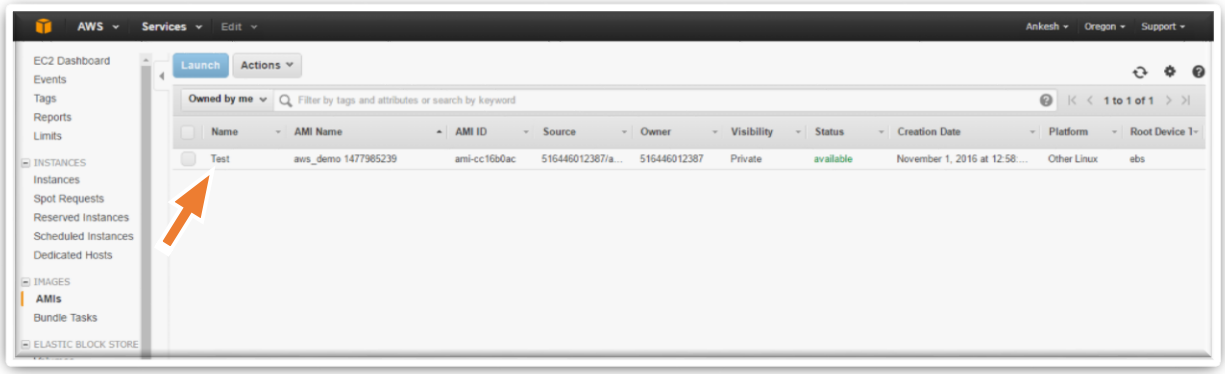
- packer validate aws_demo.json

Build Packer Image.

- packer build aws_demo.json



After the image is successfully created, we can see the same image in our AWS AMIs section (owned by me).



Now we can see the **AMI ID.** We will use this image ID in Vagrant and Terraform.

Step 4: Installing and configuring **Terraform**.

Download the Terraform installation zip file.

- wget https://releases.hashicorp.com/terraform/0.6.6/terraform_0.6.6_linux_amd64.zip

```
# git --version
git version 2.7.4
# wget https://releases.hashicorp.com/terraform/0.6.6/terraform_0.6.6_linux_amd64.zip
--2016-11-01 08:40:34--  https://releases.hashicorp.com/terraform/0.6.6/terraform_0.6.6_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 151.101.52.69
Connecting to releases.hashicorp.com (releases.hashicorp.com)|151.101.52.69|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 88109309 (84M) [application/zip]
Saving to: âterraform_0.6.6_linux_amd64.zipâ

terraform_0.6.6_linux_amd64. 100%[=========================================>]  84.03M  29.4MB/s    in 2.9s

2016-11-01 08:40:37 (29.4 MB/s) - âterraform_0.6.6_linux_amd64.zipâ saved [88109309/88109309]
```

Unzip the package:

- unzip terraform_0.6.6_linux_amd64.zip -d terraform

```
# unzip terraform_0.6.6_linux_amd64.zip -d terraform
Archive:  terraform_0.6.6_linux_amd64.zip
  inflating: terraform/terraform
  inflating: terraform/terraform-provider-atlas
  inflating: terraform/terraform-provider-aws
  inflating: terraform/terraform-provider-azure
  inflating: terraform/terraform-provider-cloudflare
  inflating: terraform/terraform-provider-cloudstack
  inflating: terraform/terraform-provider-consul
  inflating: terraform/terraform-provider-digitalocean
```

Set the terraform path in .bashrc.

- vi .bashrc
  - `PATH=$PATH:/root/terraform`

```
# vi .bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples
PATH=$PATH:/root/terraform
export PATH=$PATH:/root/packer
# If not running interactively, don't do anything
[ -z "$PS1" ] && return
```

Logout and login as sudo user. Then, check whether terraform is installed or not.

