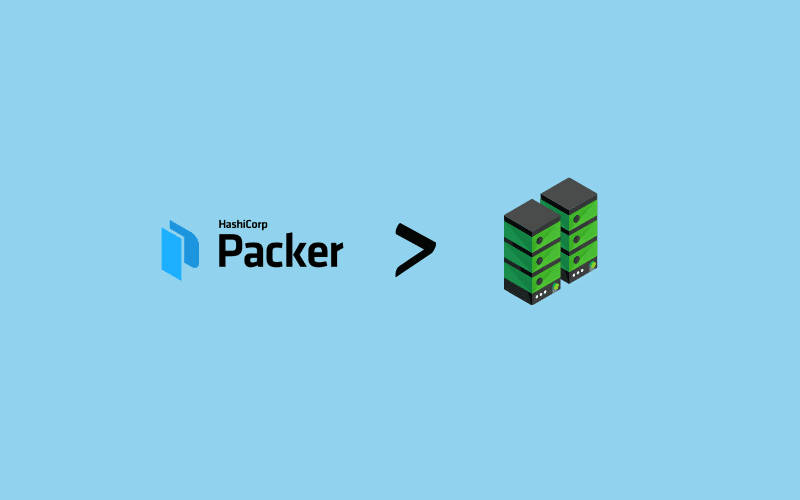
**Packer Tutorial For Beginners – Automate AMI Creation**

[](https://devopscube.com/wp-content/uploads/2021/03/kubernetes-design-7-min.png)

1

Shares

Packer is an **open-source VM image creation tool** from [Hashicorp](https://www.hashicorp.com/). It helps you automate the process of Virtual machine image creation on the cloud and on-prem virtualized environments.

To put it simply, whatever manual steps you perform to create a Virtual machine image can be **automated through a simple Packer config template**. You declare the state of the VM image you need, and Packer builds it for you.

**Table of Contents** [show](https://devopscube.com/packer-tutorial-for-beginners/)

**Packer Use Cases**

Following are the main use cases for Packer.

1. **Golden Image Creation:** With packer, you can template the configurations required for a golden VM image that can be used across organizations.
2. **Monthly VM Patching:** You can integrate Packer in your monthly VM image patching pipeline.
3. **Immutable Infrastructure:** If you want to create an immutable infrastructure using VM images as a deployable artifact, you can use Packer in your CI/CD pipeline.

Basically, all the above use cases are part of the **CI pipelines or IaaC code pipeline** where every Packer template is version controlled. Even the **packer template development or update** should go through all the CI tests before getting deployed in a project environment.

**Packer Tutorial For Beginners**

In this Packer beginner tutorial, you will learn the following.

1. The basic building blocks of Packer.
2. Packer Workflow
3. Installing Packer
4. Understanding Packer HCL template.
5. Creating an AWS AMI using Packer

