Puppet for Containerization

This book teaches you how to take advantage of the new benefits of containerization systems such as Docker, Kubernetes, Docker Swarm, and Docker UCP, without losing the panoptical power of proper configuration management.

You will learn how to integrate your containerized applications and modules with your Puppet workflow. You will also understand how to manage, monitor, and orchestrate hosts to keep deployed containers running seamlessly. With the help of this book, you can efficiently automate and document with containers, as a part of your system.

The book will also cover use cases of deploying Puppet within a containerized environment.

Who this book is written for
This book is designed for system administrators who are looking to explore containerization. Intermediate experience and expertise with Puppet is presumed.

What you will learn from this book
- Write Puppet modules to build Docker
- Create Docker Compose templates with .erb files
- Get to know the architecture of Docker schedulers
- Create a Docker network and service discovery framework
- Build a fully functional Docker scheduler with Puppet
- Make Docker production ready with Puppet

Learn about configuration management and gain complete control of your Docker containers using Puppet.
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Installing Docker with Puppet'
- A synopsis of the book’s content
- More information on Puppet for Containerization
About the Author

Scott Coulton is a solutions architect with 10 years of experience in the field of managed services and hosting space. He has extensive experience in architecture and in rolling out systems and network solutions for national and multinational companies with a wide variety of technologies including AWS, Puppet, Docker, Cisco, VMware, Microsoft, and Linux. His design strengths are in cloud computing, automation, and security space.

You can find him at https://www.linkedin.com/in/scott-coulton-22864813. You can find him on Twitter at @scottcoulton and on GitHub at https://github.com/scotty-c.
Preface

This book teaches you how to take advantage of the new benefits of containerization systems such as Docker, Kubernetes, Docker Swarm, and Docker UCP, without losing the panoptical power of proper configuration management. You will learn to integrate your containerized applications and modules with your Puppet workflow.

What this book covers

Chapter 1, Installing Docker with Puppet, covers how to create a development environment with Docker using Puppet. We will look at how to install Vagrant and VirtualBox. Then, we will look at Puppet Forge and how to search for modules and their dependencies. We will briefly touch upon r10k to be our transport mechanism from the Puppet Forge to our environment. Then, we build our environment with Puppet.

Chapter 2, Working with Docker Hub, covers a lot about the Docker Hub ecosystem: what are official images, how automated builds work, and of course, working with images in three different ways.

Chapter 3, Building a Single Container Application, contains our first Puppet module to create a Docker container. In this chapter, we will look at writing rspec-puppet unit tests to make sure that our module does what it's meant to do. We will know how to map our Puppet module dependencies with our metadata.json and fixtures.yml files.

Chapter 4, Building Multicontainer Applications, introduces Docker Compose. We will look at the docker-compose .yaml file construct. We will then take that knowledge and create a Puppet template (.erb file) and wrap that into a module. We will also touch on the Docker Compose functionality that will let us scale containers.
Chapter 5, Configuring Service Discovery and Docker Networking, introduces two very important topics when working with containers. First, we will look at service discovery, what it is, why do we need it, and lastly, the different types of service discovery.

Chapter 6, Multinode Applications, introduces all the skills that you've learned in the book so far. We are really going to step it up a notch. In this chapter, we are going to deploy four servers, and we are going to look at how to Consul cluster. In this chapter, we are going to look at the two ways to network our containers. Firstly, using the stand host IP network, that our Consul cluster will communicate on. We will also install the ELK (Elasticsearch, Logstash, and Kibana) stack.

Chapter 7, Container Schedulers, covers container schedulers such as Docker Swarm and Kubernetes. Then, we will build a dev environment containing four servers, three cluster nodes, and a master. We will also build a Docker network and service discovery framework.

Chapter 8, Logging, Monitoring, and Recovery Techniques, will take the environment that we created in the last chapter and add monitoring, logging, and recovery techniques to it. This will make our applications robust and ready for production.

