

# An Introduction To Packer



# Contents

- Chapter 1. Packer Overview.....1**
  - Introduction to Packer..... 1
  - What is Packer?.....1
- Chapter 2. Getting Started.....2**
  - Installing Chocolately..... 2
  - Installing Packer on Windows..... 3
  - Creating a Packer Template.....3
  - Building a Packer Image..... 5
- Chapter 3. Packer Commands (CLI)..... 7**
  - Packer Commands.....7
    - build command.....7
    - console command..... 7
    - fix command.....8
    - inspect command..... 8
    - validate command.....8
    - version command..... 8

# Chapter 1. Packer Overview

## Introduction to Packer

Welcome to the world of Packer! This introduction guide will show you what Packer is, explain why it exists, the benefits it has to offer, and how you can get started with it. If you're already familiar with Packer, the documentation provides more of a reference for all available features.

## What is Packer?

*Packer is an open source tool for creating identical machine images for multiple platforms from a single source configuration.*

Packer is lightweight, runs on every major operating system, and is highly performant, creating machine images for multiple platforms in parallel.

Advantages of using Packer include:

- Super fast infrastructure deployment
- Improved stability
- Multi-provider portability
- Greater testability

# Chapter 2. Getting Started

## Installing Chocolatey

1. Download Chocolatey from the [Chocolatey](https://chocolatey.org) website
2. Ensure that you are using an administrative shell to install Chocolatey.
3. Install with powershell.exe by pasting the following command into your shell and press Enter:

```
Set-ExecutionPolicy Bypass -Scope Process -Force;

[System.Net.ServicePointManager]::SecurityProtocol =

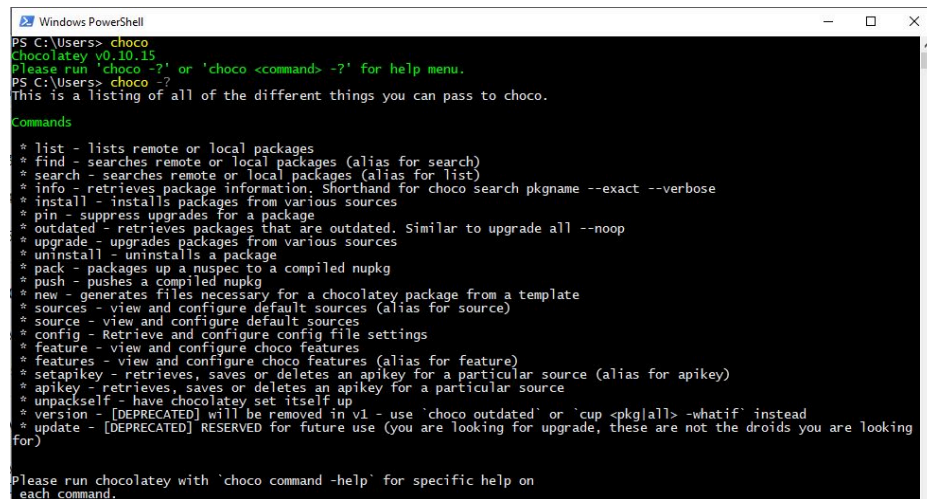
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072;

iex ((New-Object

System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

4. Wait a few seconds for the command to complete.
5. If you don't see any errors, you are ready to use Chocolatey.

You can confirm the installation of Chocolatey by typing `choco` or `choco -?:`



```
Windows PowerShell
PS C:\Users> choco
Chocolatey v0.10.15
Please run 'choco -?' or 'choco <command> -?' for help menu.
PS C:\Users> choco -?
This is a listing of all of the different things you can pass to choco.

