

Pragmatic Virtualization

Hit the ground running with Vagrant, Packer, Virtualbox and Ansible

by Damiano Venturin



Pragmatic Virtualization

Hit the ground running with Vagrant, Packer, Virtualbox and Ansible

Damiano Venturin

This book is for sale at http://leanpub.com/pragmatic_virtualization

This version was published on 2015-06-20



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.



This work is licensed under a [Creative Commons Attribution 3.0 Unported License](#)

Also By **Damiano Venturin**

Crea API che non odierai

Modernizzare Applicazioni Legacy in PHP

This manual is dedicated to all the developers who have lost ground with the recent development technologies and to those who feel isolated and trapped in their own reality.

Contents

Introduction	i
What to expect from this manual	i
Copyleft	iv
The PHP Women cause	iv
Acknowledgements	iv
1. About Virtualization	1
1.1 Get ready	1
1.2 Almost ready	2
1.3 What's virtualization	3
1.4 Why to use virtualization	3
1.5 Virtualization tools	5
1.6 VirtualBox	5
1.7 Vagrant	6
1.8 Packer	7
2. About Vagrant	9
2.1 How Vagrant works	9
2.2 Conventions used in both methods	10
2.3 Create a Vagrant template box from iso file	11
2.4 Create a Vagrant template box from Vagrant Cloud	17
2.5 Some considerations about Vagrant	20
3. Vagrant administration	22
3.1 Listing virtual machines	22
3.2 Improving the Vagrantfile	26
3.3 Destroy your virtual machine	34
3.4 Rebuild the Vagrant virtual machine	36
3.5 How to build another copy	36
4. Automation tools	37
4.1 Automation tools	37
4.2 Ansible	38
4.3 Ansible configuration	40

CONTENTS

4.4	Ansible tools	49
5.	The Suggestotron application	54
5.1	Create the database	54
5.2	Populate the database	55
5.3	Get the Suggestotron code	58
5.4	Add a virtual host for Apache	58
5.5	Run Suggestotron	60
5.6	Play with Suggestotron	61
6.	The production environment	63
6.1	Vagrant and Digital Ocean	63
6.2	Dive deeper into virtualization	77

Introduction

This is a short, modest and straightforward manual written with pragmatism in mind which sums up my experiences with virtualization .

If you don't know what's the meaning of *virtualization*, I will describe it better later on but, for now, consider it as *a set of theories and technologies that allows you to run virtual computers inside your own computer*. Yes, like a matryoshka.

What to expect from this manual

The aim of this manual is to help you to hit the ground running with virtualization.

Nowadays virtualization is considered a common practice in many activities, especially in development, and brings great advantages (see the [virtualization paragraph](#)).

I tried to keep things simple and to limit assumptions so that the reading will be easy also for total beginners in virtualization and in applications development.

I will not cover in detail the theory behind virtualization and this manual will **not** make you an expert [devop](#)¹. As always, becoming an expert requires a lot of practice and studying.

Anyway this manual will provide what's necessary to:

- create, clone, run, maintain and destroy virtual machines, locally (on your workstation) and remotely
- programmatically install, update and configure the software installed inside the virtual machine
- create different isolated environments for different development purposes
- publish your virtual machine on a public server

¹<http://en.wikipedia.org/wiki/DevOps>

The PHPBridge Project

While I was writing the manual I felt the need to make it more useful and real so I wrote it in such a way that it can fit in smoothly in the awesome [PHPBridge Project](http://phpbridge.org)².

The PHPBridge Project, generously created by the [PHP Women group](http://phpwomen.org/)³, helps beginners to learn PHP and it's split in sections:

- the [Installfest](http://phpbridge.org/installfest/)⁴ provides some instructions for installing PHP on your computer
- the [Intro to PHP](http://phpbridge.org/intro-to-php/)⁵ starts from scratch and takes you step-by-step through making a PHP application (called Suggestotron), one line at a time

Unfortunately, as written in the Installfest documentation, the virtual machine is not yet available for download.

You will be provided with a copy of the virtual machine (“the box”) on a USB thumb drive

This is not exactly a big deal because there are several ways to overcome the obstacle but it's definitely not ideal for a remote beginner.

So I thought that it would be nice to offer to the PHPBridge students a closer sight to some virtualization aspects and an easy way to produce their own virtual machines.

After reading this manual, you will end up with a Debian 64 bit virtual machine configured to run the final [Suggestotron application](http://phpbridge.org/suggestotron/)⁶. The virtual machine can be reused as many times as you want and it can become your default development environment for many projects and, optionally, you can decide to publish it on a Digital Ocean public server.



If you are in the process of learning PHP you can start with reading this manual and then you can go ahead with the PHP Bridge project. You will be capable to skip entirely the Installfest section and to continue with the [PHP Language](http://phpbridge.org/intro-to-php/php_language/)⁷ chapter and, finally, to land to the [Adding Topics](http://phpbridge.org/intro-to-php/adding_topics/)⁸ chapter.



If you are not trying to learn PHP or you are not a PHP developer, don't worry you will not have any problem.

