

WHITEPAPER

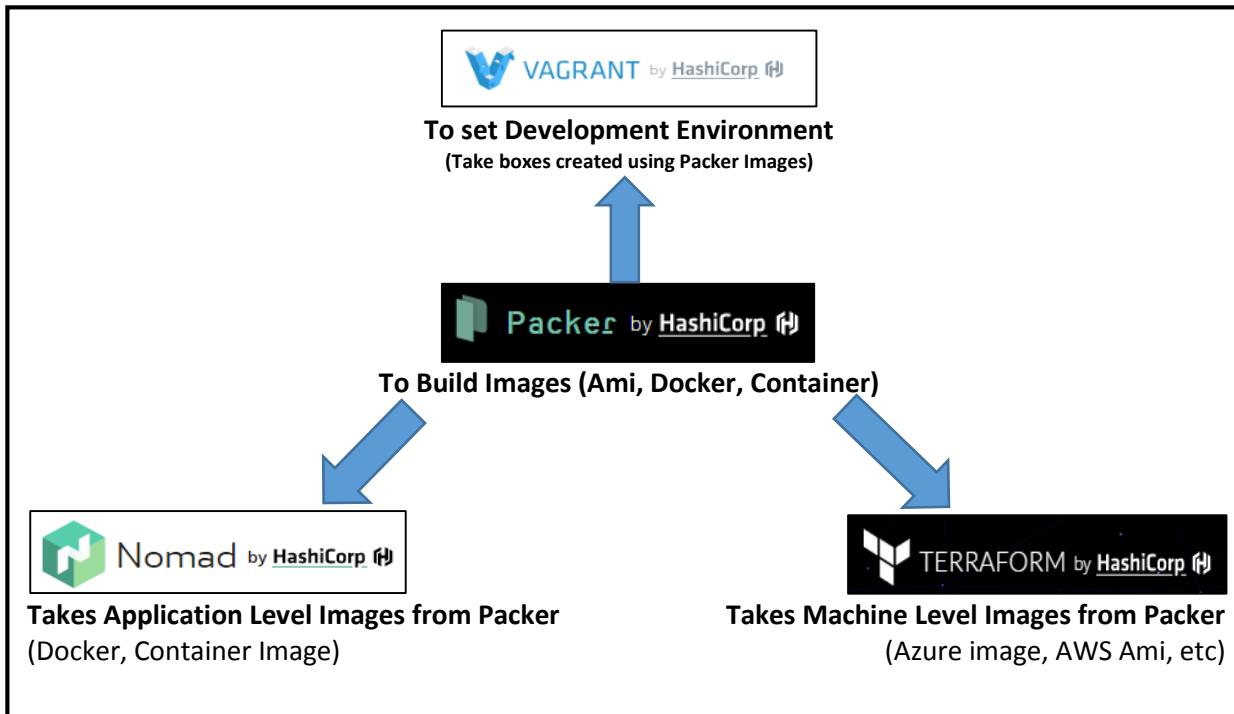


Vagrant Automated Web Server
creation Using Packer and provision
using Terraform.

Provision Secure and Run Any Infrastructure for Any Application

Contents

Introduction.....	3
Basics	
Requirement	
Pre-Requisite	
Steps	
Installing and Configuring Virtualbox & Vagrant.....	4
Installing Virtualbox	
Installing Vagrant	
Configuring both	
Installing and Configuring Packer.....	5
Installing Packer	
Setting Packer path	
Creating customizable images using Packer.....	6
Write a .json script	
Build image using packer command	
Installing and Configuring Terraform.....	8
Installing Terraform	
Setting Terraform path	
Use Packer image and deploy/destroy servers in AWS using Terraform.....	9
Write a .tf file	
Use terraform commands to deploy/plan/destroy servers	
Using Vagrant to create/destroy environment using the Packer image.....	11
Install plugins	
Initialize Vagrant and edit Vagrantfile	
Use vagrant commands to create/destroy environment	
Demonstration.....	13
Steps to be followed	
Run script and Vagrant up	
Conclusions.....	16



