Packer

- Open source machine image creation tool
- Automates installation & configurations on Packer-made
- works with multiple (aws, gcp...) platforms - even from same config
- Eliminates manual steps for Golden Image creation

Use Cases

- Create Golden Images across platforms & env
- Establish an image factory based on new commits for [illegible]
- Automate your monthly patching for new/existing workloads.
- Creat immutable infra using packer in CI/CD pipeline

Benefits

- Version controlled: Images are defined and versioned as code ([illegible])
- Consistent Images: Cross-Platform consistency
- Automate Everything. Stop the Manual Madness.

Core components

- Packer builds images using template.
- Templates can be built using JSON (old) or HCL (Hashicorp Configur
- Template defines settings using blocks:
  - ↳ Original Image to use (source)
  - ↳ when to build Image (AWS, GCP...)
  - ↳ Files to upload to Image (scripts, packages...)
  - ↳ Installations & config of Machine Images
  - ↳ Data to Retrieve when Building
- Components to make template:
  - ↳ Source
  - ↳ Builders
  - ↳ Variables
  - ↳ provisioners
  - ↳ communicators
  - ↳ Post-processors

1) Source :-

defines initial image to use to create ur customised image. Any defined source is reusable within build blocks

Eg:

↳ for building AWS Image (AMI), you need to point to an existing AMI to customise.

2) Builders:-

- Builders are responsible for creating machines from base image, customising the image as defined & then creating a resulting image
- Builders are pulgins that are developed to work with a specific platform (i.e, AWS, Azure, ...)
- 'Everything done' to images is done within BUILD block
- This is where customization "work" happens.

3) Provisioners:

- Use built-in & 3rd party integrations to install packages & configure machi images.
- Built-in integrations include file & different shell options
- 3rd party integrations include:

Ansible - run playbooks

Chef - run cookbooks

Inspec - run Inspec profiles

Powershell - Execu powShel script

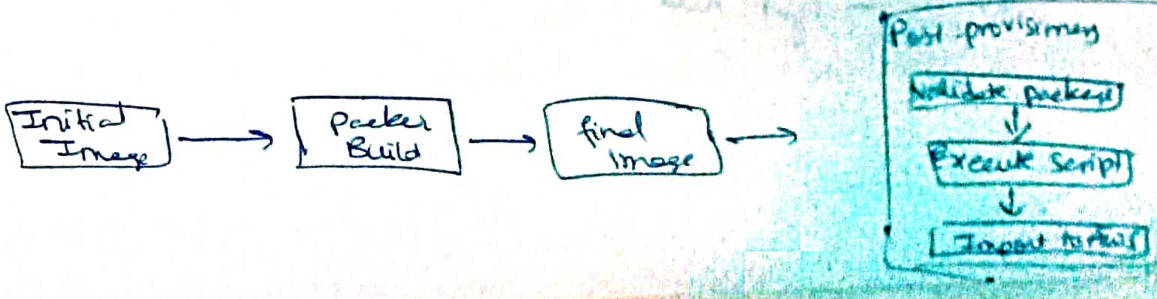
puppet - run puppet manifest

windows shell - runs commands using window cmd.

4) post-processors

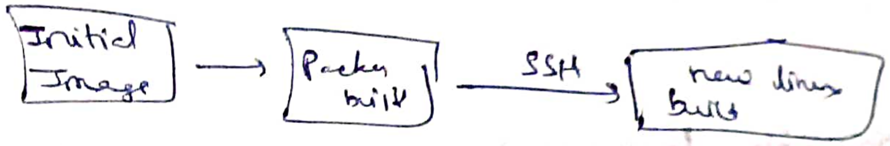
- Are executed after image is built & provisioners are complete. It can be used to upload artifacts, execute updated scripts, validate installs, or import an image. Executes after final image get created.

Eg: Import package to AWS as an AMI



5) Communicators:

. Mechanism that Packer will use to communicate with new l
& upload files, execute scripts, ...



6) Variables:

- Used to define defaults during a build.
- Can be defined in .pkrvars.hcl file or .auto.pkrvars.hcl, the default .pkr file or any other file name if referenced whe executing buid.
- You can also declare individually using -var option.

Use Packer

1) Install Packer
2) Create Template
3) Build Machine Image.