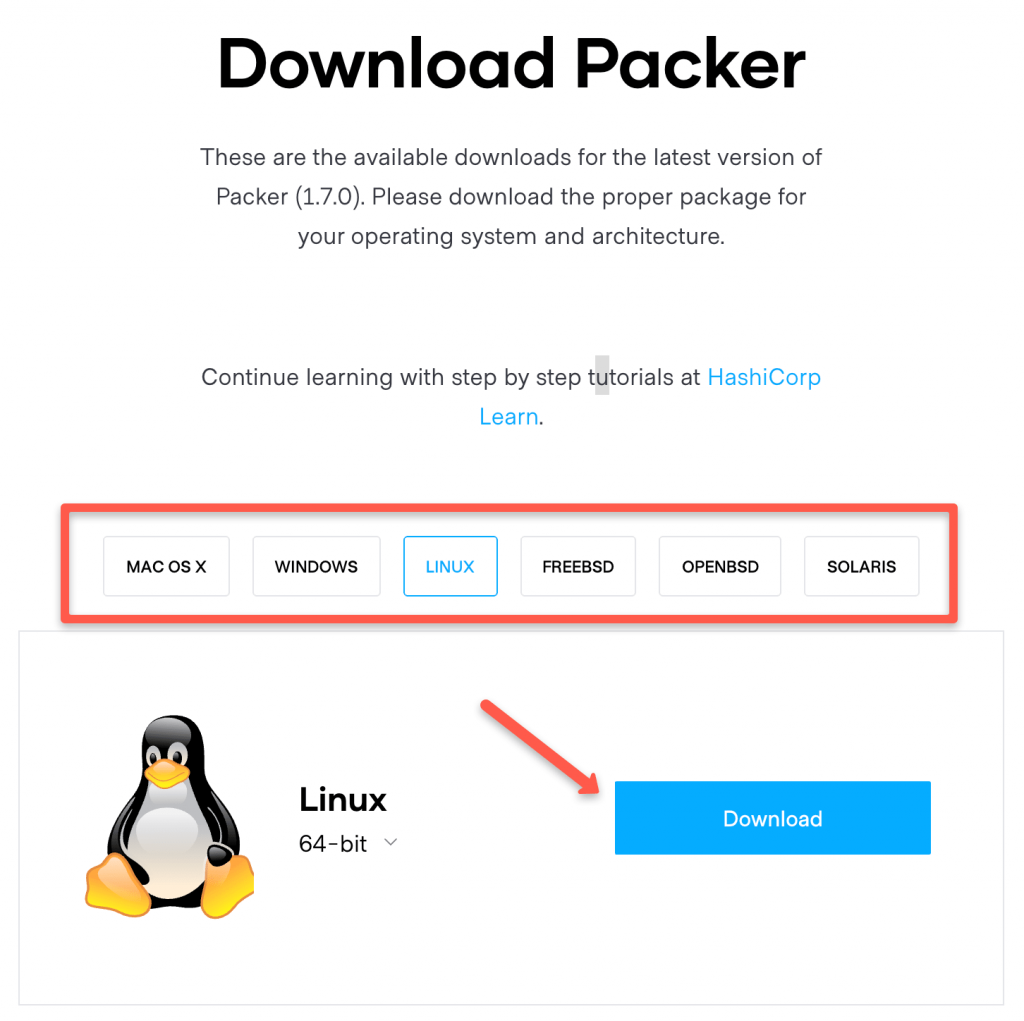
Let’s get started with the Packer setup.

**Installing Packer**

You can have Packer installed on your local workstation or on a cloud instance. In the actual project, the Packer installation would be part of the [Jenkins](https://devopscube.com/jenkins-2-tutorials-getting-started-guide/) agent or any tooling which is part of the CI/CD process.

**Note:** If you are setting up Packer on your workstation for debugging or testing purposes, you need the AWS access and secret keys for Packer to interact with AWS services. Make sure you have the AWS access keys set on the file ~/.aws/credentials. It is not mandatory to have a credentials file. You can pass the AWS credentials using the packer variable declaration as well.

**Step 1:** Download the latest Packer executable from the Packer downloads page. <https://www.packer.io/downloads.html>. It is available for Windows, Linux, and other Unix platforms.

[](https://devopscube.com/wp-content/uploads/2021/03/packer-min.png)

For Linux, you can install Packer from the respective package managers or you can get the download link from the download button and download it using wget.

For example,

wget https://releases.hashicorp.com/packer/1.7.10/packer\_1.7.10\_linux\_amd64.zip

**Step 2:** Unzip the downloaded Packer binary and set the path of the packer binary in ~/.bashrc.

Replace /path/to/packer with the actual path of Packer binary.

export PATH=$PATH:/path/to/packer

Reload bashrc file

source ~/.bashrc

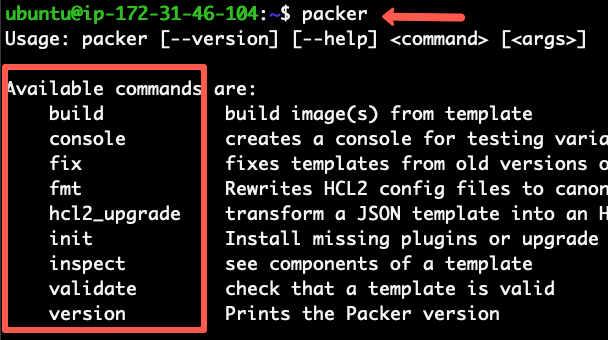
Alternatively, can move the Packer binary to the bin folder. So that it will be available in the path by default.