- **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

```
~$ sudo su -
# clear
# apt-get update && apt-get upgrade
Hit:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial-updates InRelease [95.7 kB]
Get:3 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial-backports InRelease [92.2 kB]
Get:4 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial/main Sources [868 kB]
Get:5 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial/restricted Sources [4,808 B]
Get:6 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial/universe Sources [7,728 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security InRelease [94.5 kB]
Get:8 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial/multiverse Sources [179 kB]
Get:9 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial-updates/main Sources [201 kB]
```

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

```
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

```
# sudo apt-get update && sudo apt-get install dkms virtualbox virtualbox-guest-additions-iso
so
Hit:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:3 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial-backports InRelease
Hit:4 http://download.virtualbox.org/virtualbox/debian wily InRelease
Hit:5 http://download.virtualbox.org/virtualbox/debian xenial InRelease
```

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

```
# vboxmanage --version
5.0.24_Ubuntur108355
# wget https://releases.hashicorp.com/vagrant/1.8.6/vagrant_1.8.6_x86_64.deb
--2016-11-01 06:51:55-- https://releases.hashicorp.com/vagrant/1.8.6/vagrant_1.8.6_x86_64.deb
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: 76314834 (73M) [application/x-debian-package]
Saving to: ávagrant_1.8.6_x86_64.debá

vagrant_1.8.6_x86_64.deb      100%[=====]  72.78M  28.5MB/s   in 2.6s
2016-11-01 06:51:58 (28.5 MB/s) - ávagrant_1.8.6_x86_64.debá saved [76314834/76314834]
```

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

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

```
# dpkg -i vagrant_1.8.6_x86_64.deb
Selecting previously unselected package Vagrant.
(Reading database ... 77617 files and directories currently installed.)
Preparing to unpack vagrant_1.8.6_x86_64.deb ...
Unpacking vagrant (1:1.8.6) ...
Setting up vagrant (1:1.8.6) ...
# vagrant -v
Vagrant 1.8.6
```

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

```
# mkdir packer
# cd packer/
/packer# wget https://releases.hashicorp.com/packer/0.9.0/packer_0.9.0_linux_386.zip
--2016-11-01 06:58:28-- https://releases.hashicorp.com/packer/0.9.0/packer_0.9.0_linux_386.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: 7704302 (7.3M) [application/zip]
Saving to: ápacker_0.9.0_linux_386.zipâ

packer_0.9.0_linux_386.zip 100%[=====] 7.35M 29.7MB/s in 0.2s

2016-11-01 06:58:29 (29.7 MB/s) - ápacker_0.9.0_linux_386.zipâ saved [7704302/7704302]
```

Unzip the Packer file.

- ✚ unzip packer_0.9.0_linux_386.zip

```
/packer# unzip packer_0.9.0_linux_386.zip
Archive:  packer_0.9.0_linux_386.zip
  inflating: packer
```

Set the packer path in .bashrc

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

```
# 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
export PATH=$PATH:/root/packer
# If not running interactively, don't do anything
[ -z "$PS1" ] && return

# don't put duplicate lines in the history. See bash(1) for more options
```

Logout of the system and again login using sudo user.

- ✚ exit
- ✚ sudo su -

Check whether Packer is installed or not.

- ✚ packer

```
# exit
logout
d:~$ sudo su -
# packer
usage: packer [--version] [--help] <command> [<args>]

Available commands are:
  build      build image(s) from template
  fix        fixes templates from old versions of packer
  inspect    see components of a template
  push       push a template and supporting files to a Packer build service
  validate   check that a template is valid
  version    Prints the Packer version
```

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.

```
[root@ip-172-31-10-14 ~]# 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}}"
    }
  ]
}
```

Validate the json file.

packer validate aws_demo.json

```
[root@ip-172-31-10-14 ~]# packer validate aws_demo.json
Template validated successfully.
```

Build Packer Image.