We use CLI (Command Line Interface) to interact with Packer

Packer command Lines

Packer uses a subcommand & additional arguments to execute Packer functionality. All commands start with "packer" command.

$$\$\ \underbrace{\text{packer}}_{\text{packer command}}\quad \underbrace{\text{build}}_{\text{sub command}}\quad \underbrace{\text{-var-file = var.pkr.hcl}}_{\text{argument (option)}}\quad \underbrace{\text{aws.pkr.hcl}}_{\text{template name}}$$

— lab 2 - git hub

→ $ packet help (or) $ packer -h
gives some commands of packer.

→ $ packer validate <templatename>
Shows incase of any errors, if not nothing displays

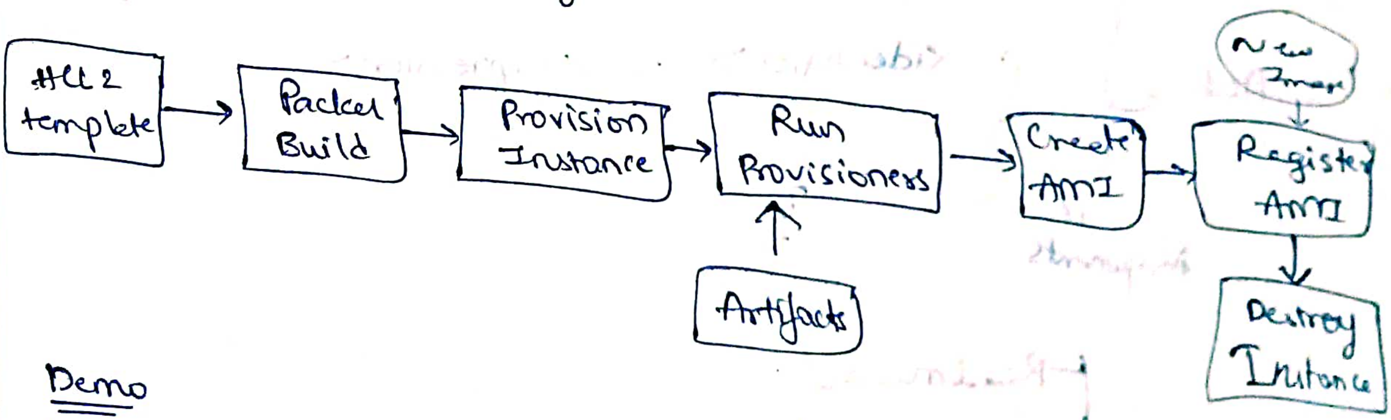
→ $ packer fmt <template name>
formats/align the code in template like terraform fmt.

→ $ packer inspect <templatename>
all info about packer file. like type, name....ia

→ $ packet build <templatename>
Its like terraform apply. It creates keypairs, SG....by default.

The Packer Workflow:-

How packer works in background.



Demo

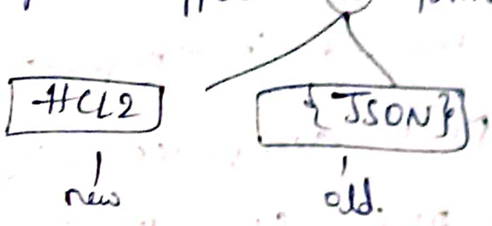
$ packet build <tname>
can cross check soon after u run this command in console. for AMI, EC2, Keypair, SG.....

AMI create చేయాలంటే ఒక instance ని create చేస్తది. So Build మొత్తం run అయ్యాక done instance is stop then terminate చేస్తది. SG, KP, EBS అన్నీ delete చేసి AMI create ఐతది.

Writing Packer Templates:-

Functionality & Behaviours of Packer is defined by template. Template consist of declarations & commands, such as what (builders, provisioners, etc.) to use, how to configure plugins & what ord run them.

packer templates supports ② formats



HCL is designed to strike a balance b/w human readable machine passable.. They are easy to develop & read.

filename → <name>.pkr.hcl

syntax

```
<Block type> "<Block label" "<block label>" {
        <identifier> = <expression>
}
```

Block

Arguments

```
├ Readme.md
├ name.pkr.hcl
└ name.pkrvars.hcl
```

→ convert json to Hcl.

$ packer hcl2_upgrade command

changes Json → Hcl template

Source Block
Build Block

} order of block within a template is not significant

⊗ The order of provisioner or post-processor blocks within a build is the only major feature where block order matters.

