PupDocker

PHP Demo Application

Deploy Docker Containers using Puppet

Ukbe Akdogan

October @2016

Tags
puppet vagrant docker php laravel

Introduction

PupDocker is a deployment tool which utilizes Puppet server-agent architecture in order to configure resources on Puppet server and Docker images. A Vagrant box hosts both Puppet server and Docker containers. All resources are created from ground up with a single vagrant up command. At the end of the deployment process three separate Docker containers are created using multiple images. Vagrant box is launched with **ubuntu/xenial64** and Docker images are created using **ubuntu:16.04** base image.

PupDocker is the project name of the deployment tool which automatically creates the vagrant environment with all the resulting components.

EmpQuery is a simple PHP application leveraging Laravel framework in order to list queried employee records from database.

Project	Repository
PupDocker	https://github.com/ukbe/pupdocker
EmpQuery	https://github.com/ukbe/empquery

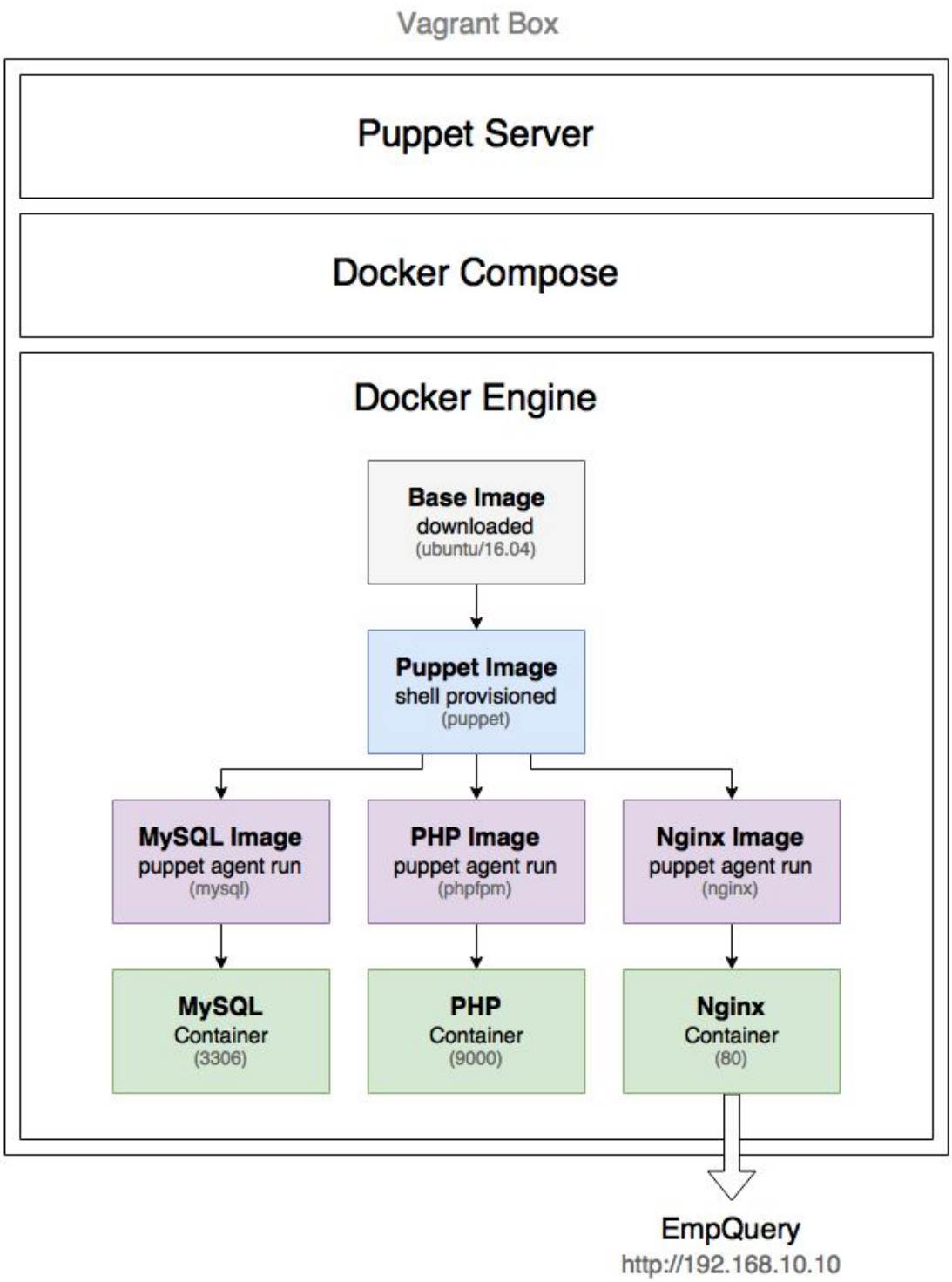
Requirements

Following tools are required to be installed on the host which you will run *vagrant up* command.

- Git
- Vagrant
- Virtualbox

Default Vagrantfile is configured to provision 4 GB of memory to Ubuntu 16.04 box. You may change this setting as you like, however 1 GB of memory causes errors while starting Puppet server.

Architecture



Puppet Server

Puppet server is installed and configured by - *install_puppetserver.sh* - shell script triggered by Vagrant. This script also installs librarien-puppet package which manages Puppet modules listed in */etc/puppetlabs/code/Puppetfile*. Puppet configuratiion is copied into /etc/puppetlabs directory with provisioning shell script.

Puppet server holds the configuration information needed to configure the Vagrant box and trigger Docker tasks. Puppet server also provides the configuration of Docker images. Puppet agent is triggered during Docker images builds (mysql, php, nginx).

Puppet configuration is stored in **./puppet** directory of project and can be listed as:

- **./bash/install_puppetserver.sh** Puppet server installation script]
- **./code** [Puppet modules, manifests, Puppetfile, ..]
