Contents

Module: Packer

- What is it?
- why is it needed?
- Terminology
- Configuring
- how to use it?

What is it?

- opensource machine image creation tool
  - os + software
- automate image creation via config template
- NOT a configuration mgmt tool
- uses config mgmt tools configure image
  - chef, puppet, ansible
- Build Images across platforms and OSes
- image formats change for different platforms
  - AMIs, VMDX/VMX, OVF
- JSON or HCL2 (Preferred) templates

What....

- Packer is lightweight,
- Runs on every major operating system,
- Highly performant
- Multiple platforms in parallel.

A *machine image* is a single static unit that contains a pre-configured operating system and installed software which is used to quickly create new running machines.

Why is it needed?

- machine (golden) image creation
  - used across environments/departments
  - deploy to
    - Production/QA/Staging
    - AWS/VMWare/OpenStack/Azure
- Templates can be reused - no need to create from scratch
- Monthly VM Patching
  - Imaging Patching Pipeline
- Immutable Infrastructure
  - deployable artifact
- superfast infrastructure deployment - automate!!!
  - launch/provision machines in seconds/minutes
- **Identical Images across Platforms**

(Other) Use-Cases

- Continuous Delivery
- Dev/Prod Parity
- Appliance/Demo Creation

Terminology

- `Artifacts` are the results of a single build,
  - Usually a set of IDs or files to represent a machine image.
  - Every builder produces a single artifact.
    - Amazon EC2 builder artifact is a set of AMI IDs (one per region).
    - VMware builder, the artifact is a directory of files comprising the created virtual machine.
- `Builds` are a single task that eventually produces
  - an image for a single platform.
  - Multiple builds run in parallel.
    - "The Packer build produced an AMI to run our web application."
    - Or:
    - "Packer is running the builds now for VMware, AWS, and VirtualBox."

Terminology....

- `Builders` are components of Packer that are able to
  - create a machine image for a single platform.
  - Builders read in some configuration and generate a machine image.
  - A builder is invoked as part of a build
    - Example builders include VirtualBox, VMware, and Amazon EC2.
- `Commands` are sub-commands for the `packer` program that perform some job.
  - An example command is "build",

    - which is invoked as `packer build`.
  - Packer ships with a set of commands out of the box.

Terminology...

- `Data Sources` are components of Packer that
  - fetch data from outside Packer
  - make it available to use within the template.
    - Examples of data sources Amazon AMI, and Amazon Secrets Manager.
- `Post-processors` are components of Packer that
  - take the result of a builder or another post-processor and
  - process that to create a new artifact.
    - Examples of post-processors are
      - compress to compress artifacts,
      - upload to upload artifacts, etc.

Terminology

- `Provisioners` are components of Packer that
  - install and configure software within a running machine
  - runs prior to that machine being turned into a static image.
  - They perform the major work of making the image contain useful software.
    - Example provisioners include shell scripts, Chef, Puppet, etc.
- `Templates` are either HCL or JSON files which
  - define one or more builds by configuring the various components of Packer.
  - Packer is able to read a template and use that information to create multiple machine images in parallel.

show template file

Template - HCL2

- Template

```
packer {
  required_plugins {
    # PLUGIN
    docker = {
      version = ">= 0.0.7"
      source = "github.com/hashicorp/docker"
    }
  }
}

# VARIABLE
variable "docker_image" {
  type    = string
  default = "ubuntu:xenial"
}

# SOURCE
source "docker" "ubuntu" {
  image  = var.docker_image
```

```
  commit = true
}

# BUILD
build {
  name = "learn-packer"
  sources = [
    "source.docker.ubuntu",
  ]
  # PROVISIONER
  provisioner "shell" {
    inline = ["echo Running ${var.docker_image} Docker image."]
  }
  # POST PROCESSOR
  post-processor "docker-tag" {
    repository = "localhost/ubuntu"
    only = ["docker.ubuntu"]
  }

}
```

How to use it - Setup

Install Methods

Using Linux on RHEL/CentOS install method

- Setup Yum (dnf also works)
  `sudo yum install -y yum-utils`
- Add the Repo
  `sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo`
- Install Packer
  `sudo yum -y install packer`
- setup the path
  `export PATH=/usr/bin:$PATH`

Configuring Packer

There are a few configuration settings that affect Packer globally by configuring the core of Packer.

