Introduction to HashiCorp Packer

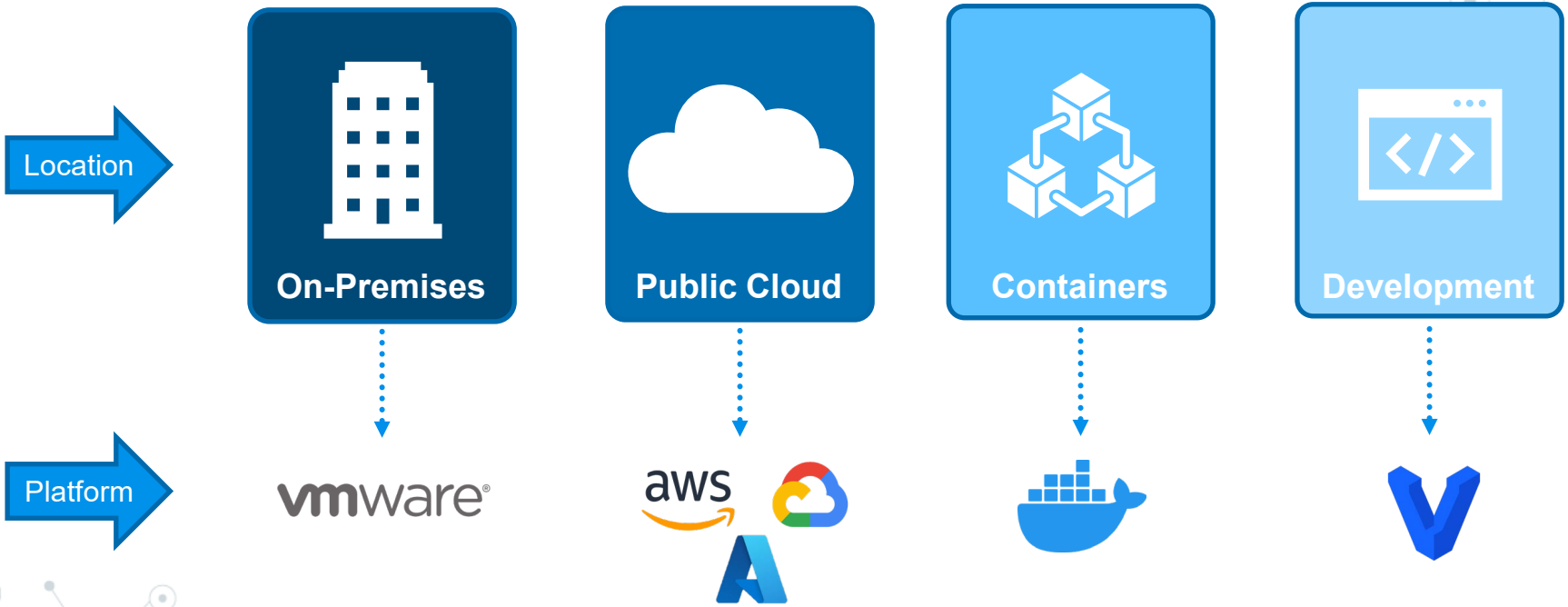
Introduction to HashiCorp Packer

Section Overview

- What is Packer?
- Primary Use Cases
- Core Components
- Installing Packer
- Interacting with Packer (CLI)
- Packer Workflow

Why Use Packer?

