***What is Hashicorp Packer***

* It builds machine images
* An AMI is a resource that stores all the configurations, permissions and metadata needed to create a virtual machine
* It can be called a golden image
* Packer isn't limited to Linux, you can provision a machine image using windows or other OS as well
* The machine images will serve as roadmaps for building other virtual images

***Why use Packer***

* Create images via a template
* You can re-run the template without making any changes
* Automate everything. It is usually tied with CICD pipelines
* Create identical images. Cross cloud, cross environment, cross platform a single packer template can create machine images for multiple use cases
* Provisions code we can use and re-use

***Packer breakdown***

1. Builder:

* Tells packer what platform, region and source image
* Contains AMI name too
* Only mandatory field

1. Provisioner

* This defines how to configure the image, most likely by using your existing configuration management platform. No need to rewrite, just reuse
* Let's us leverage what we already have

1. Post-processors

* Related to the builder, runs after the image is built. Generally generates or supplies artifacts
* What it does depends on your platform

1. Communicator

* How packer works on the machine image during creation. Y default this is SSH and does not need to be defined
* This is how packer connects to the instance and it always assumes to be ssh
* If you are using windows you have to provide the winRM communicator.
* You might not notice the communicator

* A build in packer is a combination of the builder, the provisioner and the post processors that you provide. A template can have multiple builds
* An artifact is the end result of a build; generally this is the machine image itself and any generated log of metadata files
* Command is what we need to run to manage packer. Most often this is the packer build command

***JSON***

* How to structure your data in JSON
* It is compromised of the object aka the name/value pair. Comprised of the opening and closing curly brackets. The names are in name/value pairs and end in a colon. When you have multiple objects they always end in a comma except for the last object {}
* Array aka ordered list, vector, list sequence
* The array starts with a left bracket and ends with a right bracket []
* Separate multiple pairs with a comma
* Whitespace doesn’t really matter is JSON but 2 or 4 spaces are commonly used

Hands-on in json

* To move all the lines from the first and last curly bracket, select all the lines from the second line all the way to the second to last line and do a shift plus HTML angle brackets (same as the > or the period button on the keyboard)
* To highlight everything in the vi editor select shift + v and then select until the second to the last line then shift + > to shift everything to the right
* Highlight all the objects in the builders until the second to the last curly bracket
* There should always be some space between different objects
* To validate if the template is correct always run packer validate plus name of template

{

2 "variables": {

3 "aws\_access\_key": "AKIATW3SA5KON4SJLSQD",

4 "aws\_secret\_key": "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",

5 "aws\_subnet\_id": "subnet-0b9135191840a57d3"

6 },

7 "builders": [

8 {

9 "type": "amazon-ebs",

10 "access\_key": "{{user `aws\_access\_key`}}",

11 "secret\_key": "{{user `aws\_secret\_key`}}",

12 "region": "us-east-1",

13 "subnet\_id": "{{user `aws\_subnet\_id`}}",

14 "source\_ami\_filter": {

15 "filters": {

16 "virtualization-type": "hvm",

17 "name": "ubuntu/images/\*ubuntu-focal-20.04-amd64-server- \*",

18 "root-device-type": "ebs"

19 },

20 "owners": ["099720109477"],

21 "most\_recent": true

22 },

"ami-test.json" 39L, 1288C 1

***HCL2***

* This is Hashicorp configuration language
* This is version 2 of the configuration language
* It is human readable than json. So human first and it is also json compatible
* You have the ability to add comments as well
* It is also comprised of objects and arrays. Here it looks like this name ="value" and arrays are basically the same opening and closing with straight brackets as such [ "item", "item"]
* In json you start with a curly brace and end with a curly brace. No more commas and colon switches to equal sign
* HCL includes attributes as well. Example is the ami-name which is a single configuration unit, generally a name/value pair
* Attributes can be contained in a block. A block is a collection of attributes with an annotated block type (filters)
* A body is a collection of associated blocks. This can be the entire code as it can be used to create the AMI
* Tabs or spaces still doesn’t matter with HCL

Packer Installation:

*Windows*:

* 2 options: Either through packer.io which provides you with a packer binary. You can do it through chocolately.com. We can install chocolatey through windows powershell by running as an administrator.
* First we have to make sure to set execution policy to bypass to ensure that all the commands we run are not restricted. This is done by running Set-ExecutionPolicy bypass -scope Process
* To check for execution policy you can run Get-ExecutionPolicy
* Now go to chocolatey.com and copy the link to install it and run the command on the powershell.
* After chocolatey is install, install packer by running choco install packer
* You can check version by running packer --version