- **./autosign.conf** [List of hostnames that certificates will be auto signed during Puppet]
- **./auth.conf** [Puppet API access config. Added a snippet to allow certificate management]
- **./manifests/default.pp** [Puppet equivalent of install_puppetserver.sh script. NOT USED]

Librarian-Puppet installs the required modules into */etc/puppetlabs/code/modules* directory.

Docker Images

Official Ubuntu 16.04 image is used as a base image to create other images. This image is downloaded by puppet-docker module by issuing *docker pull*.

Image: puppet [./docker/puppet-image/Dockerfile]

Initial configuration of containers are handled by Puppet server and agent running in the containers during build. **puppet** image is created to maintain a fresh copy of a non functional container with Puppet agent installed. Puppet agent installation in this container is triggered by Puppet agent on Vagrant box and handled with a a bunch of shell commands in its Dockerfile. This image is the initial image which other images are based on (mysql, php, nginx). Role assignments are specified in Dockerfile of each image with ENV setting. While building Puppet agent looks for HOSTROLE environment variable and applies required configuration.

Image: mysql [./docker/database-image/Dockerfile]

mysql image is based on **puppet** image and it is differentiated with *HOSTROLE=database* environment variable. Puppet agent applies configuration in *role_database.pp* manifest. MySQL database is installed and configured. Test data is cloned from Datacharmer GitHub repository and imported to database during build. Building this image takes more time than other images since downloading test data and import processes takes some time.

Image: phpfpm [./docker/php-image/Dockerfile]

phpfpm image is based on **puppet** image and it is differentiated with *HOSTROLE=php-fpm* environment variable. Puppet agent applies configuration in *role_php-fpm.pp* manifest. Php-fpm is installed and configured to allow fastcgi service over TCP port 9000. EmpQuery project is cloned from GitHub repository and configured with composer. Environment configuration file [*./docker/php-image/.env*] is copied into project root directory. Required PHP extensions are installed by Puppet agent.