- exit
- sudo su -
- terraform

```
# exit
logout
:~$ sudo su -
# terraform
Usage: terraform [--version] [--help] <command> [<args>]

Available commands are:
    apply      Builds or changes infrastructure
    destroy    Destroy Terraform-managed infrastructure
    get        Download and install modules for the configuration
    graph      Create a visual graph of Terraform resources
    init       Initializes Terraform configuration from a module
    output     Read an output from a state file
    plan       Generate and show an execution plan
    push       Upload this Terraform module to Atlas to run
    refresh    Update local state file against real resources
    remote     Configure remote state storage
    show       Inspect Terraform state or plan
    taint      Manually mark a resource for recreation
    version    Prints the Terraform version
```

Step 5: Use Packer image and deploy/destroy servers in AWS using Terraform.

Create a terraform project directory.

- mkdir aws_demo
- cd aws_demo

```
# mkdir aws_demo
# cd aws_demo/
```

Create a tf file and paste the below script.

- vi aws_demo.tf

```
provider "aws" {
  access_key = "AKIAJJEUHFJL5ZHKQ5WQ"
  secret_key = "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"
  region     = "us-west-2"
}

resource "aws_instance" "aws_demo" {
  ami           = "ami-cc16b0ac"
  instance_type = "t2.micro"
}
```

```
/aws_demo# vi aws_demo.tf
provider "aws" {
  access_key = "AKIAJJEUHFJL5ZHKQ5WQ"
  secret_key = "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"
  region     = "us-west-2"
}

resource "aws_instance" "aws_demo" {
  ami           = "ami-cc16b0ac"
  instance_type = "t2.micro"
```

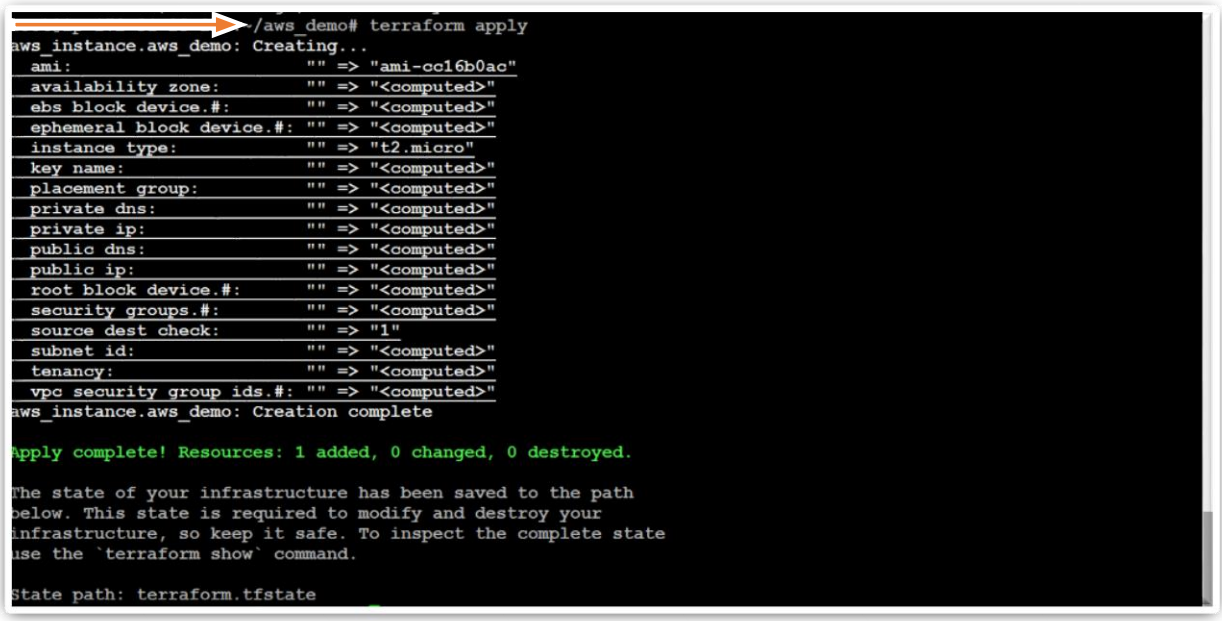
Check the plan before deploying servers in AWS.

