JAMES TURNBULL

THE PACKER BOOK

The Packer Book

James Turnbull

April 20, 2018

Version: v1.1.2 (067741e)

Website: The Packer Book

ISBN 9780988820272

Contents

Chapter 1

First steps with Packer

Packer's purpose is building images—so we're going to start by building a basic image. Packer calls the process of creating an image a *build*. *Artifacts* are the output from builds. One of the more useful aspects of Packer is that you can run multiple builds to produce multiple artifacts.

A *build* is fed from a *template*. The template is a JSON document that describes the image we want to build—both where we want to build it and what the build needs to contain. Using JSON as a format strikes a good balance between readable and machine-generated.

NOTE New to JSON? Here's a good article to get you started. You can also find an online linter to help validate JSON syntax.

To determine what sort of image to build, Packer uses components called *builders*. A builder produces an image of a specific format—for example, an AMI builder or a Docker image builder. Packer ships with a number of builders, but as we'll discover in Chapter 7, you can also add your own builder in the form of plugins.

We're going to use Packer to build an Amazon Machine Image or AMI. AMIs underpin Amazon Web Services virtual machine instances run from the Elastic Compute Cloud or EC2. You'd generally be building your own AMIs to launch instances customized for your environment or specific purpose. We've chosen to focus on Cloud-based images in this book because they are easy to describe and work with while learning. Packer, though, is also able provision virtual machines.

As we're using AMIs here, before we can build our image, you'll need an Amazon Web Services account. Let's set that up now.

Setting up Amazon Web Services

We're going to use Amazon Web Services to build our initial image. Amazon Web Services have a series of free plans (Amazon calls them tiers) that you can use to test Packer at no charge. We'll use those free tiers in this chapter.

If you haven't already got a free AWS account, you can create one at:

`https://aws.amazon.com/`

Then follow the Getting Started process.

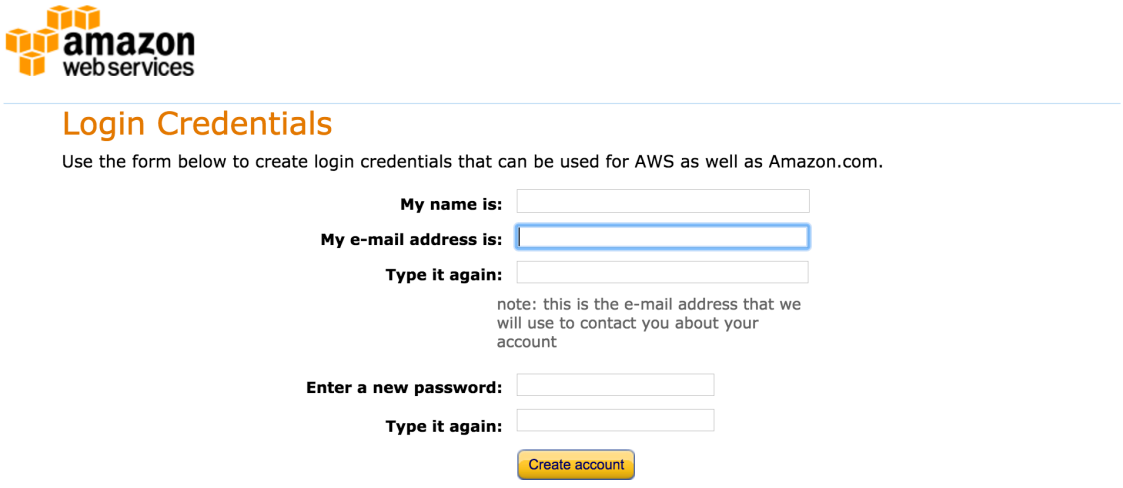

Figure 1.1: Creating an AWS account

As part of the *Getting Started* process you'll receive an access key ID and a secret access key. If you have an Amazon Web Services (AWS) account you should already have a pair of these. Get them ready. You'll use them shortly.

Alternatively, you should look at IAM or AWS Identity and Access Management. IAM allows multi-user role-based access control to AWS. It allows you to create access credentials per user and per AWS service.

Configuring it is outside the scope of this book, but here are some good places to learn more:

- IAM getting started guide.
- Root versus IAM credentials.
- Best practices for managing access keys.

We're going to use a `t2.micro` instance to create our first image. If your account is eligible for the free tier (and generally it will be) then this won't cost you anything.

WARNING There is a chance some of the examples in this book might cost you some money. Please be aware of your billing status and the state of your infrastructure.

Running Packer

Packer is contained in a single binary: `packer`. We installed that binary in the last chapter. Let's run it now and see what options are available to us.