sudo mv packer /usr/local/bin/

**Step 3:** Verify packer installation by executing the packer command.

packer version

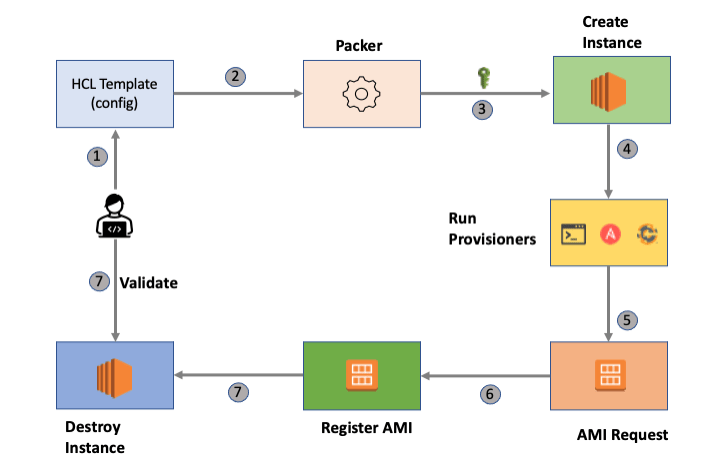
You should see the output as shown below.

[](https://devopscube.com/wp-content/uploads/2021/03/packer-output-min.png)

**Building VM Image (AWS AMI) Using Packer**

Building a VM image using Packer is a simple process. The workflow is somewhat similar to [building VMs using Vagrant](https://devopscube.com/vagrant-tutorial-beginners/)

Take a look at the following high-level Packer workflow diagram.

[](https://devopscube.com/wp-content/uploads/2021/03/packer-workflow-min.png)

Let’s break down the process.

1. Declare all the required VM configurations in an HCL (Hashicorp configuration language) or a JSON file. Let’s call it the **Packer template**.
2. To build the VM image, execute Packer with the Packer template..
3. Packer authenticates the remote cloud provider and launches a server. If you execute Packer from a cloud environment, it leverages the cloud service account for authentication.
4. Packer takes a remote connection to the server (SSH or Winrm).
5. Then it configures the server based on the provisioner you specified in the Packer template (Shell script, Ansible, Chef, etc).
6. Registers the AMI
7. Deletes the running instance.

**Packer Template Reference**

All the code used in this article is hosted on a [Github repository](https://github.com/bibinwilson/packer-templates.git). You can clone or fork it to keep it as a reference. Or you can use the template directly from the packer-aws-shell folder

git clone https://github.com/bibinwilson/packer-templates.git

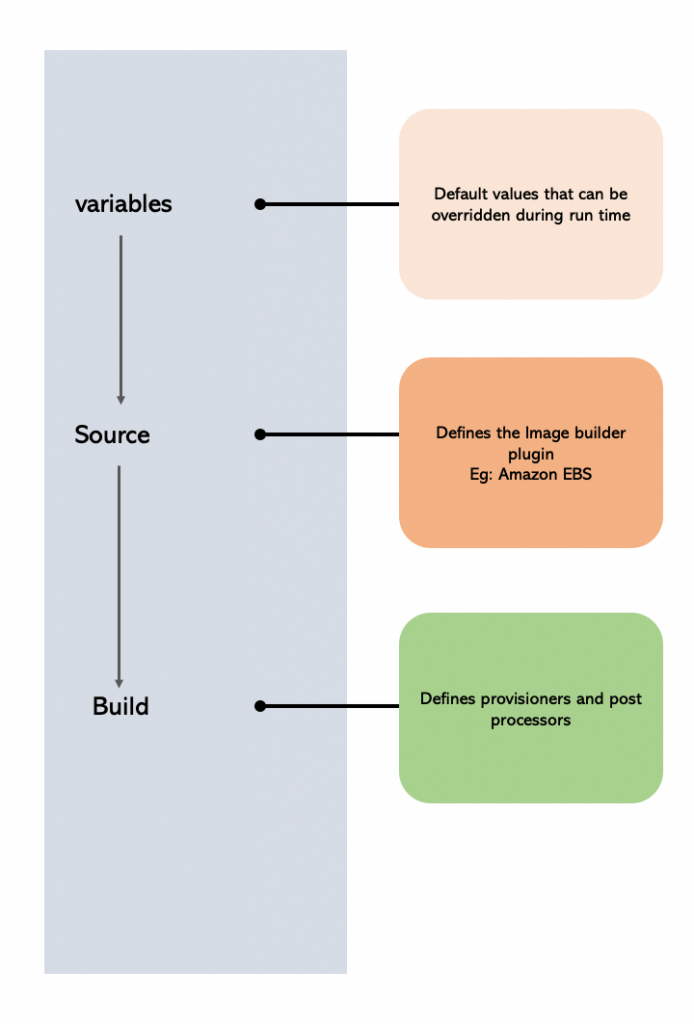
Please feel free to contribute to the repo with more templates.