- terraform plan

```
/aws_demo# terraform plan
Refreshing Terraform state prior to plan...


The Terraform execution plan has been generated and is shown below.
Resources are shown in alphabetical order for quick scanning. Green resources
will be created (or destroyed and then created if an existing resource
exists), yellow resources are being changed in-place, and red resources
will be destroyed.

Note: You didn't specify an "-out" parameter to save this plan, so when
"apply" is called, Terraform can't guarantee this is what will execute.

+ aws_instance.aws_demo
    ami:                      "" => "ami-cc16b0ac"
    availability_zone:        "" => "<computed>"
    ebs_block_device.#:       "" => "<computed>"
    ephemeral_block_device.#: "" => "<computed>"
    instance_type:            "" => "t2.micro"
    key_name:                 "" => "<computed>"
    placement_group:          "" => "<computed>"
    private_dns:              "" => "<computed>"
    private_ip:               "" => "<computed>"
    public_dns:               "" => "<computed>"
    public_ip:                "" => "<computed>"
    root_block_device.#:      "" => "<computed>"
    security_groups.#:        "" => "<computed>"
    source_dest_check:        "" => "1"
    subnet_id:                "" => "<computed>"
    tenancy:                  "" => "<computed>"
    vpc_security_group_ids.#: "" => "<computed>"
```

If the plan is correct, apply it.

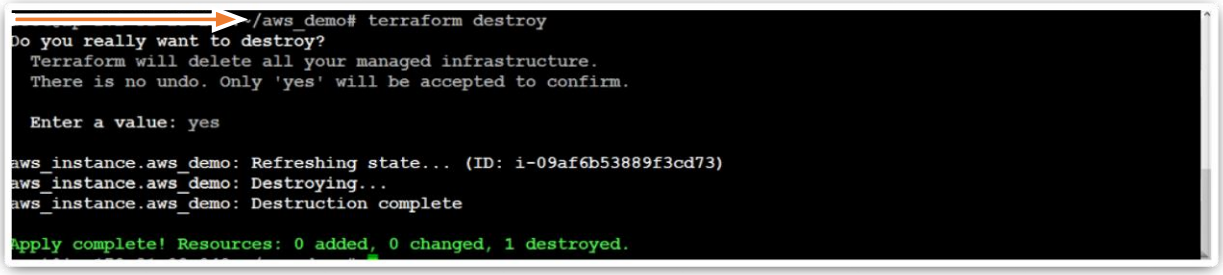
- terraform apply



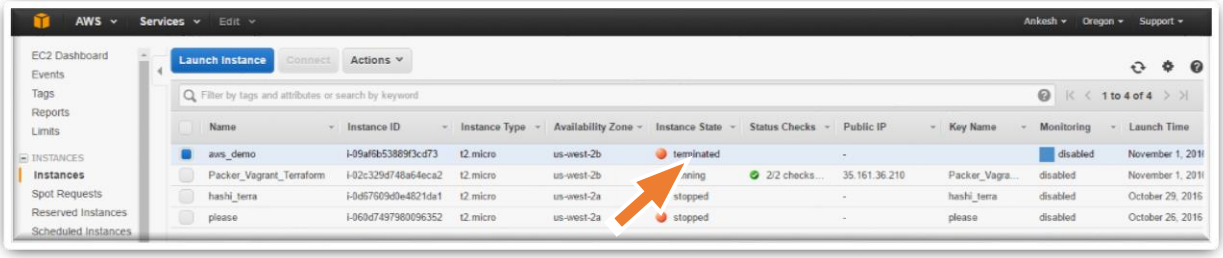
After the successful deployment of server, we can see a new aws instance is launched in our AWS.

If we want to delete the server which we have deployed.

- terraform destroy



After deleting, the changes are reflected.



Step 6: Using **Vagrant** to create/destroy development environment using the Packer image.

First of all we have to install vagrant-aws plugin.