These settings all have reasonable defaults,so you generally don't have to worry about it until you want to tweak a configuration.

If you're just getting started with Packer, don't worry about core configuration for now.

Packer's home directory

Plugins and core configuration files can exist in the home directory of Packer.

The home directory of Packer will be the first one of the following env values to be set :

unix	windows
`${PACKER_CONFIG_DIR}`	`%PACKER_CONFIG_DIR%`
`${APPDATA}`	`%APPDATA%`
`${HOME}`	`%HOME%`
user dir of `${USER}`	user dir of `${USER}`

Note: On this page "Packer's home directory" will be referenced as `PACKER_HOME_DIR`.

Packer's config file

Packer can optionally read a JSON file for the end user to set core settings. The config file of Packer will be looked up on the following paths:

unix	windows
`${PACKER_CONFIG}`	`%PACKER_CONFIG%`
`PACKER_HOME_DIR/.packerconfig`	`PACKER_HOME_DIR/packer.config/`

It is not an error if no config file was found.

Packer's config directory

Packer's configuration directory can potentially contain plugins and internal Packer files. The config dir of Packer will be looked up on the following paths:

unix	windows
`PACKER_HOME_DIR/.packer.d`	`PACKER_HOME_DIR/packer.d/`

Examples:

- On a 'unix' system, if the `$PACKER_CONFIG_DIR` env var is set to `/home/packer`, the config directory will be: `/home/packer/.packer.d/` and other values will not be checked.
- On a 'unix' system, if the `HOME` env var is `/home/azr` or the `USER` env var is `azr`, then the config directory will default to `/home/azr/.packer.d/`.
- On a 'windows' system, if the `PACKER_CONFIG_DIR` env var is set to `C:/`,the config directory will be: `C:/packer.d/` and other values will not be checked.

Full list of Environment Variables usable for Packer

Packer uses a variety of environmental variables. A listing and description of each can be found below:

- `PACKER_CACHE_DIR` - The location of the Packer cache. This defaults to `./packer_cache/`. Relative paths can be used. Some plugins can cache large files like ISOs in the cache dir.
- `PACKER_CONFIG` - The location of the core configuration file. The format of the configuration file is basic JSON.
- `PACKER_CONFIG_DIR` - The location for the home directory of Packer.
- `PACKER_GITHUB_API_TOKEN` - When using Packer init on HCL2 templates, Packer queries the public API from Github which limits the amount of queries on can set the `PACKER_GITHUB_API_TOKEN` with a Github Token to make it higher.
- `PACKER_LOG` - Setting this to any value other than "" (empty string) or "0" will enable the logger.
- `PACKER_LOG_PATH` - The location of the log file. Note: `PACKER_LOG` must be set for any logging to occur.

- `PACKER_NO_COLOR` - Setting this to any value will disable color in the terminal.
- `PACKER_PLUGIN_MAX_PORT` - The maximum port that Packer uses for communication with plugins, since plugin communication happens over TCP connections on your local host. The default is 25,000. This can also be set using the Packer's config file.

- `PACKER_PLUGIN_MIN_PORT` - The minimum port that Packer uses for communication with plugins, since plugin communication happens over TCP connections on your local host. The default is 10,000. This can also be set using the Packer's config file.

- `PACKER_PLUGIN_PATH` - a PATH variable for finding third-party packer plugins. For example: `~/custom-dir-1:~/custom-dir-2`. Separate directories in the PATH string using a colon (`:`) on posix systems and a semicolon (`;`) on windows systems. The above example path would be able to find a provisioner named `packer-provisioner-foo` in either `~/custom-dir-1/packer-provisioner-foo` or `~/custom-dir-2/packer-provisioner-foo`.
- `CHECKPOINT_DISABLE` - When Packer is invoked it sometimes calls out to checkpoint.hashicorp.com to look for new versions of Packer. If you want to disable this for security or privacy reasons, you can set this environment variable to `1`.
- `TMPDIR` (Unix) / `TMP TEMP USERPROFILE` (Windows) - The location of the directory used for temporary files (defaults to `/tmp` on Linux/Unix and `%USERPROFILE%\AppData\Local\Temp` on Windows Vista and above). It might be necessary to customize it when working with large files since `/tmp` is a memory-backed filesystem in some Linux distributions in which case `/var/tmp` might be preferred.