Chapter 9, Best Practices for the Real World, focuses more on the best practices for deploying Puppet itself within a containerized environment using all the new skills that you learned in the previous chapters. By the end of this journey, readers will be able to master Puppet and Docker and apply them in the real world.
Installing Docker with Puppet

In this chapter, we will be setting up our development environment so that we can develop our first container application. To do this, we will use Vagrant. In our first topic, we will look at how to install Vagrant. We will look at how a Vagrantfile is constructed using Puppet as the provisioner. We will also look at how to get Puppet modules from the Puppet Forge using a puppetfile and r10k. In the last topic, we will install Docker on a Centos 7 box with Puppet. The following are the topics that we will cover in this chapter:

- Installing Vagrant
- An introduction to Puppet Forge
- Installing Docker

Installing Vagrant

You may ask, why are we using Vagrant for our development environment?

Vagrant is a must-have for Puppet development. The idea that you can spin up environments for development locally in minutes was a revolution in Vagrant's early releases. The product has now grown in leaps and bounds, with multiple provisioners such as Chef and Salt. Paired with multiple virtualization backends such as VirtualBox, VMware Workstation/Fusion, KVM, and we are going to use VirtualBox and Puppet as your provisioner.
The installation

Let's install Vagrant. Firstly, we will need our virtualization backend, so let’s download and install VirtualBox. At the time of writing, we use VirtualBox 5.0.10 r104061. If that's outdated by the time you read this book, just grab the latest version.

You can download VirtualBox from https://www.virtualbox.org/wiki/Downloads. Choose the version for your OS, as shown in the following screenshot:

Once the package is downloaded, follow the given installation process for your OS.

**VirtualBox**

Follow these steps to install Vagrant on Mac OSX:

1. Go to your Downloads folder and double-click on VirtualBox.xxx.xxx.dmg. The following installation box will pop up:
2. Then, click on `VirtualBox.pkg`. Move on to the next step, as shown in the following screenshot:

![Installer screenshot](image)

The installer will then check whether the software is compatible with the Mac OSX version.
3. After this, click on **Continue**. Once the check is successful, we can move on to the next step:

![Image of installation process]

4. We then choose the default location for the installation and click on **Install**.

5. Then, enter your admin password and click on **Install Software**:

![Image of password input]

[4]
The installation is now complete. The following screenshot shows what the screen looks like after completing the installation:

Now that we have the virtualization backend, we can install Vagrant:

**DOWNLOAD VAGRANT**

Below are the available downloads for the latest version of Vagrant (1.74). Please download the proper package for your operating system and architecture.

You can find the SHA256 checksums for Vagrant 1.74 online and you can verify the checksums signature file which has been signed using HashiCorp’s GPG key. You can also download older versions of Vagrant from the releases service.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAC OS X</strong></td>
<td>Universal (32 and 64-bit)</td>
</tr>
<tr>
<td><strong>WINDOWS</strong></td>
<td>Universal (32 and 64-bit)</td>
</tr>
<tr>
<td><strong>DEBIAN</strong></td>
<td>32-bit</td>
</tr>
<tr>
<td><strong>CENTOS</strong></td>
<td>32-bit</td>
</tr>
</tbody>
</table>
At the time of writing this book, we are going to use Vagrant 1.7.4; if that is no longer the latest version, please grab the latest one. You can find this version of Vagrant at https://www.vagrantup.com/downloads.html. Again, download the installation package for your OS.