→ Commenting in HCl

→ single line comment
// → single line "
/* and */ → start & stop delimiters - might span over multiple lines.

Source Block

Define initial image to use create ~~customised~~ fixes customised image. & how to launch image & connect to image.
↓ type ↓ SSH

Builder Block

Build blocks are used in tandem with source blocks & define what Packer should do with image after it is launched. You can specify 1 or more build block in template. Each builder block can reference 1 or more source blocks. There are many config options available for given builder. Some may be optional.

Provisioners

Used to install & configure machine image. after it reboots. You would use one or many provisioners to customize the image as needed. Part of build block.

```
build {
    source = ["source. ... "]

file provisioner {
    provisioner "file" {
        destination = "/tmp"              will run 1st.
        source      = "files"
    }

shell provisioners {
    provisioner "shell" {
        script = "scripts/setup.sh"         2nd
    }
    provisioner "shell" {                   3rd
        inline = ["echo ${var.version} > ~/deployment_version"]
    }
}
```

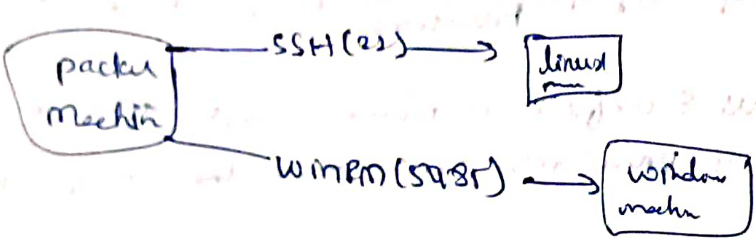
Post-processors

can be used to specify what to do after image is created part of build block & not mandatory. It creates json which summarize build process.

```
build {
  sources {
    ....
  }
  post processors "shell-local" {
    inline = ["rm/tmp/script.sh"]
  }
}
```

Communicators

mechanism that packer uses to upload files & execute scripts when creating file artifact. They are configured within source block of packer template.



SSH is default communicators so don't need to specify communicator type.

```
ssh_username = " "
winrm_username = " "
  tags {
  }
```

(.auto.pkrvars.hcl)
or
Variables (.pkrvars.hcl)

Variables or local variables used to define values for arguments throughout packer template.

```
variable "submit id" {
  type -
  default -
}
```

var.name of ${var.name} [illegible] by attaching

types
- string — takes default
- number
- list
- map
- bool

[illegible]

{ fmt   } { tf. init }
{ validate   } {   plan }
{ build   } {   apply }

→ Demo-Building AWS AMI with packer:

vault.pkr.hcl →→

```
        variable "aws_region"{
              type = string
              default = "${env{"AWS REGION"}}"
              }
      variable  "vault_zip"{
           type = str
        default = "c:\\user\\btkr\\download\\valt 1.7.1_wind
                                                    _amd64.zip"
           }
     variable  "vpc_id"{
          type   = strg
        default = "vpc.-123"
         variable "subnet_id"{
             type = st
            default = "subnet-012"
              }
        data "amazon-ami" "amazn-linux-2"{
          filters = {
              name    = "amzn2--gp2"
          root=device-type = "ebs"
          virtualization-type = "hvm"
              }
          most_recent  = true
             owners = ["amazon"]
           region   = var.aws_region.
                }
          local {
              timestamp = regex_replace (timestamp(), "{-T2-]",
                                                     "-}
             }
   source "amazon-ebs" "amazon-ebs-amazon-linux 2"{
```

```
ami_desqpritm =
ami-name =
    ami-regim =
  ami-virtualisatm = "hvm"
  associcte-public-ip-addren = true
  force-delete-snapshot = true
    force-deregitter = true
    instance-type = "t2.small"
    regim = var.aws-regim
    Source_ami = data. amazon .ami.amazn linux-2.id
     spot-price = "0"
     ssh-pty = true
     ssh-timeout = "5m"
    ssh-usernm = "ec2-user"
            tags = {
                name = "heshicorp vault"
                OS = "amazon linux 2"
              }
             subnet_id = var. subnet_id
             vpc-id = "    vpc-"
          }

build {
   Source = ["source.amazon-ebs.amazon-ebs-amazonlinux-2"]
  provisioner "file" {
       destination = "/tmp/vault.zip"
         Source = var.vault.zip
               }
    provisioner "file" {
              dest = "/tmp"
              source = "files"
               }
         }
```

```
$ packer fmt        vault.pkr.hcl
$ packer validate   vault.pkr.hcl

$ export AWS_ACCESS_KEY_ID = "        "
$ export AWS_SECRET_A_K = "        "
$   "    AWS_REGION = "        "

$ packer build      vault.pkr.hcl.
```

