DOCS

Configuration Templates (/docs/templates/configuration-templates.html)

User Variables (/docs/templates/user-variables.html)

Veewee-to-Packer (/docs/templates/veewee-to-packer.html)

BUILDERS

Amazon EC2 (AMI) (/docs/builders/amazon.html)

DigitalOcean (/docs/builders/digitalocean.html)

Docker (/docs/builders/docker.html)

Google Compute Engine (/docs/builders/googlecompute.html)

Null (/docs/builders/null.html)

OpenStack (/docs/builders/openstack.html)

Parallels (/docs/builders/parallels.html)

QEMU (/docs/builders/qemu.html)

VirtualBox (/docs/builders/virtualbox.html)

VMware (/docs/builders/vmware.html)

Custom (/docs/builders/custom.html)

PROVISIONERS

Shell Scripts (/docs/provisioners/shell.html)

File Uploads (/docs/provisioners/file.html)

Ansible (/docs/provisioners/ansible-local.html)

Chef Client (/docs/provisioners/chef-client.html)

Chef Solo (/docs/provisioners/chef-solo.html)

Puppet Masterless (/docs/provisioners/puppet-masterless.html)

Puppet Server (/docs/provisioners/puppet-server.html)

Salt (/docs/provisioners/salt-masterless.html)

Custom (/docs/provisioners/custom.html)

TEMPLATES: PROVISIONERS

Within the template, the provisioners section contains an array of all the provisioners that Packer should use to install and configure software within running machines prior to turning them into machine images.

Provisioners are *optional*. If no provisioners are defined within a template, then no software other than the defaults will be installed within the resulting machine images. This is not typical, however, since much of the value of Packer is to produce multiple identical images of pre-configured software.

This documentation page will cover how to configure a provisioner in a template. The specific configuration options available for each provisioner, however, must be referenced from the documentation for that specific provisioner.

Within a template, a section of provisioner definitions looks like this:

```
{
  "provisioners": [
    // ... one or more provisioner definitions here
  ]
}
```

For each of the definitions, Packer will run the provisioner for each of the configured builds. The provisioners will be run in the order they are defined within the template.

Provisioner Definition

A provisioner definition is a JSON object that must contain at least the `type` key. This key specifies the name of the provisioner to use. Additional keys within the object are used to configure the provisioner, with the exception of a handful of special keys, covered later.

As an example, the "shell" provisioner requires a key such as `script` which specifies a path to a shell script to execute within the machines being created.

An example provisioner definition is shown below, configuring the shell provisioner to run a local script within the machines:

```
{
  "type": "shell",
  "script": "script.sh"
}
```

Run On Specific Builds

You can use the `only` or `except` configurations to run a provisioner only with specific builds. These two configurations do what you expect: `only` will only run the provisioner on the specified builds and `except` will run the provisioner on anything other than the specified builds.

An example of `only` being used is shown below, but the usage of `except` is effectively the same:

```
{
  "type": "shell",
  "script": "script.sh",
  "only": ["virtualbox-iso"]
}
```

The values within `only` or `except` are *build names*, not builder types. If you recall, build names by default are just their builder type, but if you specify a custom `name` parameter, then you should use that as the value instead of the type.

Build-Specific Overrides

While the goal of Packer is to produce identical machine images, it sometimes requires periods of time where the machines are different before they eventually converge to be identical. In these cases, different configurations for provisioners may be necessary depending on the build. This can be done using build-specific overrides.

An example of where this might be necessary is when building both an EC2 AMI and a VMware machine. The source EC2 AMI may setup a user with administrative privileges by default, whereas the VMware machine doesn't have these privileges. In this case, the shell script may need to be executed differently. Of course, the goal is that hopefully the shell script converges these two images to be identical. However, they may initially need to be run differently.

This example is shown below:

```
{
  "type": "shell",
  "script": "script.sh",

  "override": {
    "vmware-iso": {
      "execute_command": "echo 'password' | sudo -S bash {{.Path}}"
    }
  }
}
```

As you can see, the `override` key is used. The value of this key is another JSON object where the key is the name of a builder definition (/docs/templates/builders.html). The value of this is in turn another JSON object. This JSON object simply contains the provisioner configuration as normal. This configuration is merged into the default provisioner configuration.

Pausing Before Running

With certain provisioners it is sometimes desirable to pause for some period of time before running it. Specifically, in cases where a provisioner reboots the machine, you may want to wait for some period of time before starting the next provisioner.

Every provisioner definition in a Packer template can take a special configuration `pause_before` that is the amount of time to pause before running that provisioner. By default, there is no pause. An example is shown below:

```
{
  "type": "shell",
  "script": "script.sh",
  "pause_before": "10s"
}
```

For the above provisioner, Packer will wait 10 seconds before uploading and executing the shell script.