Commands
* list - lists remote or local packages
* find - searches remote or local packages (alias for search)
* search - searches remote or local packages (alias for list)
* info - retrieves package information. Shorthand for choco search pkgname --exact --verbose
* install - installs packages from various sources
* pin - suppress upgrades for a package
* outdated - retrieves packages that are outdated. Similar to upgrade all --noop
* upgrade - upgrades packages from various sources
* uninstall - uninstalls a package
* pack - packages up a nuspec to a compiled nupkg
* push - pushes a compiled nupkg
* new - generates files necessary for a chocolatey package from a template
* sources - view and configure default sources (alias for source)
* source - view and configure default sources
* config - Retrieve and configure config file settings
* feature - view and configure choco features
* features - view and configure choco features (alias for feature)
* setapikey - retrieves, saves or deletes an apikey for a particular source (alias for apikey)
* apikey - retrieves, saves or deletes an apikey for a particular source
* unpackself - have chocolatey set itself up
* version - [DEPRECATED] will be removed in v1 - use 'choco outdated' or 'cup <pkg>all' -whatif instead
* update - [DEPRECATED] RESERVED for future use (you are looking for upgrade, these are not the droids you are looking for)

Please run chocolatey with 'choco command -help' for specific help on each command.
```

## Installing Packer on Windows

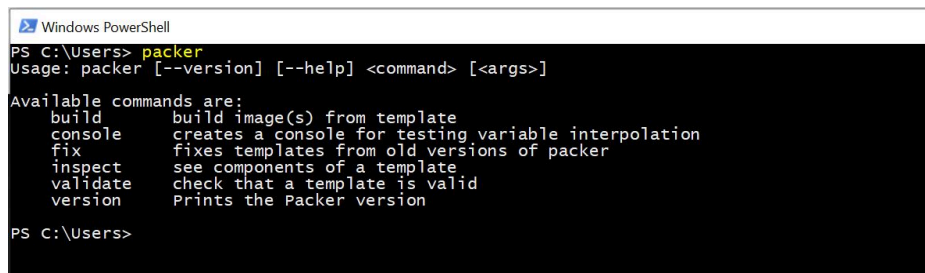
1. You can install Packer by running `choco install`.



```
Windows PowerShell
PS C:\Users> choco install packer
```

2. After installing Packer, you can verify the installation by opening a new command prompt and typing `packer` and then pressing enter.

You should see the following if packer was installed successfully:



```
Windows PowerShell
PS C:\Users> 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
  inspect    see components of a template
  validate   check that a template is valid
  version    Prints the Packer version

PS C:\Users>
```

## Creating a Packer Template

To create a Packer image for an application, you must create a Packer template. The template can live either within a local directory of the application or in a separate empty directory.

The configuration file determines the type of image being built and is written in JSON. For example, this file may be called `aws-application.json`.

1. Add variables to your Packer template

**Note:** The contents of the file begin with your AWS environment variables for your account, followed by the builders and provisioners. Variables are particularly useful when it comes to sensitive information, such as your account login and SSH key fingerprint.

```
{
  "variables": {
    "aws_access_key": "",
    "aws_secret_key": ""
  },
}
```

## 2. Add packer builders

Builders are responsible for creating machines and generating images from them for various platforms. For example, there are separate builders for EC2, VMware, VirtualBox, etc. Packer comes with many builders by default, and can also be extended to add new builders. Below is an example of an AWS builder for a web-based application using Nginx.

```
"builders": [  
  {  
    "type": "amazon-instance",  
    "access_key": "YOUR KEY HERE",  
    "secret_key": "YOUR SECRET KEY HERE",  
    "region": "us-east-1",  
    "source_ami": "ami-d9d6a6b0",  
    "instance_type": "m1.small",  
    "ssh_username": "ubuntu",  
  
    "account_id": "0123-4567-0890",  
    "s3_bucket": "packer-images",  
    "x509_cert_path": "x509.cert",  
    "x509_key_path": "x509.key",  
    "x509_upload_path": "/tmp",  
  
    "ami_name": "packer-quick-start {{timestamp}}"  
  }  
],
```

## 3. Add Provisioners to Customize the Image

Provisioners use builtin and third-party software to install and configure the machine image after booting. Provisioners prepare the system for use, so common use cases for provisioners include:

- installing packages
- patching the kernel
- creating users
- downloading application code

The `file` provisioner uploads files to machines built by Packer. The recommended usage of the `file` provisioner is to use it to upload files, and then use `shell` provisioner to move them to the proper place, set permissions, etc. The `file` provisioner can refer to a single file, such as `index.html`, or an entire directory, such as `directory/`.