***Packer Plugins***:

* 3 different flavors. Builders, provisioners or post-processors
* Any plugin must fit the layout of packer itself
* We can either download the binary and move it to the appropriate location or we have to clone a repository and build the plugin ourself with go
* To install go we run sudo apt install golang-go

***Builders***

* The builder is what does the work to build our image
* There are over 30 builders available to us to build an image
* We will look at the amazon ebs builder
* To call in values in packer and json you format them in the following form ex "access\_key": "{{user 'aws\_access\_key'}}"
* To create an ami and append the time when it was created we can do so by running the ami name as name-{{timestamp}}. This will give us the time when the ami was created plus the name of the ami.
* The source ami filter is usually gotten from aws itself. It is just an ami already present in the console that we will use to customize to what we want.
* In the setup ensure to input the virtualization type, the name of the ami. The name usually follows a convention as shown on the template. The root device type will be ebs and the owner will be found in the console. This is the number provided by the makers of that particular ami. The most recent set to true is to ensure that only the most recent ami are built.

***Communicators***

* This is how packer works on the machine during creation. By default, this is SSH and does not need to be defined.
* If we were using another communicator other SSH we would type it as "communicator": "winrm" in case of windows. However, since SSH its usually the default we will just tell packer how to ssh into the server
* So in this case it will be ssh\_username. In our case its ubuntu but its usually ec2-user
* They help packer communicate temporarily to the instance

***Example of valid template***

{

"variables": {

"aws\_access\_key": "AKIAXB5IN5PEKWT5M5OU",

"aws\_secret\_key": "SY4m3B/C/8Y7xMS0WwM9xUJT7juWPrMChmo/hhzb"

},

"builders":[

{

"type": "amazon-ebs",

"access\_key": "{{user `aws\_access\_key`}}",

"secret\_key": "{{user `aws\_secret\_key`}}",

"region": "us-east-1",

"instance\_type": "t2.micro",

"ami\_name": "packer-base-ami-{{timestamp}}",

"source\_ami\_filter": {

"filters": {

"virtualization-type": "hvm",

"name": "ubuntu/images/\*ubuntu-focal-20.04-amd64-server-\*",

"root-device-type": "ebs"

},

"owners": ["099720109477"],

"most\_recent": true

},

"ssh\_username": "ubuntu"

}

]

}

* You can run packer fix to help fix your template and pipe the new template to a new file. So it will be packer fix packer.json > packer2.json
* The packer fix automatically fixes the entire template. See below for new template

{

"builders": [

{

"access\_key": "{{user `aws\_access\_key`}}",

"ami\_name": "packer-base-ami-{{timestamp}}",

"instance\_type": "t2.micro",

"region": "us-east-1",

"secret\_key": "{{user `aws\_secret\_key`}}",

"source\_ami\_filter": {

"filters": {

"name": "ubuntu/images/\*ubuntu-focal-20.04-amd64-server-\*",

"root-device-type": "ebs",

"virtualization-type": "hvm"

},

"most\_recent": true,

"owners": [

"099720109477"

]

},

"ssh\_username": "ubuntu",

"type": "amazon-ebs"

}

],

"variables": {

"aws\_access\_key": "AKIAXB5IN5PEKWT5M5OU",

"aws\_secret\_key": "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"

}

}

***Provisioners:***

*Shell:*

* Provisions the machine using the image's default shell
* It supports whatever the default shell is on the Linux machine you are using. It will either be bash or zsh
* In windows you will use the windows shell provisioner
* The inline command requires sudo so as to append any command that requires root privileges during the process
* The shell provisioners provides a wide variety of parameters
* You can add the inline function in which you add the command or the script name if you have one already written

*Files:*

* Here you upload files and directories to the machine
* We are going to be working with multiple scripts so we created a new directory and moved the shell.sh script into the new directory
* Curl means client URL is a command tool that enables data transfer over various network protocols. It communicates with a web or application server by specifying a relevant URL and the data that need to be sent or received
* With are going to set up a prometheus file for this example. A prometheus exporter is a piece of software that can fetch statistics from another, non-prometheus system. It can turn those statistics into prometheus metrics, using a client library.
* We need the node-exporter.tar.gz file and the node\_exporter.service file

Node Exporter is a Prometheus **exporter for server level and OS level metrics with configurable metric collectors**. It helps us in measuring various server resources such as RAM, disk space, and CPU utilization

A Tar file is an archive that consists of multiple files put into one, while GZ is a compressed file format. Thus, combining TAR and GZ into a TAR. GZ **provides you with a compressed archive**.