Image: nginx [./docker/webserver-image/Dockerfile]

Nginx image is based on puppet image and it is differentiated with *HOSTROLE=webserver* environment variable. Puppet agent applies configuration in role_webserver.pp manifest. Nginx is installed and configured. Some Laravel requirements are applied on server configuration. Virtualhost is configured to forward PHP requests to **con_php** container through TCP 9000 port.

Implementation

You can see the steps to implement PupDocker environment on your host machine below.

Before starting make sure that you do not have 192.168.10.10 ip on your host or network. If this is the case then change IP configuration in Vagrantfile in order to prevent IP conflict.

1. Clone PupDocker repository in a suitable directory.

   # git clone https://github.com/ukbe/pupdocker

2. Change active directory into the project root directory.

   # cd pupdocker

3. Checkout master branch.

   # git checkout origin master

4. Trigger Vagrant

   # vagrant up

Vagrant will pull ubuntu/16.04 image from Atlas and start provisioning.
This process might take 15-20 minutes depending on your hardware and network conditions.
You can see the details of this process below.

	Shell Provisioning	Puppet Agent	Docker Compose	
Vagrant Box	**1**	**2**		
Docker Images			**3**	
Docker Containers				**4**

Step 1

- In this step, Puppet server is installed on Vagrant box.

- Configuration files and directories are copied to related locations from project directories.
- Puppet modules are installed from Puppetforge.
- Puppet agent is enabled and started.

Step 2

- Docker and Docker Compose packages are installed.
- **ubuntu:16.04** Docker image is downloaded.
- **puppet** image gets built.
- **docker-compose up** command is executed to build images and start the containers.

Step 3

- Docker Compose starts building service images (mysql, phpfpm, nginx). Images are created one by one.
- This step takes most of the installation time. (test_db clone/import, project clone, ..)

Step 4

- Docker Compose creates containers with configuration specified in docker-compose.yml file.
- This step takes just 2-3 seconds.
- At this time you can access the application at http://192.168.10.10

You can stop, start, remove containers anytime by issuing commands below. Commands should be executed in */vagrant/docker* folder.

Stop project containers.
 # docker-compose stop

Start project containers.
 # docker-compose start

Remove project containers.
 # docker-compose rm

Create project containers.
 # docker-compose up

You can also list all existing images by issuing command below.
 # docker images

Outputs

Implementing the PupDocker environment will end up with 5 Docker images and 3 running containers.

 # docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest	77343b1b7367	3 hours ago	314.8 MB
phpfpm	latest	b8e2a9dae843	3 hours ago	423.1 MB
mysql	latest	3d78845b56c4	3 hours ago	1.148 GB
puppet	latest	f7065797d6ed	3 hours ago	261.2 MB
ubuntu	16.04	c73a085dc378	2 weeks ago	127.1 MB

 # docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8e312187dc8a	nginx	"nginx"	3 hours ago	Up 3 minutes	0.0.0.0:80->80/tcp, 443/tcp	docker_con_nginx_1
7f5618329099	phpfpm	"php-fpm7.0"	3 hours ago	Up 3 minutes	0.0.0.0:9000->9000/tcp	docker_con_php_1
6312b5466eed	mysql	"mysqld"	3 hours ago	Up 3 minutes	0.0.0.0:3306->3306/tcp	docker_con_mysql_1

How to customize?

Puppet manifests allow necessary customizations however, you need to rebuild related images for changes to take effect. You can freely change Vagrant box IP.

Downsides

Most of the time data in containers should be accessible from outside. MySQL data directory, log files, project folders are some of the examples that could be expected to be accessible outside of the container. PupDocker is developed to contain data and project files in containers. However, you can still mount container volumes to host directories or other containers.

On the other hand, containers are expected to be small in size and using standard Puppet modules results large sized containers and unnecessary files in containers. Customized Puppet modules would help reducing size and improving performance of containers.

I hope you like it.

Regards,

Ukbe Akdogan