```
+ packer build aws_demo.json
```

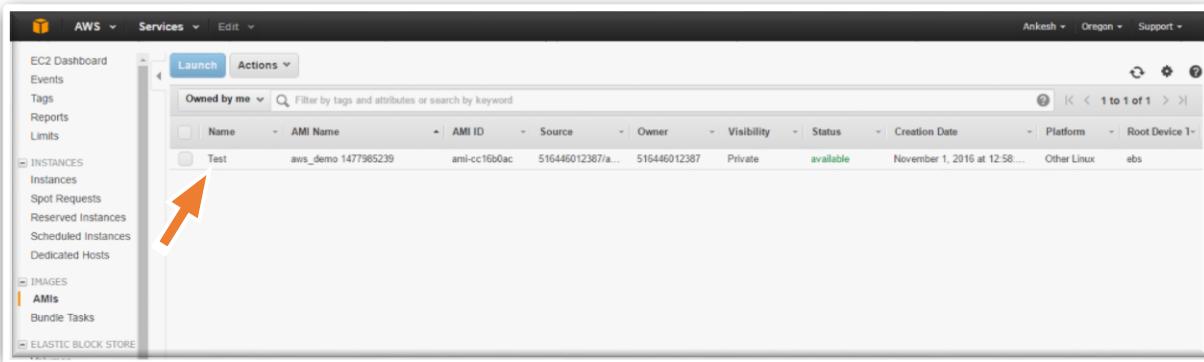
```
→ /packer# packer build aws_demo.json
amazon-ebs output will be in this color.

==> amazon-ebs: Prevalidating AMI Name...
==> amazon-ebs: Inspecting the source AMI...
==> amazon-ebs: Creating temporary keypair: packer 581843d7-73d5-21ec-71d5-29c36d8564c1
==> amazon-ebs: Creating temporary security group for this instance...
^[[A==> amazon-ebs: Authorizing access to port 22 the temporary security group...
==> amazon-ebs: Launching a source AWS instance...
amazon-ebs: Instance ID: i-04fd2c8270fe6510b
==> amazon-ebs: Waiting for instance (i-04fd2c8270fe6510b) to become ready...
==> amazon-ebs: Waiting for SSH to become available...
==> amazon-ebs: Connected to SSH!
==> amazon-ebs: Stopping the source instance...
==> amazon-ebs: Waiting for the instance to stop...
==> amazon-ebs: Creating the AMI: aws_demo 1477985239
amazon-ebs: AMI: ami-cc16b0ac
==> amazon-ebs: Waiting for AMI to become ready...
==> amazon-ebs: Terminating the source AWS instance...
==> amazon-ebs: Cleaning up any extra volumes...
==> amazon-ebs: No volumes to clean up, skipping
==> amazon-ebs: Deleting temporary security group...
==> amazon-ebs: Deleting temporary keypair...
Build 'amazon-ebs' finished.

==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs: AMIs were created:

us-west-2: ami-cc16b0ac
```

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: áterrafirm_0.6.6_linux_amd64.zip

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) - áterrafirm_0.6.6_linux_amd64.zip 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
  inflating: terraform/terraform-provider-dns
```

Set the terraform path in .bashrc.

vi .bashrc

➤ PATH=\$PATH:/root/terraform

```
# vi .bashrc
#!/bin/bash
# .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

# Don't set up additional lines in the history, on behalf of non-interactive
```

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"
> }
```

```
└──╼ # 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

```
└──╼ # 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>"
```

## Vagrant Automated Web Server Creation Using Packer and Provision Using Terraform

If the plan is correct, apply it.