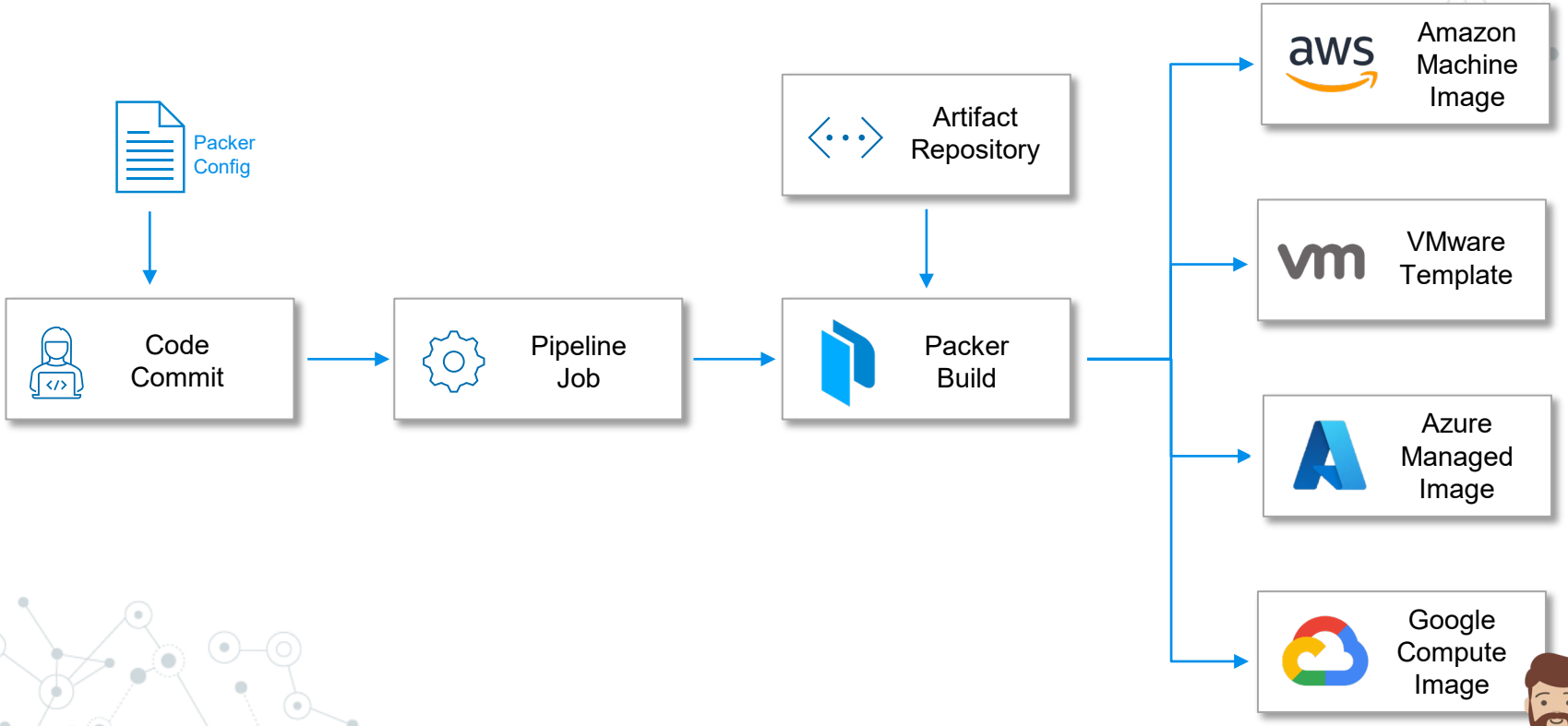
Evolving Architecture at Enterprises



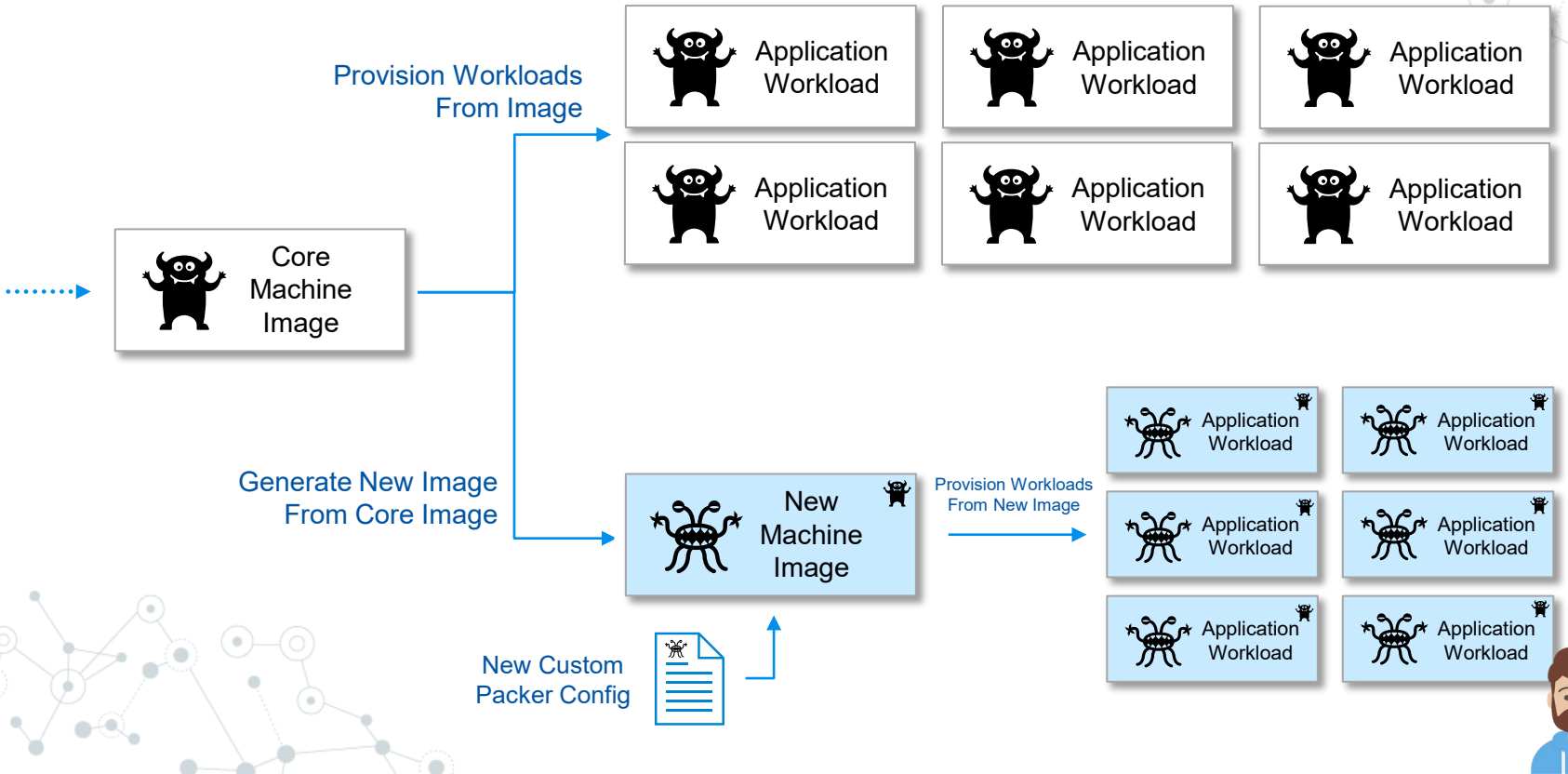
Each Platform Requires Its Own Type Of Image

