











------------------------------------------------------------------------------------------------------------------------------------------

In this tutorial series, I want to show you how you can easily build a reproducible development for symfony or plain php with ansible and vagrant.

**What do we want / need?**

Before we can dive deep into symfony in general, we need environment to run our code in. Sure we could install PHP directly on our Mac, Linux or Windows.

We need to build a environment, that’s easy to use, generic enough to extend and production alike, because we don’t want to be a “It works on my machine”.

To accomplish this, we will use an virtual machine, that is able to act as a production server, but also easy to install and maintain. And of course it should run on all major operating systems like Mac, Linux and Windows

## Vagrant + Ansible = Love

So to build our cool dev virtual machine, we will create a VirtualBox managed by Vagrant. First we need to install all the stuff to our local machine, so here you can find the installation instructions for every product:

1. [VirtualBox](https://www.virtualbox.org/wiki/Downloads)
2. [Vagrant](https://www.vagrantup.com/downloads.html)
3. [Ansible](http://docs.ansible.com/ansible/intro_installation.html)
4. Install vagrant-hostsupdater plugin:vagrant plugin install vagrant-hostsupdater.

(to automatically update /etc/hosts file)

Vagrant.configure(2) do |config|

    config.vm.define "symfony" do |symfony|

        # Base template for virtualbox, we use ubuntu 14.04 here

        symfony.vm.box = "ubuntu/trusty64"

        # Domain on which our application will respond later on

        symfony.vm.hostname  = "symfony.dev"

        # IP address will be used by the VM

        symfony.vm.network :private\_network, ip: "192.168.33.151"

        # Tell vagrant to run ansible as a provisioner

        symfony.vm.provision :ansible do |ansible|

            # where the playbook is located

            ansible.playbook = "provisioning/playbook.yml"

        end

    end

    # Access the shared vagrant directory via NFS, otherwise slow on mac and windows

    config.vm.synced\_folder ".", "/vagrant", type: "nfs"

    config.vm.provider "virtualbox" do |v|

        # tell virtualbox to give our machine 1 GB RAM and 2 Cores

        v.memory = 1024

        v.cpus = 2

    end

end

## Ansible Setup

Now it’s time to create the necessary file structure for ansible to work. We need the **provisioning** directory, with a playbook.yml inside it and some additional files.

mkdir -p provisioning/{group\_vars,roles}

**group\_vars:** Here lives the dynamic part of our configuration. We will set database parameters and additional packages here.  
**roles:** to stick to ansible best practices, we create one role for each component, e.g. one for php, nginx, postgresql and so on

1.) **Let’s create our first role in which we can set the locale for our server.**

Create another dir inside roles/ called general. We will use the general role, for configuring stuff like the language and the charset. Also for some packages, which are not crucial for the machine to act as a development machine, for example htop and vim.

mkdir -p provisioning/roles/general/tasks/

Inside the **tasks** directory, you now need to create a **main.yml** file. Ansible always looks first for a main.yml inside the tasks/ dir. So if we later decide to split up our configuration into different **.yml** files, just remember to include them into **main.yml** as a entry point for ansible.

vim provisioning/roles/general/tasks/main.yml

In this file we start defining our first tasks, which is to generate and set the server locale:

- name: generate the server locale

**locale\_gen: name={{ locale }} state=present**

**2.)** Here we see the first dynamic part of our ansible role, the {{ locale }} variable. You can set it dynamically, independent from your role inside the group\_vars directory. Therefore let’s create a file called **all**.

vim provisioning/group\_vars/all

This file will be read used for all vagrant machines provisioned with our Vagrantfile. In our case, just this single machine. Don’t care about multimachine setup now.

Now put the **locale** variable into the **all** file:

locale: en\_US.UTF-8

Ok, so we now generating it, we just need to set it. Therefore we will use another cool feature of ansible, which is **lineinfile**. **Append this to your main.yml inside tasks:**

- name: set locale

lineinfile: dest=/etc/default/locale regexp='^LANG=' line='LANG={{ locale }}'

- name: reload local

raw: source /etc/default/locale

And again we can use our **locale** variable.