✚ **terraform apply**

```
/aws_demo# terraform apply
aws_instance.aws_demo: Creating...
 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>"
aws_instance.aws_demo: Creation complete

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.

State path: terraform.tfstate
```

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

| Name                     | Instance ID         | Instance Type | Availability Zone | Instance State | Status Checks | Public IP     | Key Name       | Monitoring | Launch Time      |
|--------------------------|---------------------|---------------|-------------------|----------------|---------------|---------------|----------------|------------|------------------|
| aws_demo                 | i-09af6b53889f3cd73 | t2.micro      | us-west-2b        | running        | Initializing  | 35.161.202.42 | Packer_Vagr... | disabled   | November 1, 2016 |
| Packer_Vagrant_Terraform | i-02c329d74ba64eca2 | t2.micro      | us-west-2b        | running        | 2/2 checks... | 35.161.36.210 | Packer_Vagr... | disabled   | November 1, 2016 |
| hashi_terra              | i-0d57609d0e4821da1 | t2.micro      | us-west-2a        | stopped        | -             | -             | hashi_terra    | disabled   | October 29, 2016 |
| please                   | i-060d7497980096352 | t2.micro      | us-west-2a        | stopped        | -             | -             | please         | disabled   | October 26, 2016 |

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

✚ **terraform destroy**

```
/aws_demo# terraform destroy
Do you really want to destroy?
Terraform will delete all your managed infrastructure.
There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

aws_instance.aws_demo: Refreshing state... (ID: i-09af6b53889f3cd73)
aws_instance.aws_demo: Destroying...
aws_instance.aws_demo: Destruction complete

Apply complete! Resources: 0 added, 0 changed, 1 destroyed.
```

After deleting, the changes are reflected.

| Name                     | Instance ID         | Instance Type | Availability Zone | Instance State | Status Checks | Public IP     | Key Name       | Monitoring | Launch Time      |
|--------------------------|---------------------|---------------|-------------------|----------------|---------------|---------------|----------------|------------|------------------|
| aws_demo                 | i-09af6b53889f3cd73 | t2.micro      | us-west-2b        | terminated     | -             | -             | Packer_Vagr... | disabled   | November 1, 2016 |
| Packer_Vagrant_Terraform | i-02c329d74ba64eca2 | t2.micro      | us-west-2b        | running        | 2/2 checks... | 35.161.36.210 | Packer_Vagr... | disabled   | November 1, 2016 |
| hashi_terra              | i-0d57609d0e4821da1 | t2.micro      | us-west-2a        | stopped        | -             | -             | hashi_terra    | disabled   | October 29, 2016 |
| please                   | i-060d7497980096352 | t2.micro      | us-west-2a        | stopped        | -             | -             | please         | disabled   | October 26, 2016 |

## 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**

```
vagrant plugin install vagrant-aws
Installing the 'vagrant-aws' plugin. This can take a few minutes...
Installed the plugin 'vagrant-aws (0.7.2)'

mkdir vagrant_aws_demo
cd vagrant_aws_demo/
/vagrant_aws_demo# vagrant init
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

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
```

```
~/vagrant_aws_demo# 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"] # Group names if default VPC, group IDs otherwise.
 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" # Normally ../ssh/id_rsa..
end
```

## Vagrant Automated Web Server Creation Using Packer and Provision Using Terraform

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.

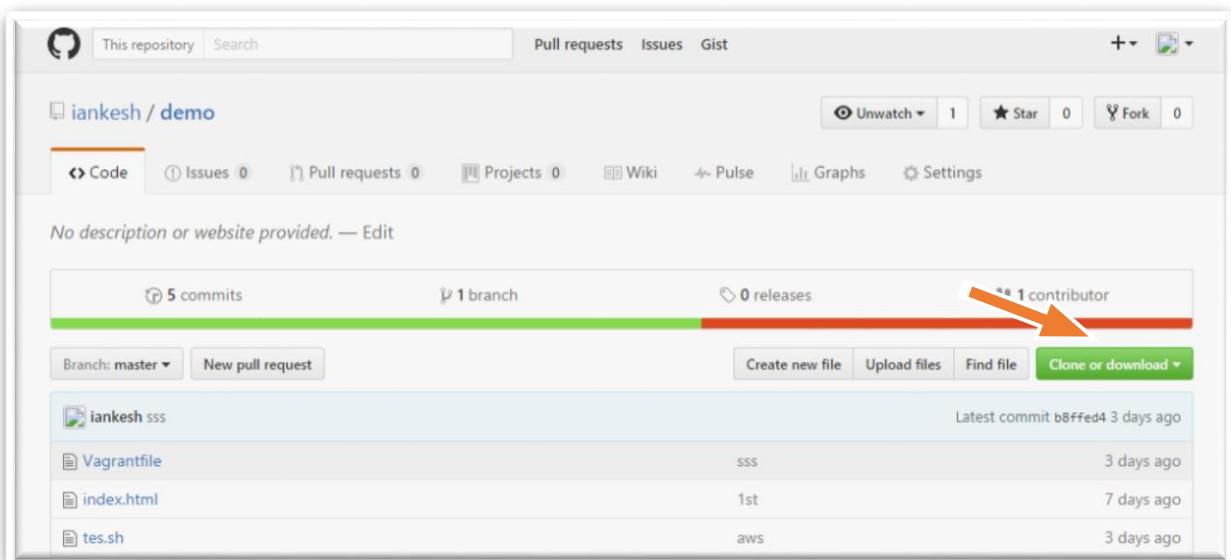
In Guest machine:

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.



## Vagrant Automated Web Server Creation Using Packer and Provision Using Terraform

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]: AKIAJF06D4OIN5KJJNKA
AWS Secret Access Key [None]: Na/hw0keYWyZwQHWgPcwRocSrPgOL4VonPawwGG
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.

The screenshot shows a web browser window with the following details:

- HashiCorp Tools Demo**
- by ANKESH (M1036336)
- DevOps COE
- Ankesh K**  
Ankesh is in DevOps-COE  
MID - M1036336  
Ph - 8197434654  
Desk - MTC-5F-120
- Sanjay CSK**  
Sanjay is in DevOps-COE  
MID - M1036322  
Ph - 7598660673  
Desk - MTC-5F-122
- Sameer M**  
Sameer is in DevOps-COE  
MID - M1036298  
Ph - 8884986885  
Desk - MTC-5F-121
- Subhash V**  
Subhash is in DevOps-COE  
MID - M1036316  
Ph - 734990408  
Desk - MTC-5F-119

## Vagrant Automated Web Server Creation Using Packer and Provision Using Terraform

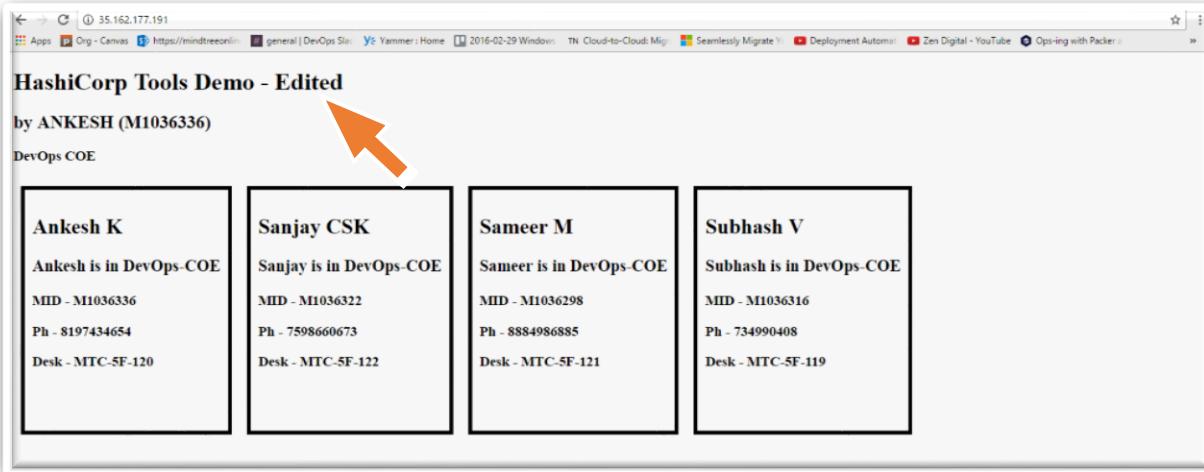
Let's make changes in the index.html file from the host machine. The host machine is linked to the guest machine. So the changes will be reflected once we give this command.

 **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