Backnd లో proces

Instance create ఐతది, with keypair, SG, ami which are temporary. కావాల్సిన files ఇచ్చిన destination లో upload చేస్తది. SSH ki connect avthadi with public IP. ఇవి జరిగాక instance ready ఐతది.

1st - Creates tempary keypair & SG.
↓
0.0.0.0/0.

2nd - launches instance.

3rd - adds tags

4th - connect to SSH with public IP

5th - connected to SSH

6th - then uploads files to given destination

7th - stoping instance.

~~8th~~ 9th - Once it is stops it creates snapshots of the instance. And it creates an AMI from that snapshot.

8th - waits instance to stop & stopped.

9th →

10th) Creating an AMI from stopped instance

11th → waiting for AMI to be ready (takes time) & gives ami id

12th - It modifies attributes like desp.

13th - tagging snapshots & adding tag.

14th creating snapshot tag

15th – terminating instance

16th – cleanup & deletion up SG, keypair

Build finished &

The artifacts of successful builds are AMIs created

us-e1 -amid -1234..

→ To build ~~multiple~~ AMI in diff regions

↳ need to add regions list ami_regions = ["–", "", "" in source block.

→ To build Images for diff Operating System (~~diff~~ multiple Image types)

☆

```
data "amazon-ami" "windows-2019" {
    filters = {
        name = "widnws-server-2019"
    }
    most_recent = true
    owners = ["81234"]
    region = "eu-west-1"
}
local { timestamp = regex ........ }
source "amazon-ebs" "windows2019" {
```

2 sources & 2 data blocks for 2012 & 2019 win & then 1 build block for them where we need to give source of 2 blocks in 1 build block.

☆ → Variable → priority

low → high

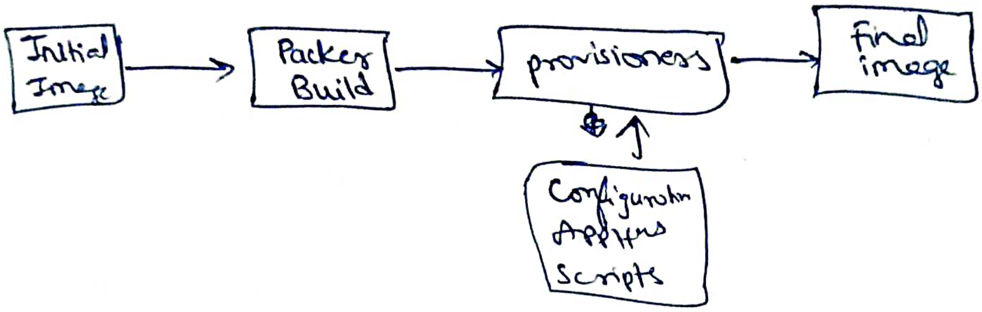
default values → Env variables → variable in variable definition file → using -var or -var-file CLI option → variable entered via CLI prompt.

→ local variables cannot be overridden

constant. can defined within template.

→ Provisioners

3rd party integrations to install packages & configure the M Image.



↳ Uses

1) installing packages

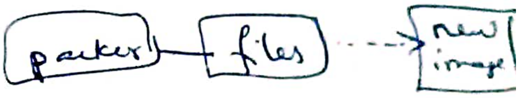
2) creating users

3) downloading appltn code.

↳ upld single file

```
provisioner "file" {
        source = "packer.zip"
        destination = "/tmp/packer.zip"
    }
```

↳ upld multiple file



```
provisioner "file" {
source = "/files"
destination = "/tmp"
        }
```

Use scripts even by executing scripts.

```
provisioner "shell" {
        script = " install.sh"
    }
```

run command on machine

```
provisioner "shell" {
        inline = [
            "echo Installing updates",
            "sudo  apt-get update",
            "sudo apt-get install -y nginx"
        ]
    }
```