Automate Image Builds Across Platforms

Golden Images For All Your Workloads

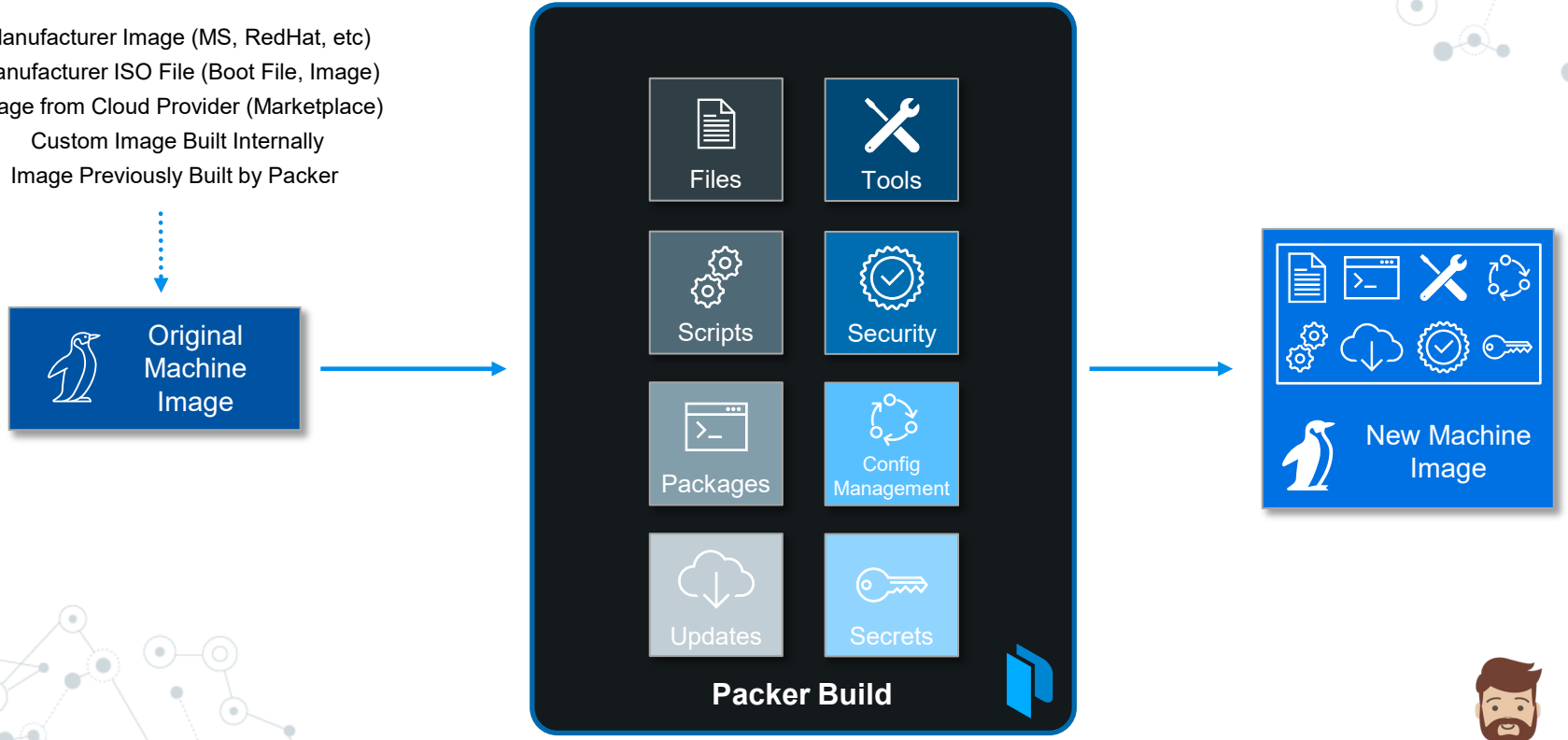


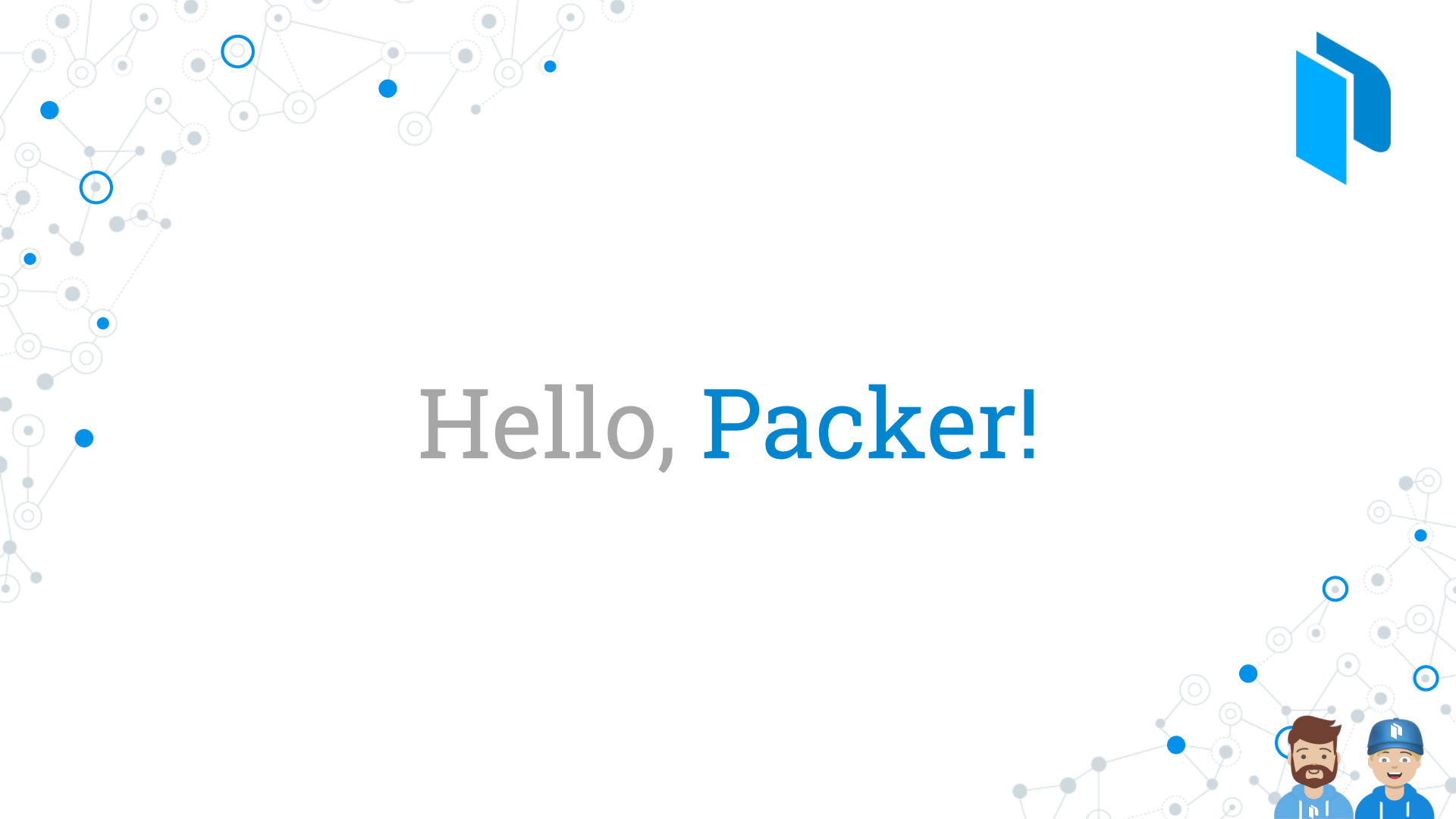
Maintain Consistency Across Workloads



Building Your Ideal Image for Workloads

Manufacturer Image (MS, RedHat, etc)
Manufacturer ISO File (Boot File, Image)
Image from Cloud Provider (Marketplace)
Custom Image Built Internally
Image Previously Built by Packer




Hello, Packer!

HashiCorp Suite of Tools

HashiCorp Terraform

HashiCorp Vault

HashiCorp Consul

HashiCorp Nomad

HashiCorp Waypoint

HashiCorp Boundary

HashiCorp Packer

HashiCorp Vagrant

What is HashiCorp Packer?

- ✓ Open-Source Machine Image Creation Tool
- ✓ Automates the Installation and Configuration on Packer-made Images
- ✓ Works with Multiple Platforms – Even from the Same Configuration
- ✓ Eliminates Manual Steps for Golden Image Creation

HashiCorp Packer

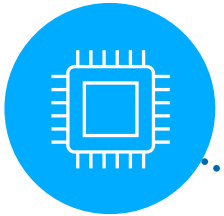
aws docker vmware Azure Google Cloud openstack DigitalOcean