**Packer HCL Type Templating**

**Note:** From version v1.7 Packer uses HCL format templates. JSON support is also there. But it’s recommended to change to HCL format as the newly developed features will not be added to the JSON-based template. If your exiting project or organization users JSON templates, you can convert them to HCL templates. See [Upgrading JSON to HCL](https://www.packer.io/docs/commands/hcl2_upgrade) for more information.

Let’s have a look at the HCL Packer template in detail.

Here is the high-level packer template structure.

[](https://devopscube.com/wp-content/uploads/2021/03/packer-template-file-min.png)

An HCL template has a few blocks. Let’s **understand each block with sample configuration** before we create the AMI with Packer

**Variables**

Using the variable block, you can declare the default Packer parameters as variables. The declared variables can be overridden from the command line during the packer build process.

You can make use of packer variables for dynamic configuration when used with CI tools like Jenkins.

Let’s understand different scenarios for variable usage.

**Using Variable Within Template**

The variables block holds all the default variables within a template. An example is shown below.

variable "ami\_id" {

type = string

default = "ami-6f68cf0f"

}

The declared variables can be accessed in other parts of the template using "${var.app\_name}" syntax. An example is shown below.

source\_ami = "${var.ami\_id}"

**Using Environment Variables in Templates**

Packer lets you use the system environment variables. First, you need to declare the environment variables in the variable section to use them in the other parts of the template.

Let’s say, you want to use the SCRIPT\_PATH environment variable that holds the path to a shell script that has to be used in the shell provisioner. You can declare that variable as shown below.

variable "script\_path" {

default = env("SCRIPT\_PATH")

}

Then, you can use the script\_path variable in the provisioned block like shown below.

provisioner "shell" {

script = "${var.script\_path}/demo-script.sh"

}

**Using Command Line Variables**

All the variables declared in the variable block can be overridden during the Packer run time using -var a flag followed by the variable value.

For example,

packer build -var 'app\_name=httpd' ami.pkr.hcl

**Note:** In the above example, ami.pkr.hcl is the packer template file.

**Using a HCL file**

You can also use an HCL file to eternalize all the variables in the template. Let’s say we have a variable file named vars.packer.hcl, you can add the variables as shown below.

app\_name = "httpd"

ami\_id = "ami-6f68cf0f"

And during the packer build, you can pass this file as shown below.

packer build -var-file="vars.packer.hcl"

**Using a JSON File**

You can also list all the variables in a JSON file and pass it during the packer build as shown below.

packer build -var-file=variables.json ami.pkr.hcl

**locals**

Locals are also variables, but they cannot be overridden. It is scoped within the Packer template.

locals {

app\_name = "httpd"

}

You can interpolate a variable with a local as shown below.

locals {

ami\_name = "${var.app\_name}-prodcution"

}

If you want to pass any **sensitive information in a variable**, you can convert it to a local value and then mark it as sensitive. This way, the value won’t be printed anywhere in the Packer build logs.

local "secret\_key" {

key = "${var.secret\_key}"

sensitive = true

}

**Source**

Packer works on the concept of the builder plugin. For every cloud provider, there are specific builder plugins that interact with the Cloud provider API to create the images.

In this block, we mention the builder plugin details. For example, if you want to build AWS AMI, you can specify amazon-ebs as a builder plugin. You have to provide all the mandatory parameters associated with the amazon-ebs builder plugin.

Also, In the following code block, you can see that we are calling ${var.ami\_id} declared in the variable block.

Check the [official builder plugin documentation](https://www.packer.io/docs/builders) to know about the list of supported builders.

source "amazon-ebs" "httpd" {

ami\_name = "PACKER-DEMO-${local.app\_name}"

instance\_type = "t2.micro"

region = "eu-west-1"

source\_ami = "${var.ami\_id}"

ssh\_username = "ec2-user"

tags = {

Env = "DEMO"

Name = "PACKER-DEMO-${var.app\_name}"

}

}

**build**

In this block, we declare all the provisioners and post-processor.

build {

sources = ["source.amazon-ebs.httpd"]

provisioner "shell" {

script = "script.sh"

}

}

**Provisioner**

Provisioner block is part of the build block. Here you can mention how you want to configure the VM image. You can **run shell scripts, Ansible playbooks, Check cookbooks**, etc. See the [provisioners page](https://www.packer.io/docs/provisioners) for all the supported provisioners.