Now we got our variables, created our first role. The only thing that’s missing now, is the playbook itself. Just create a file named **playbook.yml**:

vim provisioning/playbook.yml

---

# valid for all hosts provisioned inside this vagrant instance

- hosts: all

# become replaces sudo since ansible v.1.9

become: true

# roles to be executed

roles:

- general

Save the file, and now we should be ready, to do our first boot of the machine and check if our provisioning works:

vagrant up

You will be prompted for the sudo password, since vagrant tries to alter the **/etc/hosts** and **/etc/exports** file. After the normal boot of the virtual machine, you can see the start of the provisioning. If not just type:

**vagrant provision**

And you should get a very similar output to this:

PLAY [all] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [setup] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [symfony]

TASK [general : generate the server locale] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [symfony]

TASK [general : set locale] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [symfony]

TASK [general : reload local] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [symfony]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

symfony : ok=4 changed=0 unreachable=0 failed=0

**Version our config**

That’s nearly it for the first part of this little tutorial. The last thing, I want to do is, version our configuration into a repo, so that we can track changes. Which is quite easy.

Initialize your Git Repo:

git init

Add a gitignore:

vim .gitignore------------------------**Add .vagrant/ and playbook.retry to .gitignore:**

.vagrant/

playbook.retry

Add our files to the index and commit:

git add .

git commit -m '01 - setup vagrant and ansible, created first role'

That’s it

In the next part, I will show you, how to install PHP, nginx and postgresql in your machine.

Retrospect……..

----------------------------------------------------------------------------------------------------------------------------------

**In this part, I will show you how to install PHP 7 along with some dependencies.**

PHP 7

Let’s create a new role, where our PHP action can happen. Therefore create a new directory inside roles, called php:

mkdir -p provisioning/roles/php/tasks

As you can see, I also created the tasks directory. With Ansible we can easily add the PPA to our system. Create the main.yml file with the following content:

- name: add php ppa

apt\_repository: repo='ppa:ondrej/php' state=present

It does the same, as if you would run add-apt-repository ppa:ondrej/php. After adding the ppa, we can start with installing PHP and some dependencies, which we will need later in our symfony project. To be able to change the PHP version easily, we just put the version in our group\_vars. To do so, just append this part to group\_vars/all:

php:

version: 7.1

After that we can complete our task for installing PHP requirements in provisioning/roles/php/tasks/main.yml:

- name: install php requirements for symfony

apt: name=php{{ php.version }}-{{ item }} state=present

with\_items:

- json

- xml

- mbstring

- intl

- opcache

- pgsql

Now you learned another cool thing in ansible and that is with\_items. With with\_items you can simply loop throug a list of items, either right declared here, or declared in the playbook or in your group\_vars/all file.

**If you want to use MySQL instead of PostgreSQL, be sure to replace pqsql with mysql to get the right package.**

**I didn’t include PHP-FPM here for a purpose. In my opinion it deserves an own role, because we will to configure some stuff here. So let’s do a new role here:**

mkdir -p provisioning/roles/php-fpm/{tasks,templates,handlers}

If you don’t know what these curly braces mean, go check out my post about brace expansion, that’s what it is called.

Let’s write the first task, therefore create the main.yml file:

- name: install php{{ php.version }}-fpm

apt: name=php{{ php.version }}-fpm state=present

This will install PHP-FPM, create a default pool config and start the service. Because the standard config won’t work properly with vagrant and we’d have to change things manually, let’s delete it and create our own.

- name: delete default config

file: path=/etc/php/{{ php.version }}/fpm/pool.d/www.conf state=absent

You learned another cool thing here, the file module. You can use it to create, delete and modify files. Now we would have no pool available, so we need to create one. To do so, we first need to talk about templates.

Template time!