Primary Use Cases

Establish an Image Factory Based on New Commits for Continuous Delivery

Automate Your Monthly Patching For New/Existing Workloads

Create Golden Images Across Platforms and Environments



Create Immutable Infrastructure Using Packer in CI/CD Pipeline

Benefits of Using Packer

Version Controlled

Images are Defined and Versioned as Code

Rather than the manual control of images, using Packer allows you to version control your images to simplify long-term management and updates within the organization

Consistent Images

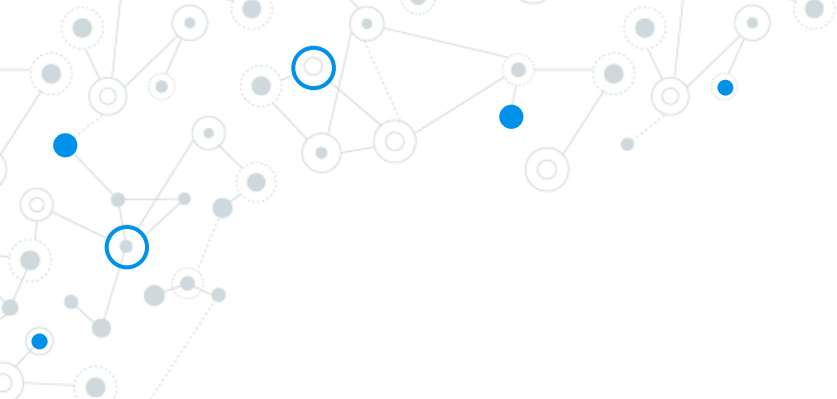
Cross-Platform Consistency

When you use multiple platforms for workloads, maintaining like-for-like images becomes a demanding task. Packer can assist with management across multiple platforms.

Automate Everything

Stop the Manual Madness

Manually having to manage images across multiple platforms is prone to human-error and creates unneeded administration overhead.