The script or ansible-playbook should be present in the folder where you have the Packer template.

provisioner "shell" {

script = "script.sh"

}

Also, there is a file provisioner using which you can copy files from the packer machine to the AMI.

For example,

provisioner "file"{

source = "/build/artifact/code.jar"

destination = "/app/code.jar"

}

**Post-Processor**

Post-Processor is not a mandatory block. In this block, we can specify what to do after creating the Image. For example, you can execute a local shell script to create a file with all the image details.

post-processor "shell-local" {

inline = ["echo foo"]

}

**Creating AWS AMI Using Packer HCL Template**

Now let’s see how to create an AMI using the Packer HCL template.

**Note:** I have given a basic example with minimal configurations just for your understanding. When it comes to Packer implementation in the real-time project, more customizations may be needed in terms of variables, naming schemes, builder plugging, etc.

Here is what the template does.

1. Packer **deploys an ec2 RHEL instance** based on AMI id ami-01e78c5619c5e68b4 in the eu-west-1 region.
2. **Executes the shell script** on the RHEL instance from the script.sh script provided in the template directory
3. **Registers the AMI** with the name given in the template.
4. Deletes the running machine.

Let’s get our hands dirty with a practical example.

**Step 1:** Create a folder named “packer-vm“.

mkdir packer-vm

**Step 2:** Inside the packer-vm folder, create a file named “vm.pkr.hcl” and copy the following HCL template.

If we put together all the blocks I explained in the above section, we get the following HCL Template.

variable "ami\_id" {

type = string

default = "ami-01e78c5619c5e68b4"

}

variable "app\_name" {

type = string

default = "httpd"

}

locals {

app\_name = "httpd"

}

source "amazon-ebs" "httpd" {

ami\_name = "PACKER-DEMO-${local.app\_name}"

instance\_type = "t2.micro"

region = "us-west-2"

source\_ami = "${var.ami\_id}"

ssh\_username = "ec2-user"

tags = {

Env = "DEMO"

Name = "PACKER-DEMO-${var.app\_name}"

}

}

build {

sources = ["source.amazon-ebs.httpd"]

provisioner "shell" {

script = "script/script.sh"

}

post-processor "shell-local" {

inline = ["echo foo"]

}

}

**Step 3:** Create a folder named script inside the packer-vm folder and cd into the script directory.

mkdir script && cd script

**Step 3:** Create a file name script.sh and copy the following shell script to it. It just updates the yum repo and installs httpd. Here, you can add all the shell scripts that you want to get executed during the AMI creation.

#!/bin/bash

sudo yum -y update

sudo yum install -y httpd

**Step 4:** [**Very Important**] Now that we have all the required files, let’s understand the required packer authentication.

If you are running Packer on your workstation for debugging or testing purposes, you **need the AWS access and secret keys for Packer to interact with AWS services**. Make sure you have the AWS access keys set on the file ~/.aws/credentials.

Alternatively, you can pass AWS keys in the source block, as shown below. It is **not a recommended practice**. Accidently you might commit this to a source code repo.

source "amazon-ebs" "example" {

access\_key = "ASDFASDFASDFJQWEKONNOASDF"

secret\_key = "SDFQ324876134HJKLASDBJKASDJFHBAJSDFJKHASKDG"

}

If you are running Packer from an **AWS ec2 instance**, make sure you add an **IAM role with permissions** to interact with AWS ec2 services. I have added the required IAM policy JSON in Github. See [Packer IAM policy JSON](https://github.com/bibinwilson/packer-templates/blob/main/packer-aws-shell/aws-policy.json)

**Step 5:** Now let’s validate the Packer HCL template before we build it. Validation is very important in the CI process.

packer validate vm.pkr.hcl

**Step 6:** Run the packer build using the following command.

packer build vm.pkr.hcl

The above command runs for almost 3 – 5 minutes.

If you check your AMI list, you should see an AMI named PACKER-DEMO-httpd.

**Unit Testing Packer Templates**

Packer templates should be treated as any other Infrastructure as code that involved testing.