* We need to create a files directory
* To create the files directory we run mkdir files
* And run the command curl -J -L to preserve the files structure
* The trailing slash after a directory dictates how it is uploaded. Directories with a slash upload only the contents of the directory (/tmp/somefile1.tar)
* Directories without a slash create a new directory (/tmp/directory/\*)

***Ansible:***

* We provision using a remote ansible server by running ansible playbook over SSH
* Or we can install ansible on the remote and use it to configure the localhost
* We will be using option 2 in this case
* Here we are trying to get to the same results we did earlier except that we want to do so using ansible
* Sometimes you need to help out the provisioners by adding some inline code for them to function
* We need to run an inline command to install ansible provisioner cause the ansible local provisioner does not install ansible
* Other provisioners are chef, puppet etc
* To install ansible on the local you run sudo apt install ansible -y

{

"variables": {

"aws\_access\_key": "AKIAXB5IN5PEKWT5M5OU",

"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-east-1",

"instance\_type": "t2.micro",

"ami\_name": "packer-base-ami-{{timestamp}}",

"source\_ami\_filter": {

"filters": {

"virtualization-type": "hvm",

"name": "ubuntu/images/\*ubuntu-focal-20.04-amd64-server-\*",

"root-device-type": "ebs"

},

"owners": ["099720109477"],

"most\_recent": true

},

"ssh\_username": "ubuntu"

}

],

"provisioners":[

{

"type": "shell",

"inline": [

"sudo apt update -y && sudo apt upgrade -y",

"sudo apt install ansible -y"

]

},

{

"type": "file",

"source": "app",

"destination": "/home/ubuntu/"

},

{

"type": "ansible-local",

"playbook\_file": "playbook.yml"

}

***Post-Processors:***

* Related to the builder, run after the image is built. Generally or supplies artifacts
* For example provisioning a vagrant box
* The output : vagrant.tar.gz tells packer to take our image and compress it into a tar.gz file
* Post-processors let us do the following: repackage the image (amazon import, docker import, vagrant), upload or alter artifacts (artifice, compress, manifest, docker build, docker tag), and more such as checksum (evaluate checksums for us) and shell (This will run commands for us after the build is complete)

***Parallel builds:***

* One benefit of packer it lets us build cross platform
* This helps us run 2 builders in parallel

AMI-BUILD

* The kickstart file(KS) is a simple text file containing a list of items each identified by a keyword. It is in the root users home directory

What is the BIOS in Linux?

A BIOS (basic input output system) is a small program that controls a personal computer's hardware from the time the computer is started until the main operating system (e.g., Linux, Mac OS X or MS-DOS) takes over

What is GRUB boot Linux?

Grub is **the boot menu**. If you have more than one operating system installed, it allows you to select which one to boot. Grub is also useful for troubleshooting. You can use it to modify the boot arguments or to boot from an older kernel.

* When you boot your server BIOS starts first by running a small program that controls the personal hardware of the system until the main OS takes over. GRUB then takes over BIOS at boot time, loads itself, load the Linux kernel into memory and then turn over execution to the kernel. GRUB supports multiple Linux kernels and allows users to select between them at boot time using a menu.
* [An introduction to GRUB2 configuration for your Linux machine | Opensource.com](https://opensource.com/article/17/3/introduction-grub2-configuration-linux#:~:text=GRUB%20stands%20for%20GRand%20Unified,over%20execution%20to%20the%20kernel.)
* Headless software (e.g. "headless Java" or "headless Linux",) is **software capable of working on a device without a graphical user interface (GUI)**. Such software receives inputs and provides output through other interfaces like network or serial port and is common on servers and embedded devices.
* [What is headless system? - Definition from WhatIs.com (techtarget.com)](https://www.techtarget.com/iotagenda/definition/headless-system#:~:text=Headless%20computers%20are%20usually%20embedded,possible%20hosts%20of%20embedded%20systems.)
* [How to Mount ISO File on Linux | Linuxize](https://linuxize.com/post/how-to-mount-iso-file-on-linux/#:~:text=An%20ISO%20file%20is%20an,are%20distributed%20as%20ISO%20images.)

What is the checksum of a file?

A checksum is **a string of numbers and letters that's used to “check” whether data or a file has been altered during storage or transmission**. Checksums often accompany software downloaded from the web so that users can ensure the file or files were not modified in transit.



An ISO file is **an archive file that typically contains the complete image of a CD or DVD**. For example, most operating systems such as Windows, Linux, and macOS are distributed as ISO images.

What does LSB do in Linux?

The LSB (Linux Standard Base) is a project which aims to **standardize many of the basic software system structures of Linux**. It is based on (and extends) several open standards, such as POSIX. Among the LSB's goals is to standardize Linux, such that software can run in compiled form on any compliant Linux distribution.

[Linux Standard Base - Wikipedia](https://en.wikipedia.org/wiki/Linux_Standard_Base)