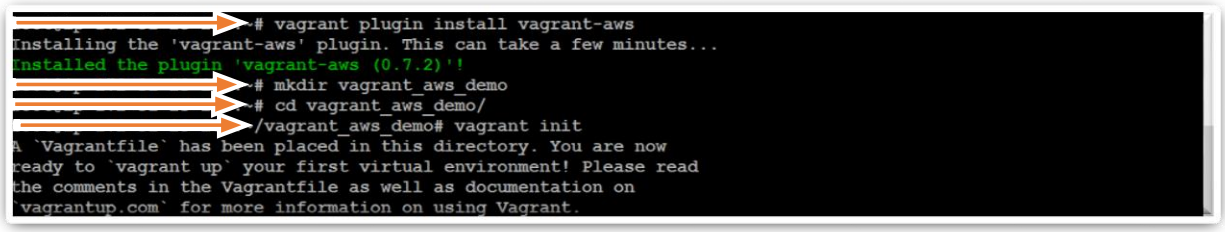
- vagrant plugin install vagrant-aws

Creating a Vagrant project directory.

- mkdir vagrant_aws_demo
- cd vagrant_aws_demo

Initialize Vagrant inside the project directory.

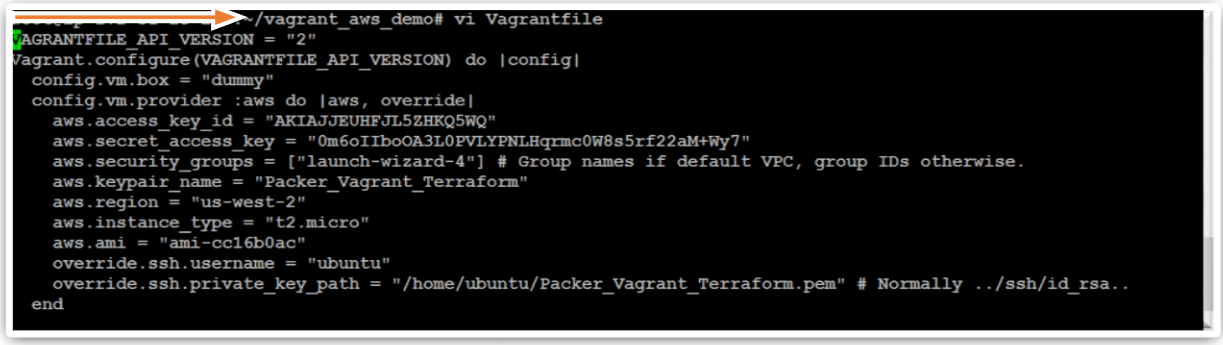
- vagrant init



Edit the Vagrant file and paste this script in it.

- vi Vagrantfile

```
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "dummy"
config.vm.provider :aws do |aws, override|
    aws.access_key_id = "AKIAJJEUHFJL5ZHKQ5WQ"
    aws.secret_access_key =
"0m6oIIboOA3L0PVLYPNLHqrmc0W8s5rf22aM+Wy7"
    aws.security_groups = ["launch-wizard-4"]
    aws.keypair_name = "Packer_Vagrant_Terraform"
    aws.region = "us-west-2"
    aws.instance_type = "t2.micro"
    aws.ami = "ami-cc16b0ac"
    override.ssh.username = "ubuntu"
    override.ssh.private_key_path =
"/home/ubuntu/Packer_Vagrant_Terraform.pem"
  end
end
```



We will then add a dummy box.

- vagrant box add dummy https://github.com/mitchellh/vagrant-aws/raw/master/dummy.box

Start Vagrant and it will create the environment.