Core Components of Packer

Core Components

- HashiCorp Packer builds images using a **template**
- Templates can be built using either JSON(old) or HCL2(recommended for Packer 1.7.0+)
- Template defines settings using blocks:
  - ✓ Original Image to Use (source)
  - ✓ Where to Build the Image (AWS, VMware, OpenStack)
  - ✓ Files to Upload to the Image (scripts, packages, certificates)
  - ✓ Installation and Configuration of the Machine Image
  - ✓ Data to Retrieve when Building

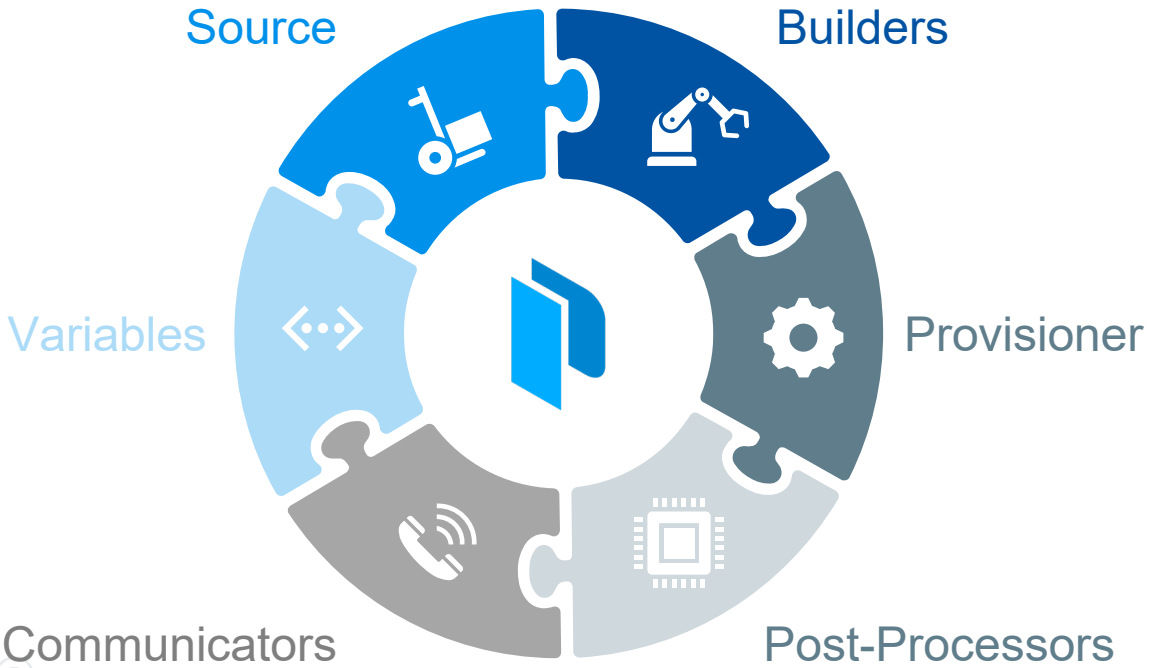
Core Components

Template Example (HCL2)

```
source "amazon-ebs" "aws-example" {
  ami_name      = "${var.ami_name}"
  instance_type = "t3.medium"
  region        = "us-east-1"
  source_ami_filter {
    filters = {
      name                = "${var.source_ami_name}"
      root-device-type    = "ebs"
      virtualization-type = "hvm"
    }
    owners = ["amazon"]
  }
  ssh_username = "ec2-user"
  subnet_id    = "${var.subnet_id}"
  tags = {
    Name = "${var.ami_name}"
  }
  vpc_id = "vpc-1234567890"
}
```

```
build {
  sources = ["source.amazon-ebs.aws-example"]

  provisioner "file" {
    destination = "/tmp"
    source      = "files"
  }
  provisioner "shell" {
    script = "scripts/setup.sh"
  }
  provisioner "shell" {
    script = "scripts/vault.sh"
  }
}
```

Core Components



Only available in HCL templates

Source

- Source defines the initial image to use to create your customized image. Any defined source is reusable within build blocks

- For example:
  - Building a new AWS image (AMI), you need to point to an existing AMI to customize
  - Creating a new vSphere template requires the name of the source VM
  - Building new Google Compute images needs a source image to start

azure.pkr.hcl

```
source "azure-arm" "azure-arm-centos-7" {
  image_offer     = "CentOS"
  image_publisher = "OpenLogic"
  image_sku       = "7.7"
  os_type         = "Linux"
  subscription_id = "${var.azure_subscription_id}"
}
```

Builders

- Builders are responsible for creating machines from the base image, customizing the image as defined, and then creating a resulting image
- Builders are plugins that are developed to work with a specific platform (i.e., AWS, Azure, VMware, OpenStack, Docker)
- Everything done to the image is done within the **BUILD** block
- This is where the customization "work" happens

azure.pkr.hcl

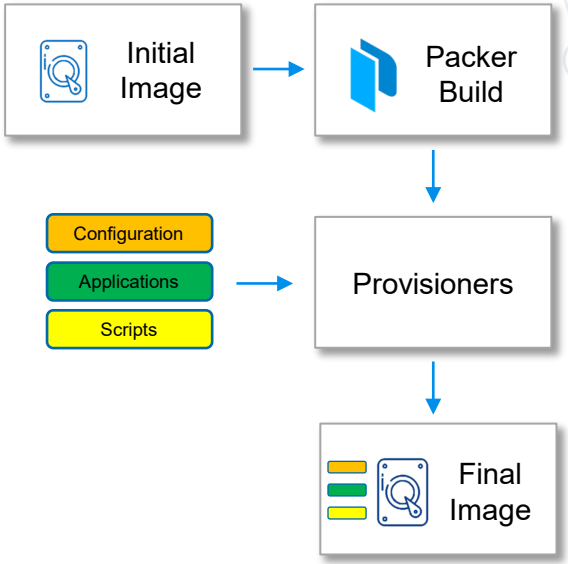
```
build {
  source = ["source.azure-arm.azure-arm-centos-7"]

  provisioner "file" {
    destination = "/tmp/package_a.zip"
    source      = "${var.package_a_zip}"
  }
}
```