Listing 1.1: The packer binary

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

Available commands are:
    build       build image(s) from template
    fix         fixes templates from old versions of packer
    inspect     see components of a template
    push        push a template and supporting files to a Packer
 build service
    validate    check that a template is valid
    version     Prints the Packer version
```

The `packer` binary builds images using the `build` sub-command and inputting a JSON file called a template.

Let's create a directory hold our Packer templates.

Listing 1.2: Creating a template directory

```
$ mkdir packer_templates
```

Now let's create an initial template to generate our initial AMI.

Creating an initial template

Let's create an empty template file to start.

Listing 1.3: Creating an empty template file

```
$ touch initial_ami.json
```

Now let's open an editor and populate our first template file. The template file defines the details of the image we want to create and some of the how of that creation. Let's see that now.

Listing 1.4: Our initial_ami.json file

```
{
  "variables": {
    "aws_access_key": "",
    "aws_secret_key": ""
  },
  "builders": [{
    "type": "amazon-ebs",
    "access_key": "{{user `aws_access_key`}}",
    "secret_key": "{{user `aws_secret_key`}}",
    "region": "us-east-1",
    "source_ami": "ami-a025aeb6",
    "instance_type": "t2.micro",
    "ssh_username": "ubuntu",
    "ami_name": "packer-example {{timestamp | clean_ami_name}}"
  }]
}
```

The template is a nested JSON hash with a series of blocks defined—in this case `variables` and `builders`. Let's take a look at both blocks and their contents.

Variables

The `variables` block contains any user-provided variables that Packer will use as part of the build. User variables are useful in these three ways:

- As shortcuts to values that you wish to use multiple times.
- As variables with a default value that can be overridden at build time.
- For keeping secrets or other values out of your templates.

User variables must be defined in the `variables` block. If you have no variables then you simply do not specify that block. Variables are key/value pairs in string form; more complex data structures are not present. These variables are translated into strings, numbers, booleans, and arrays when parsed into Packer. Packer assumes a list of strings separated by commas should be interpreted as an array.

NOTE Packer also has another type of variable, called template variables, that we'll see later in the book.

User variables can either have a specified value or be defined *null*—for example “”. If a variable is null then, for a template to be valid and executed, its value must be provided in some way when Packer runs.

In addition to defined values, variables also support environment variables.

Environment variables

A common configuration approach is to use environment variables to store configuration information. Inside the `variables` block we can retrieve the value of

an environment variable and use it as the value of a Packer variable. We do this using a function called `env`.

Functions allow operations on strings and values inside Packer templates. In this case the `env` function only works in the `variables` block when setting the default value of a variable. We can use it like so:

TIP You can find a full list of the available functions in the Packer engine documentation.

Listing 1.5: Environment variables

```
{
  "variables": {
    "aws_access_key": "{{env `AWS_ACCESS_KEY`}}",
    "aws_secret_key": "{{env `AWS_SECRET_KEY`}}"
  },
```

Note that the function and the environmental variable to be retrieved are enclosed in double braces: `{{ }}`. The specific environment variable to be retrieved is specified inside back ticks.

NOTE You can only use environment variables inside the `variables` block. This is to ensure a clean source of input for a Packer build.

Populating variables

If you're not setting a default value for a variable then variable values must be provided to Packer at runtime. There are several different ways these can be populated. The first is via the command line at build time using the `-var` flag.

Listing 1.6: Specifying a variable on the command line

```
$ packer build \
  -var 'aws_access_key=secret' \
  -var 'aws_secret_key=reallysecret' \
  initial_ami.json
```

You can specify the `-var` flag as many times as needed to specify variable values. If you attempt to define the same variable more than once, the last definition of the variable will stand.

You can also specify variable values in a JSON file. For example, you could create a file called `variables.json` in our `initial_ami` directory and populate it.

Listing 1.7: Specifying variable values in a file

```
{
  "aws_access_key": "secret",
  "aws_secret_key": "reallysecret"
}
```

You can then specify the `variables.json` file on the command line with the `-var -file` flag like so:

Listing 1.8: Specifying the variable file

```
$ packer build \
  -var-file=variables.json \
  initial_ami.json
```

You can define multiple files using multiple instances of the `-var-file` flag. Like variables specified on the command line, if a variable is defined more than once then the last definition of the variable stands.

Builders

The next block is the engine of the template, the `builders` block. The builders turn our template into a machine and then into an image. For our Amazon AMI image we're using the `amazon-ebs` builder. There are a variety of Amazon AMI builders; the `amazon-ebs` builder creates AMI images backed by EBS volumes.

TIP See this storage reference page for more details.

Listing 1.9: The initial_ami builders block