```

"provisioners": [
  {
    "type": "file",
    "source": "index.html",
    "destination": "/usr/share/nginx/html/"
  },
  {
    "type": "file",
    "source": "directory-name/",
    "destination": "/usr/share/nginx/html/directory-name/" }
]
}

```

The `shell` provisioner provisions machines built by Packer using shell scripts. Shell provisioning is the easiest way to get software installed and configured on a machine.

```

"provisioners": [
  {
    "type": "shell",
    "script": "create-directories.sh"
  }
]
}

```

## Building a Packer Image

Once your template is complete, you can proceed to build the application image.


1. Validate the Packer template and ensure that the JSON syntax and configuration values are correct.

```

$ packer validate aws-application.json

Template validated successfully.

```

 **Note:** If the template validation was not successful, that means there is an error in your configuration file. The `validate` command should tell you where to find the error.

2. To create your Packer image, execute `packer build` with the name of the template file. The output will look something like this:

```
$ packer build aws-application.json

aws output will be in this color.

==> aws: Waiting for source machine to become available...

==> aws: Waiting for SSH to become available...

==> aws: Connected to SSH!

==> aws: Provisioning with shell script: add-directories.sh

==> aws: Uploading index.html => /usr/share/nginx/html/

==> aws: Uploading css/ => /usr/share/nginx/html/css/

==> aws: Uploading js/ => /usr/share/nginx/html/js/

==> aws: Stopping source machine (6163c9e1-0ee6-eccb-c8bb-9f4369c73bb0)...

==> aws: Waiting for source machine to stop (6873c9e1-0ee8-eccb-c8bb-9f4376c73bb0)...

==> aws: Creating image from source machine...

==> aws: Waiting for image to become available...

==> aws: Deleting source machine...

==> aws: Waiting for source machine to be deleted...

Build 'aws' finished.

==> Builds finished. The artifacts of successful builds are:

--> aws: Image was created: c7da3349-5b2d-2fc7-bde5-503a3f8450b1
```

**Artifacts** are the result of a single build, including a set of IDs for files that represent the final machine image. Every builder produces a single artifact. For the AWS builder, the artifact is the new image ID.



# Chapter 3. Packer Commands (CLI)

## Packer Commands

### *Overview*

Packer is controlled using a command-line interface. All interaction with Packer is done via the `packer tool`. Like many other command-line tools, the `packer` tool takes a subcommand to execute, and that subcommand may have additional options as well. Subcommands are executed with `packer SUBCOMMAND`, where "SUBCOMMAND" is the actual command you wish to execute.

If you run `packer` by itself, help will be displayed showing all available subcommands and a brief synopsis of what they do. In addition to this, you can run any `packer` command with the `-h` flag to output more detailed help for a specific subcommand.

### `build` command

`packer build`

#### **DESCRIPTION**

Build images from a template.

The `packer build` command takes a template and runs all the builds within it in order to generate a set of artifacts. The various builds specified within a template are executed in parallel, unless otherwise specified. And the artifacts that are created will be outputted at the end of the build.

### `console` command

`packer console`

#### **DESCRIPTION**

Creates a console for testing variable interpolation.

The `packer console` command allows you to experiment with Packer variable interpolations. You may access variables in the Packer config you called the console with, or provide variables when you call console using the `-var` or `-var-file` command line options.

## `fix` command

`packer fix`

### DESCRIPTION

Fixes templates from old versions of packer.

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. After you update to a new Packer release, you should run the fix command to make sure your templates work with the new release.

## `inspect` command

`packer inspect`

### DESCRIPTION

See components of a template.

The `packer inspect` command takes a template and outputs the various components a template defines. This can help you quickly learn about a template without having to dive into the JSON itself. The command will tell you things like what variables a template accepts, the builders it defines, the provisioners it defines and the order they'll run, and more.

## `validate` command

`packer validate`

### DESCRIPTION

Checks that a template is valid.

The `packer validate` Packer command is used to validate the syntax and configuration of a template. The command will return a zero exit status on success, and a non-zero exit status on failure. Additionally, if a template doesn't validate, any error messages will be outputted.

## `version` command

`packer version`

## DESCRIPTION

Prints the Packer version.