Uses the Azure builder to create a new Azure Machine Image

packer.io/docs/builders

Provisioners

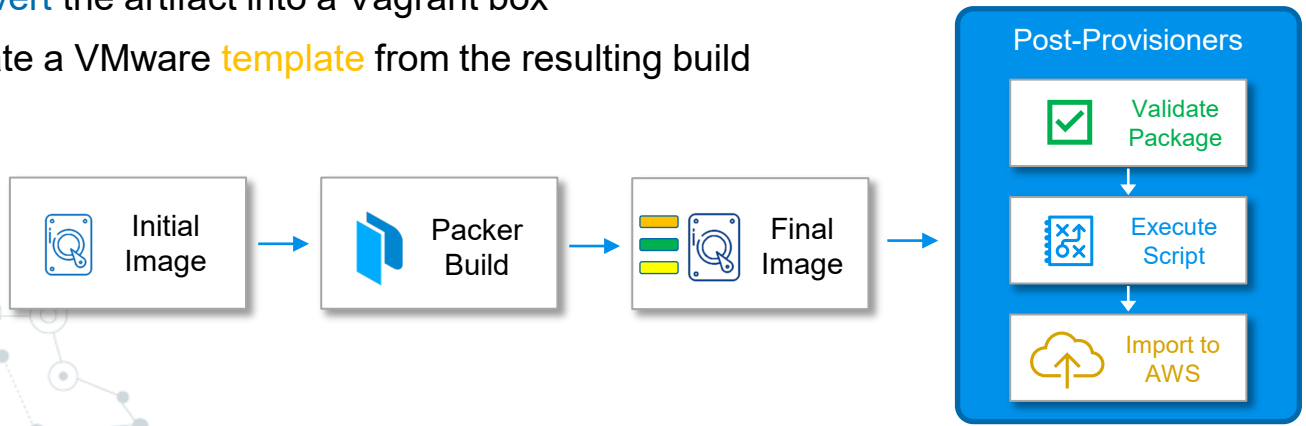
- Provisioners use built-in and third-party integrations to install packages and configure the machine image
- Built-in integrations include file and different shell options
- Third-party integrations include:
  - Ansible – run playbooks
  - Chef – run cookbooks
  - InSpec – run InSpec profiles
  - PowerShell – execute PowerShell scripts
  - Puppet – run Puppet manifest
  - Salt – configure based on Salt state
  - Windows Shell – runs commands using Windows cmd



Post-Processors

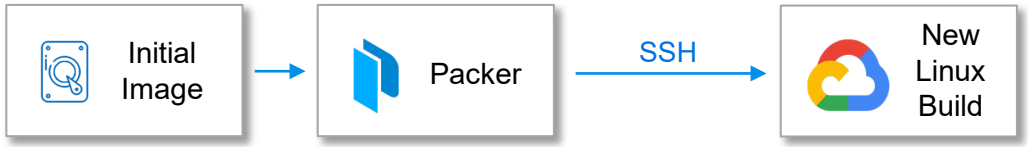
- Post-processors are executed after the image is built and provisioners are complete. It can be used to upload artifacts, execute uploaded scripts, validate installs, or import an image

- Examples include:
  - Validate a package using a checksum
  - Import a package to AWS as an Amazon Machine Image
  - Push a Docker image to a registry
  - Convert the artifact into a Vagrant box
  - Create a VMware template from the resulting build



Communicators

- Communicators are the mechanism that Packer will use to communicate with the new build and upload files, execute scripts, etc.
- Two Communicators available today:
  - SSH
  - WinRM



Variables

- HashiCorp Packer can use variables to define defaults during a build
- Variables can be declared in a .pkrvars.hcl file or .auto.pkrvars.hcl, the default .pkr file, or any other file name if referenced when executing the build.
- You can also declare individually using the –var option
- Variable declaration block:

```
variable "image_id" {
  type        = string
  description = "The id of the machine image (AMI) to use for the server."
  default = "ami-1234abcd"

  validation {
    condition     = length(var.image_id) > 4 && substr(var.image_id, 0, 4) == "ami-"
    error_message = "The image_id value must be a valid AMI id, starting with \"ami-\"."
  }
}
```

About HashiCorp Packer

HashiCorp Packer is platform agnostic….meaning it can be installed and run on many different underlying platforms

Cloud-Based Machines (AWS Instances, Azure Virtual Machines, Google Compute)

VMware Virtual Machines

Physical Servers

Desktop/Laptop

About HashiCorp Packer

HashiCorp Packer is also available for many different operating systems:

- ✓ macOS
- ✓ Windows
- ✓ Linux
- ✓ FreeBSD
- ✓ OpenBSD
- ✓ Solaris

Using Packer

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

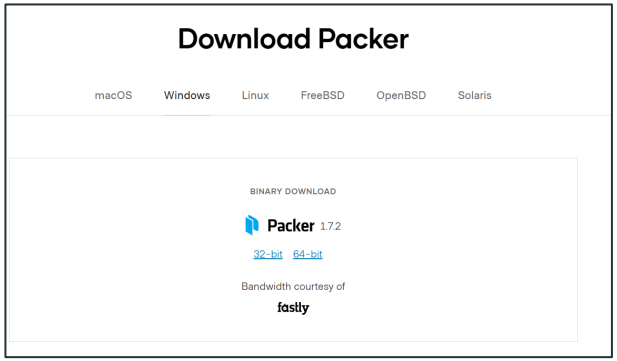
More information in the Writing Packer Templates Section

More information in this Section and throughout the course

Install HashiCorp Packer

So where do I download Packer?