Build first image - 1 - parallel build (duplicated build lines!!!)

- create a working directory
  ```
  mkdir packer_tutorial
  cd packer_tutorial
  ```
- create a packer manifest - docker-ubuntu.pkr.hcl

```
packer {
  required_plugins {
    docker = {
      version = ">= 0.0.7"
      source = "github.com/hashicorp/docker"
        # if you using podman either comment out previous line
        # AND
        # uncomment this next line instead
        #source  = "github.com/Polpetta/podman"
      #OR
        # leave source as docker
        # AND
        # execute the following command as root
        #sudo ln -s /usr/bin/podman /usr/bin/docker
    }
  }
}

source "docker" "ubuntu-xenial" {
  image  = "ubuntu:xenial"
  commit = true
}

source "docker" "ubuntu-zesty" {
  image  = "ubuntu:zesty"
  commit = true
}

build {
  name = "learn-packer-xenial"
  sources = [
    "source.docker.ubuntu-xenial"
  ]
  provisioner "shell" {
    environment_vars = [
```

```
      "FOO=hello world from xenial",
    ]
    inline = [
      "echo Adding file to Docker Container",
      "echo \"FOO is $FOO\" > example.txt",
    ]
  }
}
#DUPLICATE BUILD SECTION
  build {
  name = "learn-packer-zesty"
  sources = [
    "source.docker.ubuntu-zesty"
  ]
  provisioner "shell" {
    environment_vars = [
      "FOO=hello world from zesty",
    ]
    inline = [
      "echo Adding file to Docker Container",
      "echo \"FOO is $FOO\" > example.txt",
    ]
  }
}
```

---

build first image - 2

- check format
  `packer fmt .`
- validate file
  `packer validate .`
- build
  `packer build docker-ubuntu.pkr.hcl`
  you will see some output
- check what is create
  ```
  # get the image id - should be one just created
  docker images
  # or
  # podman images
  ```
  the image at the top of the list is the one

Check the output

- check the container has the required file - do for each image created
  ```
  # connect to it and check the file and contents are there
  docker run -it image_id
    cat example.txt
    cat /etc/os-release
  ```

Parallel Build - tagging, tidy-up

- removed build section, added some variables and post-processor section - docker-ubuntu2.pkr.hcl

```
packer {
  required_plugins {
    docker = {
      version = ">= 0.0.7"
      source = "github.com/hashicorp/docker"
        # if you using podman either comment out previous line
        # AND
        # uncomment this next line instead
        #source  = "github.com/Polpetta/podman"
      #OR
        # leave source as docker
        # AND
        # execute the following command as root
        #sudo ln -s /usr/bin/podman /usr/bin/docker
    }
  }
}

variable "docker_image" {
  type    = string
  default = "ubuntu:xenial"
}

variable "docker_image_zesty" {
  type    = string
  default = "ubuntu:zesty"
}

source "docker" "ubuntu" {
  image  = var.docker_image
  commit = true
}

source "docker" "ubuntu-zesty" {
  image  = var.docker_image_zesty
  commit = true
}

build {
  name = "learn-packer"
  sources = [
    # Added source into this build
    "source.docker.ubuntu",
    "source.docker.ubuntu-zesty"
  ]

  provisioner "shell" {
    environment_vars = [
      "FOO=hello world.",
    ]
    inline = [
      "echo Adding file to Docker Container",
      "echo \"FOO is $FOO\" > example.txt",
    ]
  }
  provisioner "shell" {
    inline = ["echo Running ${var.docker_image} Docker image."]
  }

  post-processor "docker-tag" {
```

```
    repository = "localhost/ubuntu"
    only = ["docker.ubuntu"]
  }

  post-processor "docker-tag" {
    repository = "localhost/ubuntu-zesty"
    only   = ["docker.ubuntu-zesty"]
  }
}
```

Overriding variables and useful commands

- using the same template but using **ubuntu:focal** variable
  `packer build -var 'docker_image=ubuntu:focal' docker-ubuntu2.pkr.hcl`
- validate, format and hcl2_upgrade
  ```
  # packer --help
  packer fmt docker-ubuntu2.pkr.hcl
  packer validate docker-ubuntu2.pkr.hcl
  # upgrade a json template to HCL - YAH!
  packer hcl2_upgrade test.json
  # see components of template
  packer inspect docker-ubuntu.pkr.hcl
  ```