You can write Packer tests using [Check Inspec](https://github.com/inspec/inspec).

When implementing packer in production workflows, ensure you write necessary tests as part of your CI/CD pipeline.

Check out [this repo](https://github.com/travis-ci/packer-templates/tree/master/spec) for packer inspec reference.

**Creating AWS AMI Using Packer JSON Template [For Old Versions]**

**Important Note:** JSON type templating is not in development anymore. This section is just for a reference to the old type JSON template.

Let’s have a look at the JSON style template in detail.

The JSON template has the following three main parts.

1. **variables**: – Where you define custom variables.
2. **builders**: – Where you mention all the required AMI parameters.
3. **provisioners**: – Where you can integrate a shell script, ansible play, or a chef cookbook for configuring a required application in the AMI.

An example template for packaging an AWS AMI is given below.

{

"variables": {

"ami\_id": "ami-01e78c5619c5e68b4",

"app\_name": "httpd"

},

"builders": [{

"type": "amazon-ebs",

"region": "eu-west-2",

"source\_ami": "{{user `ami\_id`}}",

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

"ssh\_username": "ec2-user",

"ami\_name": "PACKER-DEMO-{{user `app\_name` }}",

"tags": {

"Name": "PACKER-DEMO-{{user `app\_name` }}",

"Env": "DEMO"

}

}],

"provisioners": [

{

"type": "shell",

"script": "demo-script.sh"

}

]

}

In the above example configuration, we are passing the AWS access keys and secret keys as variables. It is not recommended to use access keys as variables. If you have already set the credentials in ~/.aws/credentials file, you don’t need to pass access keys as variables.

Also, we have used a shell provisioner which calls a demo-script.sh file.

**Packaging an Image**

In this example, we will bake a t2.micro AMI using a shell provisioner. A shell script that has an update and HTTPD install instruction.

We assume that you have the AWS access keys and region set in the ~/.aws/credentials file.

Here we are going to user Oregon region and a Redhat AMI with AMI id ami-6f68cf0f

Follow the steps given below to set up the project.

**Step 1:** Create a folder call packer

mkdir packer

**Step 2:** Create a script file named demo-script.sh and copy the following contents on to it.

#!/bin/bash

sudo yum -y update

sudo yum install -y httpd

The above script just updates the repository and installs httpd.

**Step 3:** Create a httpd.json file with the following contents.

{

"variables": {

"ami\_id": "ami-6f68cf0f",

"app\_name": "httpd"

},

"builders": [{

"type": "amazon-ebs",

"region": "eu-west-1",

"source\_ami": "{{user `ami\_id`}}",

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

"ssh\_username": "ec2-user",

"ami\_name": "PACKER-DEMO-{{user `app\_name` }}",

"tags": {

"Name": "PACKER-DEMO-{{user `app\_name` }}",

"Env": "DEMO"

}

}],

"provisioners": [

{

"type": "shell",

"script": "demo-script.sh"

}

]

}

We have our template ready. The next step is to execute the template for packaging the AMI with HTTPD installation.

**Step 4:** let’s validate and inspect our template using the following commands.

packer validate httpd.json

packer inspect httpd.json

**Note:** If you are using command line variables or a file as a variable, you should pass it while validating it. An example is shown below.

packer validate -var='env=TEST' httpd.json

The above command should validate and inspect without any errors.

**Step 5:** To build our new AMI, use the following command.

packer build httpd.json

The above command will build a new AMI.

You can also debug the image creation. Check out [this thread](http://discuss.devopscube.com/t/debugging-packer-image-creation/32) for packer debugging.

**Few Packer Tips**

**Tip 1:** You can capture the output of the image build to a file using the following command.

packer build httpd.json 2>&1 | sudo tee output.txt

Tip 2: Then you can extract the new AMI id to a file named ami.txt using the following command.

tail -2 output.txt | head -2 | awk 'match($0, /ami-.\*/) { print substr($0, RSTART, RLENGTH) }' > sudo ami.txt

**Conclusion**

In this **packer tutorial for beginners,** I have covered the basics of image creation using Packer.

There are more customizations you can do in your Paker templates when it comes to a real-time project.

Again, it depends on what you want to achieve using packer.

You might be using Jenkins for image build pipelines or maybe custom utilities.

Either way, drop a comment and let me know how you manage images in your project.