**Vagrant**

Here, we are just going to complete a standard installation. Follow these steps to do so:

1. Go to the folder in which you downloaded `vagrant-1.7.4.dmg` and double-click on the installer. You will then get the following pop up:

   ![Vagrant installer interface](image)

2. Double-click on `vagrant.pkg`.

3. Then, in the next dialogue box, click on **Continue**:

![Continue Button]

4. Then, click on the **Install** button:

![Install Button]

---

[7]
5. Enter your admin password in the given field:

6. Once the installation is complete, open your terminal application. In the command prompt, type `vagrant`. After this, you should see the following screenshot:
Vagrantfile

Now that we have a fully working Vagrant environment, we can start with and look at how Vagrant works and how we are going to provision our machines. As this book is not about Vagrant, we won’t be writing a Vagrantfile from scratch. Instead, I have created a Vagrantfile that we will be using throughout the book:

```bash
✓ [scottcoulton@scotts-MacBook-Pro ~]
08:18 $ vagrant
Usage: vagrant [options] <command> [<args>]

-v, --version                  Print the version and exit.
-h, --help                     Print this help.

Common commands:
  box                          manages boxes: installation, removal, etc.
  connect                     connect to a remotely shared Vagrant environment
  destroy                     stops and deletes all traces of the vagrant machine
  global-status               outputs status Vagrant environments for this user
  halt                        stops the vagrant machine
  help                        shows the help for a subcommand
  hosts                       Information about hostnames managed by the vagrant-hosts plugin
  init                        initializes a new Vagrant environment by creating a Vagrantfile
  login                       log in to HashiCorp’s Atlas
 ocker                       packages a running vagrant environment into a box
  pe-build                    Commands related to PE Installation
  plugin                      manages plugins: install, uninstall, update, etc.
  provision                   provisions the vagrant machine
  push                        deploys code in this environment to a configured destination
  rdp                         connects to machine via RDP
  reload                      restarts vagrant machine, loads new Vagrantfile configuration
  resume                      resume a suspended vagrant machine
  scp                         copies data into a box via SCP
  share                       share your Vagrant environment with anyone in the world
  ssh                         connects to machine via SSH
  ssh-config                  outputs OpenSSH valid configuration to connect to the machine
  status                      outputs status of the vagrant machine
  suspend                     suspends the machine
  up                           starts and provisions the vagrant environment
  vbguest                     prints current and latest Vagrant version

For help on any individual command run `vagrant COMMAND -h`

Additional subcommands are available, but are either more advanced or not commonly used. To see all subcommands, run the command `vagrant list-commands`.
```

You can download or Git pull the repo from https://github.com/scotty-c/vagrant-template.
Let's look at the Vagrant file construct:

```ruby
# mode: ruby
# vi: set ft=ruby:

# Specify minimum Vagrant version and Vagrant API version
Vagrant.require_version ">= 1.6.0"
VAGRANTFILE_API_VERSION = "2"

# Require YAML module
require 'yaml'

# Read YAML file with box details
servers = YAML.load_file('servers.yml')

# Create boxes
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  # Iterate through entries in YAML file
  servers.each do |server|
    config.vm.define server[name] do |vm|
      vm.hostname = server[name]
      vm.box = server[box]
      vm.network "private_network", ip: server[ip]

      server[forward_ports].each do |port|
        vm.network :forwarded_port, guest: port[guest], host: port[host]
      end

      vm.provider :virtualbox do |vm|
        vm.cpus = server[cpu]
        vm.memory = server[memory]
      end

      server[shell_commands].each do |sh|
        vm.provision "shell", inline: sh[shell]
      end

      vm.provision :puppet do |puppet|
        puppet.temp_dir = "tmp"
        puppet.options = ["--modulepath/tmp/modules", '--verbose']
        puppet.hiera_config_path = "hiera.yaml"
      end
    end
  end
end
```
As you can see from the preceding screenshot, the Vagrantfile is actually a Ruby file. As it is Ruby, it opens up a world of opportunities for us to make our code elegant and efficient. So, in this Vagrantfile, we have extracted all the low-level configurations and replaced them with a few parameters. Why are we doing this? The reason is to split up our logic from our configuration and also iterate our configuration in order to stop replication of our code. So, where is all the configuration stored? The answer is in the servers.yaml file. This is where we set the vagrant box that we want to deploy, the number of CPUs for the box, the internal network’s IP, the hostname, the forwarded ports between the guest and host, and the RAM and shell provider for bash commands that we need to get the environment ready for Puppet to run, for example, downloading modules and their dependencies from the Puppet Forge:

The benefit of this approach is also that any developer using a Vagrantfile does not need to actually modify the logic in the Vagrantfile. They only need to update the configuration in servers.yaml. As we go through the book, we will work with the other files in the repository, such as Puppetfile, hieradata, and manifests. Now that we have set up our Vagrant environment, let’s look at how to get our Puppet modules from the Puppet Forge.

**Welcome to the Puppet Forge**

In this topic, we will look at how to find modules from the Puppet Forge. Then, we will see how to pull them with their dependencies using a puppetfile and r10k. This will set us up for our last topic, *Installing Docker with Puppet*.

**The Puppet Forge**

One of the great things about puppetlabs and their products is the community. If you ever get a chance to attend PuppetConf or a Puppet Camp, depending on where you live, I would really recommend you to attend it. There will be a wealth of knowledge there and you will meet some really great people.
Installing Docker with Puppet

The Puppet Forge is a website that puppetlabs runs. It is a place where other Puppet developers publish modules that are ready to use. You might be asking, what about GitHub? Can't you get modules from there? Yes, you can. The difference between the Puppet Forge and GitHub is that the Puppet Forge is the stable, releasable version of the module, whereas GitHub is the place to contribute to the module, that is, a place to create pull requests.

You can find the Puppet Forge at https://forge.puppetlabs.com/.

The following screenshot shows the home page of Puppet Forge:

![Puppet Forge Home Page]

Now that we have been introduced to the Puppet Forge, let's use it to find our Docker module that we will be using to build our environment.

We are going to use the garethr/docker Docker module, which you can find at https://forge.puppetlabs.com/garethr/docker.
Now that we have selected our module, we can move on to setting up our puppetfile:

```
Creating our puppetfile
```

In the previous topic, we cloned our Vagrant template using Git. In that repo, there is also a puppetfile. A puppetfile is used as a control file for our modules. It will list all the modules that we need (in this instance, just to install Docker). r10k will then reference the puppetfile and pull the modules from the Puppet Forge into our environment's directory.

As modules have dependencies, we need to make sure that we capture them in our puppetfile. For the Docker module, we have three dependencies: `puppetlabs/stdlib (>= 4.1.0)`, `puppetlabs/apt (>= 1.8.0 <= 3.0.0)`, and `stahnma/epel (>= 0.0.6)`, as shown in the following screenshot.

Now, we know all the modules that we need to build a Docker environment. We just need to add them to our puppetfile.
Installing Docker with Puppet

The following screenshot is an example of what the puppetfile should look like:

Now, when we run `vagrant up`, r10k will pull the modules from the Puppet Forge. We invoke r10k on line 13 of `servers.yaml` with the `r10k puppetfile install—verbose` command. The following screenshot shows the output of this command:

```
# /usr/bin/ruby env
require "socket"
$hostname = Socket.gethostname

forge = 'http://forge.puppetlabs.com'

mod 'puppetlabs/stdlib', '4.1.0'
mod 'puppetlabs/apt', '2.2.1'
mod 'stahnma/epel', '0.0.6'
mod 'garethr/docker', '5.0.0'
```

If we are successful, the terminal will provide the following output:

```
== node-01: INFO  
--- Updating module /tmp/modules/stdlib
== node-01: INFO  
--- Updating module /tmp/modules/apt
== node-01: INFO  
--- Updating module /tmp/modules/epel
== node-01: INFO  
--- Updating module /tmp/modules/docker
```

Now that we have our puppetfile set up, we can install Docker.
Installing Docker

In this topic, we will put together all the configuration from our Vagrant repo and knowledge of the Puppet Forge to create the Docker environment.