```
"builders": [{
  "type": "amazon-ebs",
  "access_key": "{{user `aws_access_key`}}",
  "secret_key": "{{user `aws_secret_key`}}",
  "region": "us-east-1",
  "source_ami": "ami-a025aeb6",
  "instance_type": "t2.micro",
  "ami_name": "packer-example {{timestamp | clean_ami_name}}"
}]
```

Specify the builder as an element inside a JSON array. Here we've only specified one builder, but you can specify more than one to build a series of images.

TIP We'll see more about multiple builders in Chapter 7.

Specify the builder you want to use using the `type` field, and note that each build in Packer has to have a name. In most cases this defaults to the name of the builder, here specified in the `type` key as `amazon-ebs`. However, if you need to specify multiple builders of the same type—such as if you're building two AMIs—then you need to name your builders using a `name` key.

Listing 1.10: Naming builders blocks

```
"builders": [
  {
    "name": "amazon1",
    "type": "amazon-ebs",
. . .
  },
  {
    "name": "amazon2",
    "type": "amazon-ebs",
  }
]
```

Here we've specified two builders, one named `amazon1`, the other `amazon2`, both with a type of `amazon-ebs`.

NOTE If you specify two builders of the same type, you must name at least one of them. Builder names need to be unique.

To configure the builder, we also specify a number of parameters, called keys, varying according to the needs and configuration of the builder. The first two keys we've specified, `access_key` and `secret_key`, are the credentials to use with AWS to build our image. These keys reference the variables we've just created. We reference variables with the `user` function. Again, our function is wrapped in braces, `{{ }}`, and the input of the variable name is surrounded by back ticks.

Listing 1.11: Referencing variables

```
"access_key": "{{user `aws_access_key`}}",
"secret_key": "{{user `aws_secret_key`}}",
```

This will populate our two keys with the value of the respective variables.

We also specify some other useful keys: the region in which to build the AMI, and the source AMI to use to build the image. This is a base image from which our new AMI will be constructed. In this case we've chosen an Ubuntu 17.04 base AMI located in the `us-east-1` region.

TIP If you're running this build in another region, then you'll need to find an appropriate image.

We also specify the type of instance we're going to use to build our image, in our case `t2.micro` instance type, which should be in the free tier for most accounts.

Lastly, we specify a name for our AMI:

Listing 1.12: Naming our AMI

```
"ami_name": "packer-example {{timestamp | clean_ami_name}}"
```

Our AMI name uses two functions: `timestamp` and `clean_ami_name`. The `timestamp` function returns the current Unix timestamp. We then feed it into the `clean_ami_name` function, which removes any characters that are not supported in an AMI name. This also gives you some idea of how you can call functions

and chain functions together to pipe the output from one function as the input of another.

The resulting output of both functions is then added as a suffix to the name `packer-example`. So the final AMI name would look something like:

```
packer-example 1495043044
```

We do this because AMI images need to be uniquely named.

NOTE There's also a `uuid` function that can produce a UUID if you want a more granular name resolution than time in seconds.

Communicators

Packer builders communicate with the remote hosts they use to build images with a series of connection frameworks called communicators. You can consider communicators as the "transport" layer for Packer. Currently, Packer supports SSH (the default), and WinRM (for Microsoft Windows), as communicators. Communicators are highly customizable and expose most of the configuration options available to both SSH and WinRM. You configure most of these options in the individual builder. We'll see some of this in Chapter 4, and there is also documentation on the Packer site.

Validating a template

Before we build our image, let's ensure our template is correct. Packer comes with a useful validation sub-command to help us with this. It performs syntax checking and validates that the template is complete.

We can run validation with the `packer validate` command.

Listing 1.13: Validating a template

```
$ packer validate initial_ami.json
Failed to parse template: Error parsing JSON: invalid character
'"' after object key:value pair
At line 4, column 6 (offset 49):
    3:     "aws_access_key": ""
    4:     "
           ^
```

Oops. Looks like we have a syntax error in our template. Let's fix that missing comma and try to validate it again.

Listing 1.14: Validating the template again

```
$ packer validate initial_ami.json
Template validated successfully.
```

Now we can see that our `initial_ami.json` template has been parsed and validated. Let's move on to building our image.

TIP You can use the `packer inspect` command to interrogate a template and see what it does.

Building our image

When Packer is run, the `amazon-ebs` builder connects to AWS and creates an instance of the type specified and based on our source AMI. It then creates a new AMI from the instance just created and saves it on Amazon. We execute the build by running the `packer` command with the `build` flag like so:

Listing 1.15: Building our initial AMI image

```
$ packer build initial_ami.json
amazon-ebs output will be in this color.

==> amazon-ebs: Prevalidating AMI Name...
==> amazon-ebs: Error querying AMI: NoCredentialProviders: no
valid providers in chain
==> amazon-ebs: caused by: EnvAccessKeyNotFound: failed to find
credentials in the environment.
==> amazon-ebs: SharedCredsLoad: failed to load profile, .
==> amazon-ebs: EC2RoleRequestError: no EC2 instance role found
==> amazon-ebs: caused by: RequestError: send request failed

. . .