- packer.io
- releases.hashicorp.com/packer



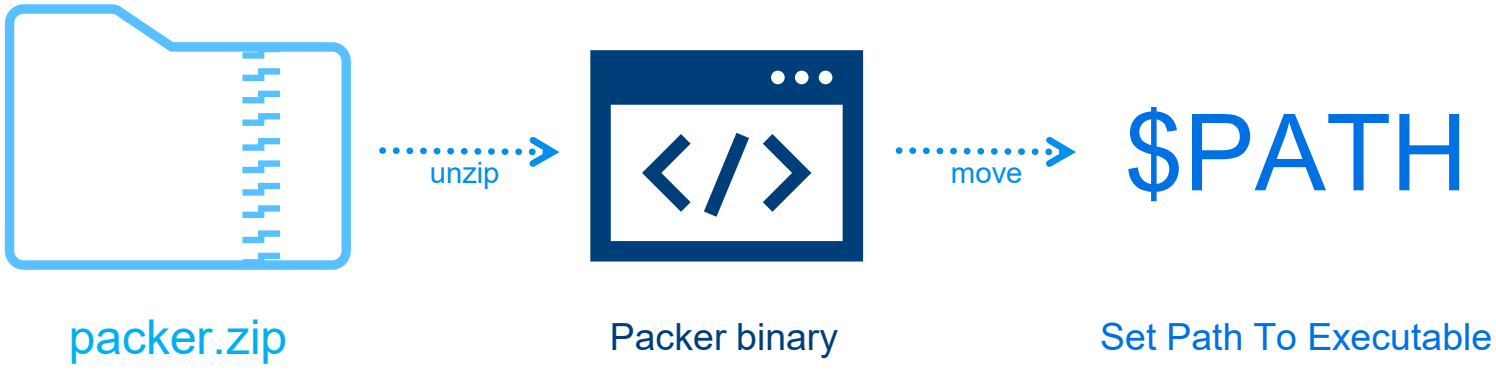
You can also download/install Packer using your preferred package manager (apt, yum, homebrew, chocolatey)

Terminal

```
$ curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
$ sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"
$ sudo apt-get update && sudo apt-get install packer
```

Install HashiCorp Packer

Manual Install



Packer Command Line

The command line interface (CLI) is how users/applications interact with Packer

- There is no UI or API for Packer

Terminal

```
$ packer
Usage: packer [--version] [--help] <command> [<args>]

Available commands are:
    build           build image(s) from template
    console         creates a console for testing variable interpolation
    fix             fixes templates from old versions of packer
    hcl2_upgrade    transform a JSON template into an HCL2 configuration
    inspect         see components of a template
    validate        check that a template is valid
    version         Prints the Packer version
```

Packer Command Line

`packer autocomplete`

Install Packer autocomplete to enable tab completion when using the CLI:

Terminal

```
$ packer -autocomplete-install

$ packer p
plugin  build

$ packer build -
-color              -debug              -except             -force
-machine-readable   -on-error           -only               -parallel
-timestamp           -var                -var-file
```

Type p and hit <tab>

Packer Command Line

Similar to any other command-line based tool, Packer uses a subcommand and additional arguments to execute Packer functionality

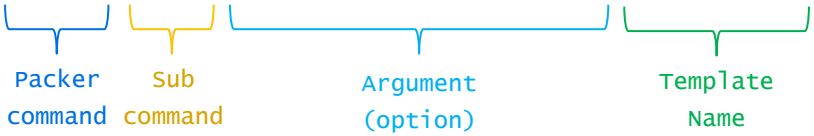
Terminal

```
$ packer <sub-command> <argument> <argument>
```

All commands start with the '`packer`' command

Terminal

```
$ packer build -var-file=vars.pkr.hcl aws.pkr.hcl
```



Packer Command Line

Subcommands available in Packer:

- `build` - build image(s) from template
- `console` - creates a console for testing variable interpolation
- `fix` - fixes templates from old versions of packer
- `hcl2_upgrade` - transform a JSON template into an HCL2 configuration
- `inspect` - see components of a template
- `validate` - check that a template is valid
- `version` - prints the Packer version

Most of the commands accept or require flags or arguments to execute the desired functionality

Packer Command Line

`packer build`

The `packer build` command takes a Packer template and runs all the defined builds to generate the desired artifacts. The `build` command provides the core functionality of Packer.

Terminal

```
$ packer build base-image.pkr.hcl
```

Important Options/Arguments

- -debug – enables debug mode for step-by-step troubleshooting
- -var – sets a variable in the Packer template
- -var-file – use a separate variable file

Packer Command Line

`packer fix`

The `packer fix` command takes a template and finds backwards incompatible parts of it and brings it up to date so it can be used with the latest version of Packer. Use after you update Packer to a new release version.

JSON ONLY COMMAND

- As of Packer 1.7.2, the command is not yet available with HCL2 templates yet, as it is not yet needed.

Terminal

```
$ packer fix old-template.json > new-template.json
```

Packer Command Line

`packer fmt`

The `packer fmt` command is used to format your Packer templates and files to the preferred HCL canonical format and style.