Setting our manifests

The first thing that we need to do to install Docker is set our manifest to include the Docker class on our node. To do this, let's go to our Vagrant repo. In the repo, there is a file in the manifests directory called default.pp. We need to edit the file to include the Docker class node 'node-01' { include docker}. We can now save the file, and we are ready to run our environment.

The first step is to open our terminal and change to root of the Vagrant repo. Then, we need to enter the vagrant up command:

```
14:50 $ vagrant up
```

This will now provide us with our CentOS 7 box. Install r10k and then run Puppet and apply the Docker class. This will take about 4 minutes depending on your laptop and network connection. If the box was provisioned successfully, you will see the following output:

```
[scottcoulton@scotts-MacBook-Pro ~]
```

We can also verify that the Docker installation was successful by logging in to the box via SSH. We will do that with the vagrant ssh command. Once we are in, we will sudo up to root (sudo -i). Now, let's just check whether Docker is installed with the docker command.
You will see the following output on the terminal:

```
Usage: docker [OPTIONS] COMMAND [arg...]
  docker daemon [--help | ...]
  docker [-h | --help | --version]

A self-sufficient runtime for containers.

Options:
  --config=/path/to/docker.conf Location of client config files
  --config-opts=data
  --debug=false Enable debug mode
  --disable-legacy-registry=false Do not contact legacy registries
  --host=[] Daemon socket(s) to connect to
  --hversion=false Print version
  --log-level=info Set the logging level
  --tls=false Use TLS; implied by --tlsv1
  --tlsv1=false Use TLS; implied by --tlsv1
  --tlsv1.2=false Use TLS version 1.2
  --tlsv1.3=false Use TLS version 1.3
  --tlsv1.4=false Set the TLS version
  --tlsv1.5=false
  --tlsv1.6=false
  --tlsv1.7=false
  --tlsv1.8=false
  --tlsv1.9=false
  --tlsv1.10=false
  --tlsv1.11=false

Commands:
  attach        Attach to a running container
  build         Build an image from a Dockerfile
  commit        Create a new image from a container’s changes
  cp            Copy files/folders between a container and the local filesystem
  create        Create a new container
  diff          Inspect changes on a container’s filesystem
  events        Get real time events from the server
  exec          Run a command in a running container
  export        Export a container’s filesystem as a tar archive
  history       Show the history of an image
  images        List images
  import        Import the contents from a tarball to create a filesystem image
  info          Display system-wide information
  inspect       Return low-level information on a container or image
  kill          Kill a running container
  load          Load an image from a tar archive or URL
  login         Register or login to a Docker registry
  logout        Log out from a Docker registry
  logs          Fetch the logs of a container
  network       Manage Docker networks
  pause         Pause all processes within a container
  pull          Pull an image or a repository from a registry
  push          Push an image or a repository to a registry
  rename        Rename a container
  restart       Restart a container
  rm            Remove one or more containers
  rmi           Remove one or more images
  run           Run a command in a new container
  save          Save one or more images
  search        Search the Docker Hub for images
  start         Start one or more stopped containers
  stop          Stop a running container
  tag           Tag an image into a repository
  top           Display the running processes of a container
  unpause       Unpause all processes within a container
  version       Show the Docker version information
  volume        Manage Docker volumes
  wait          Block until a container stops, then print its exit code

Run `docker COMMAND --help` for more information on a command.
```
Summary

In this chapter, we covered how to create a development environment with Docker using Puppet. We looked at how to install Vagrant and VirtualBox. Then, we looked at the Puppet Forge, how to search for modules and their dependencies. We then took the dependencies and mapped them to a puppetfile. We briefly touched on r10k, which is our transport mechanism from the Puppet Forge to our environment. Then, we built our environment with Puppet.

In the next chapter, we’ll take a look at how to access Docker Hub and pull public images.
Where to buy this book

You can buy Puppet for Containerization from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.