- vagrant up --provider=aws

```
/vagrant_aws_demo# vagrant up --provider=aws
Bringing machine 'default' up with 'aws' provider...
==> default: Warning! The AWS provider doesn't support any of the Vagrant
==> default: high-level network configurations (`config.vm.network`). They
==> default: will be silently ignored.
==> default: Launching an instance with the following settings...
==> default:  -- Type: t2.micro
==> default:  -- AMI: ami-cc16b0ac
==> default:  -- Region: us-west-2
==> default:  -- Keypair: Packer_Vagrant_Terraform
==> default:  -- Security Groups: ["launch-wizard-4"]
==> default:  -- Block Device Mapping: []
==> default:  -- Terminate On Shutdown: false
==> default:  -- Monitoring: false
==> default:  -- EBS optimized: false
==> default:  -- Source Destination check:
==> default:  -- Assigning a public IP address in a VPC: false
==> default:  -- VPC tenancy specification: default
==> default: Waiting for instance to become "ready"...
==> default: Waiting for SSH to become available...
==> default: Machine is booted and ready for use!
==> default: Rsyncing folder: /root/vagrant_aws_demo/ => /vagrant
```

To use the environment.

- vagrant ssh

```
/vagrant_aws_demo# vagrant ssh
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-45-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
0 updates are security updates.
```

If we want to delete the environment we created.

- vagrant destroy

```
/vagrant_aws_demo# vagrant destroy
    default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Terminating the instance...
```

Demonstration:-

Now we have completed these three processes:

I. Creating image using Packer.
II. Deploying virtual machines using Terraform.
III. Creating Development environment using Vagrant.

To show a demo on Vagrant automated Web Server, we have to follow these steps:

In Host machine:

1. Create a virtual machine.
2. Installing pip.
3. Installing awscli.
4. Configuring awscli.
5. Creating key pair.
6. Installing virtualbox.
7. Installing vagrant.
8. Installing plugins.

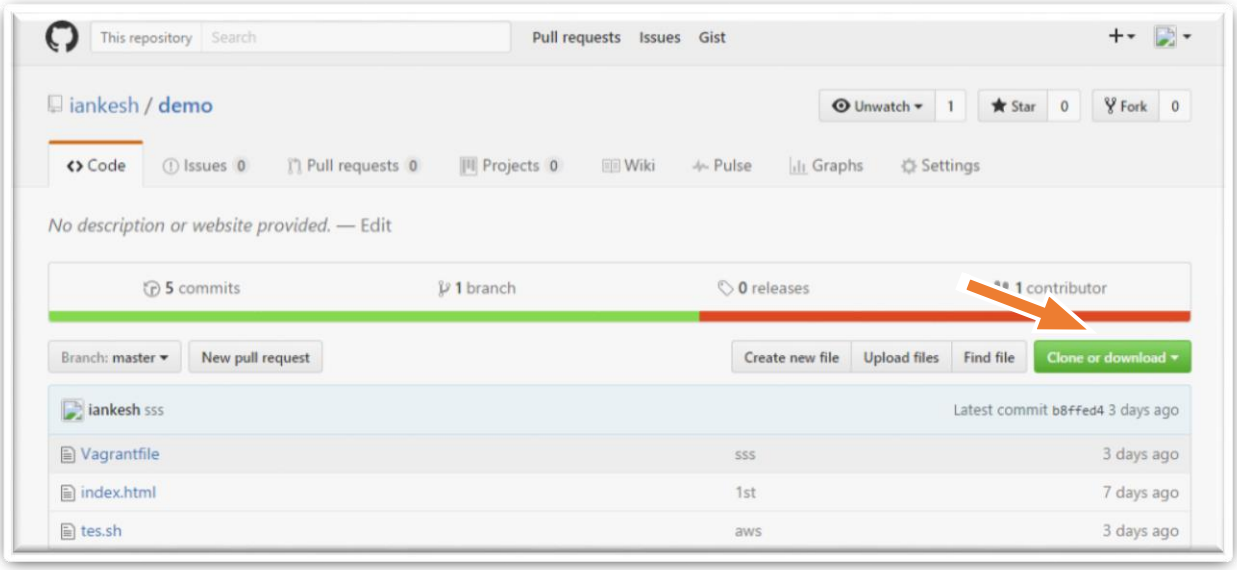
In Guest machine:

1. vagrant ssh.
2. Installing apache.
3. Linking host machine to guest machine.
4. Source code will be in Host machine.
5. Reflecting my web page in guest machine which is in my GitHub repo.

We are going to automate these 13 tasks by using one script ->> “tes.sh”.