Terminal

```
$ packer fmt base-image.pkr.hcl
```

Not Formatted

base-image.pkr.hcl

```
variable "vpc_id" {
type=string
default="vpc-06626bb552084b94b"
}
```

packer fmt

Formatted Correctly

base-image.pkr.hcl

```
variable "vpc_id" {
  type    = string
  default = "vpc-06626bb552084b94b"
}
```

Packer Command Line

`packer inspect`

The `packer inspect` shows all components of a Packer template including variables, builds, sources, provisioners and post-processsors

Terminal

```
$ packer inspect aws-ubuntu.pkr.hcl
Packer Inspect: HCL2 mode

> input-variables:
> local-variables:
> builds:
  > Amazon AMI:
    sources:
      amazon-ebs.ubuntu
    provisioners:
      shell
    post-processors:
      0:
        manifest
```

Packer Command Line

`packer validate`

The `packer validate` will validate the syntax and the configuration of your Packer template. This is your first validation for templates after writing or updating them.

Terminal

```
$ packer validate temp.pkr.hcl
```

Template validated
(no error message)

Terminal

```
$ packer validate vmware-image.pkr.hcl
Template validation failed. Errors are shown below.

Errors validating build 'vmware'. 1 error(s) occurred:

* Either a path or inline script must be specified.
```

Template did not validate
successfully
(error shown)

Packer Command Line

`packer hcl2_upgrade`

The `packer hcl2_upgrade` translates a template written in the older JSON format to the new HCL2 format.

Terminal

```
$ packer hcl2_upgrade base-image.json

Successfully created base-image.json.pkr.hcl
```

Converted JSON to HCL2 format

base-image.json

```
{
  "variables": {
    "deployment_version": "version-1",
    "ami_prefix": "amzn2",
    "app_name": "consul",
    "consul_version": "1.9.3",
  },
```

base-image.json.pkr.hcl

```
variable "deployment_version" {
  type    = string
  default = "version-1"
}
variable "ami_prefix" {
  type    = string
  default = "amzn2"
}
variable "app_name" {
  type    = string
  default = "consul"
}
```

Packer Command Line

Environment Variables

Packer has a few environment variables that you should know:

- PACKER_LOG – enable Packer detailed logs (off by default)
- PACKER_LOG_PATH – set the path for Packer logs to specific file (rather than stderr)
- PKR_VAR_xxx – define a variable value using ENV rather than in a template

Terminal

```
$ export PACKER_LOG=1
$ export PACKER_LOG_PATH=/var/log/packer.log
$ packer build base-image.pkr.hcl
```

1. Enable Detailed Logs
2. Set a path for logs
3. Run the `packer build`

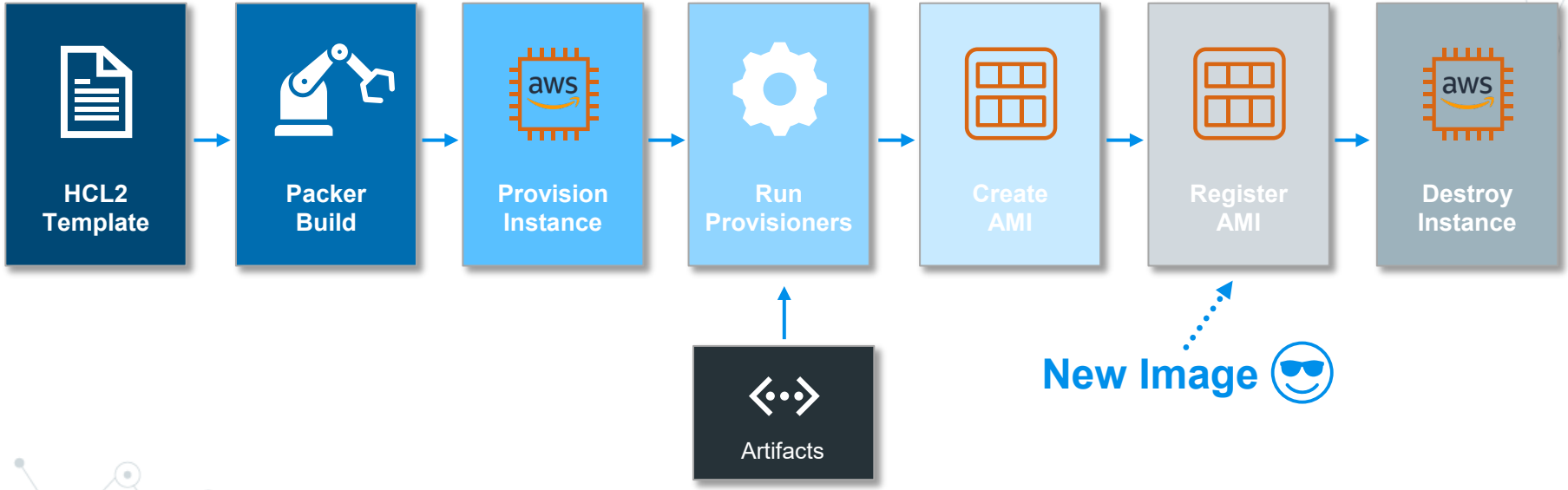
Terminal

```
$ export PKR_VAR_aws_region=us-east-1
$ packer build aws-base-image.pkr.hcl
…
```

Declare a value for the aws_region variable using ENV

The Packer Workflow

AWS Example



Introduction to HashiCorp Packer

Section Recap

- What is Packer?
- Primary Use Cases
- Core Components
- Installing Packer
- Interacting with Packer (CLI)
- Packer Workflow

END OF SECTION