Templates are files filled with variables that will be populated or rendered with values either from the playbook, the group or host var or from default values. Ansible uses the Jinja2 templating library, which should be quite familiar if you ever used Django, Symfony with Twig or also Flask with Jinja.

You can easily recognize variables by their opening and closing double curly braces: {{ }}

But the cooler thing are control structures like loops and conditions, with them you can render your templates based on conditions. Control structures alway begin with an opening curly brace, followed by a percent sign and the same in reverse for closing ones: {# #}

Let’s create our template, therefore create a file called pool-template.j2 in provisioning/roles/php-fpm/ and insert this code:

[{{ item.name }}]

user = {{ item.user }}

group = {{ item.group }}

listen = /run/php/php-{{ item.name }}-fpm.sock

listen.owner = {{ item.user }}

listen.group = www-data

pm = dynamic

pm.max\_children = 5

pm.start\_servers = 2

pm.min\_spare\_servers = 1

pm.max\_spare\_servers = 3

We only have variables in here, no control structures, but wait. I call the variable item.name and item.user, they don’t exist. Let’s create them in group\_vars/all:

apps:

- name: symfony

user: vagrant

group: vagrant

What’s wrong with me? Why did I call it apps and not item? Don’t worry, we will get to that in a second.

You could just put ansible directly in there, but let’s you want to reuse this role in production, then you are easily able to change the username to app\_projectname or so.

And another cool thing here, we just made a list of dicts in yaml called apps, so we can easily iterate over it and create as much pool configs as we want. And that’s why I called it item in the template, so we can now iterate over the apps list with the awesome function with\_items. Let’s use it and create our rendering task:

- name: render php fpm pools config

template: src=pool-template.j2 dest=/etc/php/{{ php.version }}/fpm/pool.d/{{ item.name }}.conf

with\_items: '{{ apps }}'

Ain’t that simple? Just give template your template file at src and render it to dest. Imagine what cool things you can do with this 😉

The last thing we need for today is to restart the FPM service, to activate our new configuration. Therefore you create a handler. Handlers in ansible are executed after a sucessful run of the playbook, easy as that. You just need to notify them in a task that changed that requires the handler to run.

In our case it’s the rendering of the pool configs, which needs a restart of php-fpm. Just append this notify to our last task:

notify: restart php-fpm

Too easy? Ok, of course we need to create this handler to work! We already create d a directory for them, so just create another main.yml inside there and execute the service module there:

- name: restart php-fpm

service: name=php{{ php.version }}-fpm state=restarted

Let’s test our work! But don’t forget to amend your new roles first!

To do so open up you playbook.yml and add the php and the php-fpm role:

...

- php

- php-fpm

...

Since the php role adds the PPA it has to run before the php-fpm role, just keep that in mind.

Let’s hit vagrant provision and see what we get here:

vagrant provision

Nice!

TASK [php : add php ppa] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [symfony]

TASK [php : install php requirements for symfony] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [symfony] =&gt; (item=[u'php7.1-json', u'php7.1-xml', u'php7.1-mbstring', u'php7.1-intl', u'php7.1-opcache', u'php7.1-pgsql'])

TASK [php-fpm : install php7.1-fpm] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [symfony]

TASK [php-fpm : delete default config] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [symfony]

TASK [php-fpm : render php fpm pools config] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [symfony] =&gt; (item={u'group': u'vagrant', u'name': u'symfony', u'user': u'vagrant'})

RUNNING HANDLER [php-fpm : restart php-fpm] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [symfony]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

symfony : ok=10 changed=7 unreachable=0 failed=0

**Version everything!**

Don’t forget to version your changes!

git add .

git commit -m '02 - install php, php-fpm and render templates'

Wow, we learned a lot of stuff today. We learned how to use services, templates, lists, dicts and a lot more. And just imagine, what badass stuff you can do with this 😉

See you in the next part and don’t forget to subscribe to the newsletter or follow me on any social net. Thanks for reading!