Steps to be followed:

Clone the repo from GitHub.



- git clone https://github.com/iankesh/demo.git

```
# git clone https://github.com/iankesh/demo.git
Cloning into 'demo'...
remote: Counting objects: 14, done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 14 (delta 5), reused 9 (delta 1), pack-reused 0
Unpacking objects: 100% (14/14), done.
Checking connectivity... done.
```

Let's go inside the directory. We can see that these three files are copied inside a directory.

```
# ls
demo
# cd demo/
root@ip-172-31-46-26:~/demo# ls
index.html  tes.sh  Vagrantfile
```

Run the script -> tes.sh

- ./tes.sh

It will prompt to put Access Key and Secret Key of your AWS account.

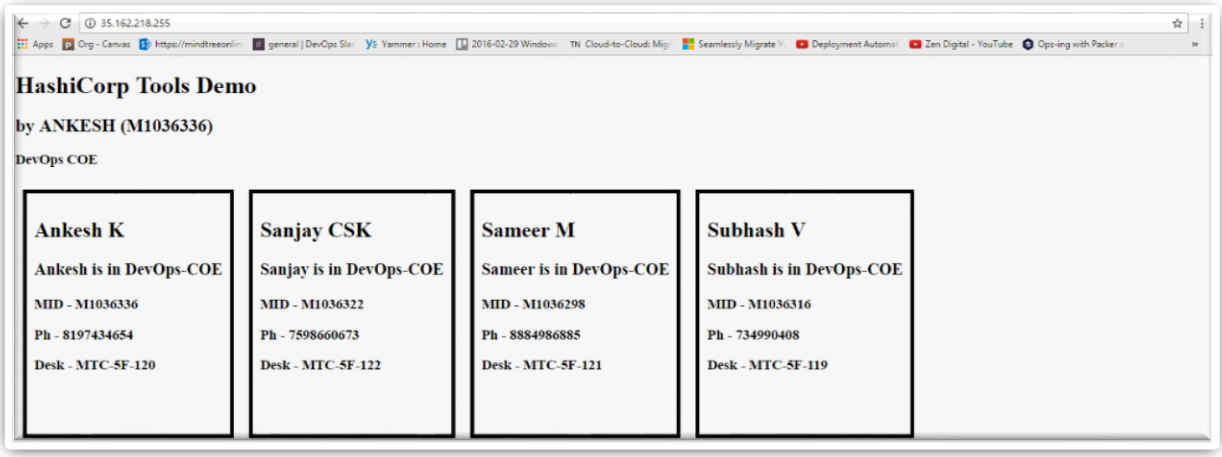
```
configuring aws
AWS Access Key ID [None]: AKIAJFO6D4OIN5KJJNKA
AWS Secret Access Key [None]: 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
Default region name [None]: us-west-2
Default output format [None]:
```

We can start vagrant by using this command.

- vagrant up –provider=aws

```
:~/demo# vagrant up --provider=aws
Bringing machine 'default' up with 'aws' provider...
==> default: Warning! The AWS provider doesn't support any of the Vagrant
==> default: high-level network configurations (`config.vm.network`). They
==> default: will be silently ignored.
==> default: Launching an instance with the following settings...
==> default:  -- Type: t2.micro
==> default:  -- AMI: ami-da07a6ba
==> default:  -- Region: us-west-2
==> default:  -- Keypair: MyKeyPair
==> default:  -- Security Groups: ["launch-wizard-4"]
==> default:  -- Block Device Mapping: []
==> default:  -- Terminate On Shutdown: false
==> default:  -- Monitoring: false
==> default:  -- EBS optimized: false
==> default:  -- Source Destination check:
==> default:  -- Assigning a public IP address in a VPC: false
==> default:  -- VPC tenancy specification: default
==> default: Waiting for instance to become "ready"...
==> default: Waiting for SSH to become available...
==> default: Machine is booted and ready for use!
==> default: Rsyncing folder: /root/demo/ => /vagrant
```

It will create a development environment. And it is up and running with my webpage.



