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: POST-PROCESSORS

The post-processor section within a template configures any post-processing that will be done to images built by the builders. Examples of post-processing would be compressing files, uploading artifacts, etc.

Post-processors are *optional*. If no post-processors are defined within a template, then no post-processing will be done to the image. The resulting artifact of a build is just the image outputted by the builder.

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

Within a template, a section of post-processor definitions looks like this:

```
{
  "post-processors": [
    // ... one or more post-processor definitions here
  ]
}
```

For each post-processor definition, Packer will take the result of each of the defined builders and send it through the post-processors. This means that if you have one post-processor defined and two builders defined in a template, the post-processor will run twice (once for each builder), by default. There are ways, which will be covered later, to control what builders post-processors apply to, if you wish.

Post-Processor Definition

Within the `post-processors` array in a template, there are three ways to define a post-processor. There are *simple* definitions, *detailed* definitions, and *sequence* definitions. Don't worry, they're all very easy to understand, and the "simple" and "detailed" definitions are simply shortcuts for the "sequence" definition.

A **simple definition** is just a string; the name of the post-processor. An example is shown below. Simple definitions are used when no additional configuration is needed for the post-processor.

```
{
  "post-processors": ["compress"]
}
```

A **detailed definition** is a JSON object. It is very similar to a builder or provisioner definition. It contains a `type` field to denote the type of the post-processor, but may also contain additional configuration for the post-processor. A detailed definition is used when additional configuration is needed beyond simply the type for the post-processor. An example is shown below.

```
{
  "post-processors": [
    {
      "type": "compress",
      "format": "tar.gz"
    }
  ]
}
```

A **sequence definition** is a JSON array comprised of other **simple** or **detailed** definitions. The post-processors defined in the array are run in order, with the artifact of each feeding into the next, and any intermediary artifacts being discarded. A sequence definition may not contain another sequence definition. Sequence definitions are used to chain together multiple post-processors. An example is shown below, where the artifact of a build is compressed then uploaded, but the compressed result is not kept.

```
{
  "post-processors": [
    [
      "compress",
      { "type": "upload", "endpoint": "http://example.com" }
    ]
  ]
}
```

As you may be able to imagine, the **simple** and **detailed** definitions are simply shortcuts for a **sequence** definition of only one element.

Input Artifacts

When using post-processors, the input artifact (coming from a builder or another post-processor) is discarded by default after the post-processor runs. This is because generally, you don't want the intermediary artifacts on the way to the final artifact created.

In some cases, however, you may want to keep the intermediary artifacts. You can tell Packer to keep these artifacts by setting the `keep_input_artifact` configuration to `true`. An example is shown below:

```
{
  "post-processors": [
    {
      "type": "compress",
      "keep_input_artifact": true
    }
  ]
}
```

This setting will only keep the input artifact to *that specific* post-processor. If you're specifying a sequence of post-processors, then all intermediaries are discarded by default except for the input artifacts to post-processors that explicitly state to keep the input artifact.

Note: The intuitive reader may be wondering what happens if multiple post-processors are specified (not in a sequence). Does Packer require the configuration to keep the input artifact on all the post-processors? The answer is no, of course not. Packer is smart enough to figure out that at least one post-processor requested that the input be kept, so it will keep it around.

Run On Specific Builds

You can use the `only` or `except` configurations to run a post-processor only with specific builds. These two configurations do what you expect: `only` will only run the post-processor on the specified builds and `except` will run the post-processor 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. `only` and `except` can only be specified on "detailed" configurations. If you have a sequence of post-processors to run, `only` and `except` will only affect that single post-processor in the sequence.

```
{
  "type": "vagrant",
  "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.