==> Builds finished but no artifacts were created.
```

Ah. Our build has failed as we haven't specified values for our variables. Let's try that again, this time with some variable values defined.

Listing 1.16: Building our initial AMI image again

```
$ packer build \
  -var 'aws_access_key=secret' \
  -var 'aws_secret_key=reallysecret' \
  initial_ami.json
```

TIP The Amazon EC2 builders also support AWS local credential configuration. If we have local credentials specified, we could skip defining the variable values.

Now when we run the build you'll start to see output as Packer creates a new instance and builds an image from it.

Listing 1.17: Building the actual initial_ami image

```
amazon-ebs output will be in this color.

==> amazon-ebs: Prevalidating AMI Name...
    amazon-ebs: Found Image ID: ami-a025aeb6
==> amazon-ebs: Creating temporary keypair: packer_591c9ddd-aff8-
2980-8656-3f4187dc0627
==> amazon-ebs: Creating temporary security group for this
instance...
==> amazon-ebs: Authorizing access to port 22 the temporary
security group...
==> amazon-ebs: Launching a source AWS instance...
    amazon-ebs: Instance ID: i-0da4443cbe9779acc
==> amazon-ebs: Adding tags to source instance
    amazon-ebs: Adding tag: "Name": "Packer Builder"
==> amazon-ebs: Waiting for SSH to become available...
==> amazon-ebs: Connected to SSH!
==> amazon-ebs: Stopping the source instance...
==> amazon-ebs: Waiting for the instance to stop...
==> amazon-ebs: Creating the AMI: initial-ami 1495047645
    amazon-ebs: AMI: ami-6a40397c
==> amazon-ebs: Terminating the source AWS instance...
==> amazon-ebs: Cleaning up any extra volumes...
==> amazon-ebs: No volumes to clean up, skipping
==> amazon-ebs: Deleting temporary security group...
==> amazon-ebs: Deleting temporary keypair...
Build 'amazon-ebs' finished.

==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs: AMIs were created:

us-east-1: ami-6a40397c
```

We can see our log output is colored for specific builders. Log lines from a specific builder are prefixed with the name of the builder: `amazon-ebs`.

TIP You can also output logs in machine-readable form by adding the `-machine-readable` flag to the build process. You can find the machine-readable output's format in the Packer documentation.

We can see Packer has created an artifact: our new AMI `ami-6a40397c` in the `us-east-1` region. Let's have a quick look at that AMI in the AWS Console.

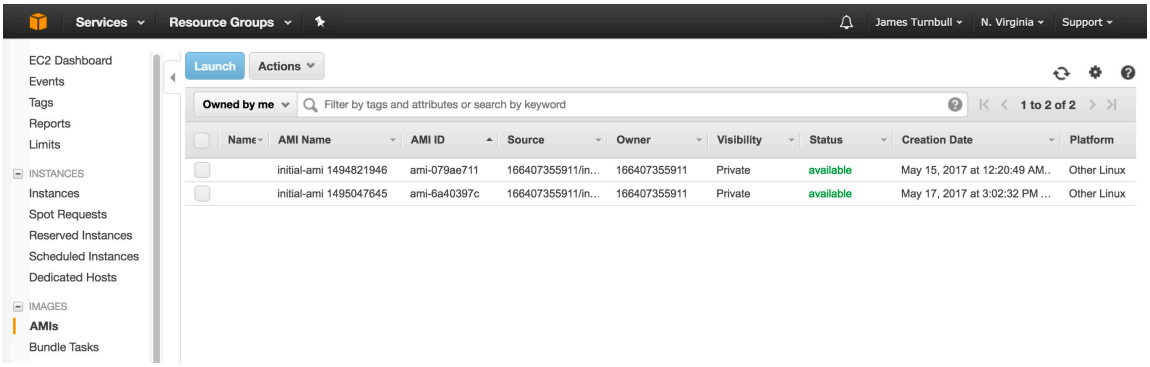

Figure 1.2: Our new AMI

TIP Your new AMI occupies storage space in your AWS account. That costs some money—not a lot, but some. If you don't want to pay that money, you should clean up any AMIs you create during testing.

We can see the AMI has been named using the concatenation of the `packer-example` text and the output of the `timestamp` function. Neat!

NOTE If the build were for some other image type—for example, a virtual machine—then Packer might emit a file or set of files as artifacts from the build.

Summary

You've now created your first Packer image. It wasn't very exciting though—just a clone of an existing AMI image. What happens if we want to add something to or change something about our image? In the next chapter we'll learn all about Packer's provisioning capabilities.

List of Figures

Listings

Index

Thanks! I hope you enjoyed the book.

ISBN 9780988820272