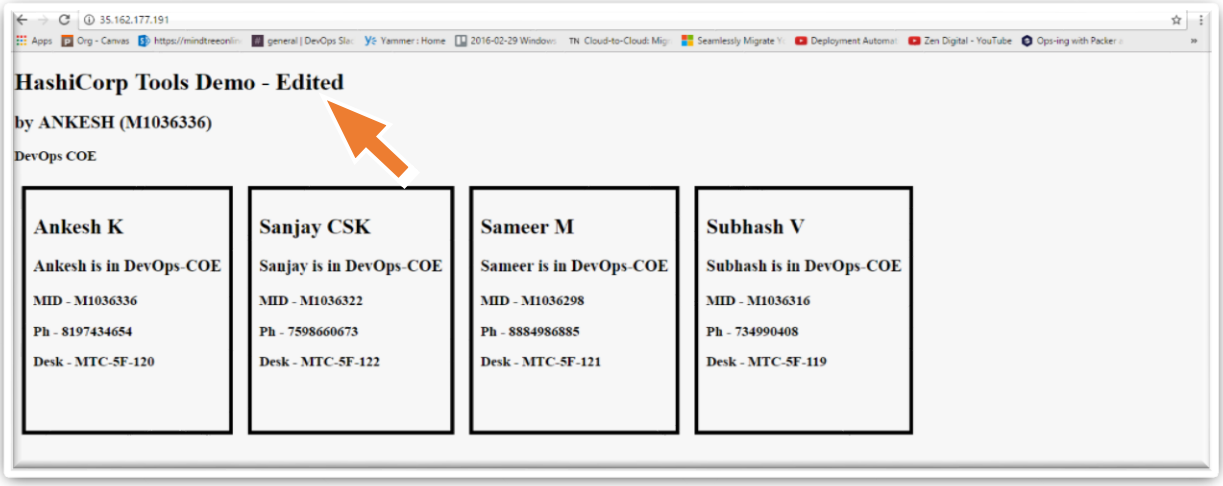
Let's make changes in the index.html file from the host machine. The host machine is linked to the guest machine. So the changes will be reflected once we give this command.

- vagrant reload

```
:~/demo# vi index.html
<!DOCTYPE html>
<html lang="en-us">
<head>
<style>
.city {
  float: left;
  margin: 10px;
  padding: 10px;
  max-width: 300px;
  height: 300px;
  border: 5px solid black;
}
</style>
</head>
<body>
<h1>HashiCorp Tools Demo - Edited</h1>
<h2>by  ANKESH (M1036336)</h2>
<h3>DevOps COE<h3>
<div class="city">
  <h2>Ankesh K</h2>
```

```
:~/demo# vagrant reload
==> default: Stopping the instance...
==> default: Warning! The AWS provider doesn't support any of the Vagrant
==> default: high-level network configurations (`config.vm.network`). They
==> default: will be silently ignored.
==> default: Starting the instance...
==> default: Waiting for instance to become "ready"...
==> default: Waiting for SSH to become available...
==> default: Machine is booted and ready for use!
==> default: Rsyncing folder: /root/demo/ => /vagrant
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: flag to force provisioning. Provisioners marked to run always will still run.
```

We can see the changes reflected in our web page.



Now, we can destroy our environment using this command.

- vagrant reload

```
:~/demo# vagrant destroy
    default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Terminating the instance...
```

Conclusions

Run a single command — "vagrant up" — and sit back as Vagrant puts together your complete development environment. Say goodbye to the "works on my machine" excuse as Vagrant creates identical development environments for everyone on your team.

Packer is easy to use and automates the creation of any type of machine image. It embraces modern configuration management by encouraging you to use automated scripts to install and configure the software within your Packer-made images. Packer brings machine images into the modern age, unlocking untapped potential and opening new opportunities.

Terraform enables you to safely and predictably create, change, and improve production infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned.

For more information about HashiCorp tools, visit: https://www.hashicorp.com/

Provision Secure and Run Any Infrastructure for Any Application