²<http://phpbridge.org>

³<http://phpwomen.org/>

⁴<http://phpbridge.org/installfest/linux>

⁵<http://phpbridge.org/intro-to-php>

⁶<http://phpbridge.org/intro-to-php/>

⁷http://phpbridge.org/intro-to-php/php_language

⁸http://phpbridge.org/intro-to-php/adding_topics

Perspective

Virtualization is a huge topic and you can tackle it from many angles and it's really hard to write a manual that covers all the operating systems and all the virtualization softwares.

This is why I felt obliged to choose a perspective that, on one hand, will force some strict requirements and, on the other, will cut off some virtualization technologies, so don't be surprised by the following requirements and by my software selection.

Limits and requirements

These are the minimum requirements for your workstation:

- 64 bit hardware
- A Linux Debian based operating system
- 5GB of free space
- 2GB of RAM



If you have a different hardware architecture or a different operating system you can still use this manual but be ready to adjust some commands.

These are the virtualization softwares in use:

- **Packer** to create a virtual machine starting from an iso image
- **VirtualBox** as the virtualization environment for the virtual machine
- **Vagrant** as virtualization manager
- **Ansible** as automation tool for installing and configure software inside the virtual machine
- **Digital Ocean** as hosting provider for the production environment

These are the list of services and softwares installed and running inside the virtual machine:

- Apache
- PHP5
- Composer
- Psysh
- Xdebug

Somebody will not be happy with my choices (sorry) but, at least, it brings some quick and effective results for some others.



I encourage you to adjust this manual for any different perspective and to re-publish it for free. See the next paragraph.

Copyleft

You can copy, share, modify and reuse this manual accordingly to the [Creative Commons Attribution 3.0 Unported License](https://creativecommons.org/licenses/by/3.0/deed.en_US)⁹.

If you have got this manual from a friend I strongly suggest you to go and get a new fresh copy **for free** from the [Leanpub website](https://leanpub.com/pragmatic_virtualization)¹⁰ because in this way you can be notified by Leanpub about any future update. The updates, of course, will be for free.

While you are on Leanpub, please consider donating to support the PHP Women cause. That would be really appreciated.



I'd love to have this manual reviewed by a native English speaker. If you want to review and improve this manual please send me a Github PR: [content](#)¹¹ and [extra](#)¹². Thank you.

The PHP Women cause

This manual and its author support the PHP Women group.

PHP Women is an inclusive and global network providing support within the PHP community.

Any donation for this manual will be entirely forwarded to the PHP Women group that will use the money to send to conferences developers who can not afford it.

Knowing the importance of conferences in building relationships with other developers and in encouraging people to join the community, I think this is an awesome idea.

Please help to break the isolation barrier! :-)

Acknowledgements

Many thanks to:

- [PHP Women](#)¹³, of course, for the great job done with the [PHPBridge project](#)¹⁴
- [PHP The Right Way](#)¹⁵ from which I took inspiration and some chunks of text :-)

⁹https://creativecommons.org/licenses/by/3.0/deed.en_US

¹⁰https://leanpub.com/pragmatic_virtualization

¹¹https://github.com/damko/pragmatic_virtualization

¹²https://github.com/damko/pragmatic_virtualization_extra

¹³<http://phpwomen.org/>

¹⁴<http://phpbridge.org/>

¹⁵<http://www.phptherightway.com/>

1. About Virtualization

1.1 Get ready

This manual is split in two parts each one hosted in a Github repositories:

- [the content](#)¹
- [the extra](#)²

The content is, of course, the manual itself. It's publicly available and you can fork and reuse it.

The *extra* repository is as much important as the content because it contains all you need (scripts, directories, configuration files etc) to follow the instructions.

The *extra* repository [branches](#)³, in fact, are bound to the chapters of the manual, and chapter by chapter, you will be told which branch name you should use.

To download and use the *extra* repository you need to have [Git](#)⁴ installed in your system and a [Github](#)⁵ account.

If Git is not yet installed in your system, open a terminal and install it with this command:

```
sudo apt-get install git git-core
```

Once Git is installed you are ready to download the *extra* repository:

```
cd <your-favorite-directory>  
git clone https://github.com/damko/pragmatic_virtualization_extra.git
```

This will create the `pragmatic_virtualization_extra` subdirectory inside `your-favorite-directory`.

From now on I will refer to this directory path as `$pve` so it's convenient for you to save the `$pve` variable in your bash environment by doing:

¹https://github.com/damko/pragmatic_virtualization

²https://github.com/damko/pragmatic_virtualization_extra

³https://github.com/damko/pragmatic_virtualization_extra/branches/all

⁴<http://git-scm.com/>

⁵<http://github.com>

```
cd pragmatic_virtualization_extra  
pve=$(pwd)
```

In this way by typing `cd $pve` in your terminal you will be brought in the `pragmatic_virtualization_extra` directory.



You should run these last two commands any time you open a new terminal or after a reboot.

Now that you have your *extra* on the hard drive you can switch to the specific branch in use with this command:

```
git checkout <branch-name>
```

where `branch-name` is given chapter by chapter.

1.2 Almost ready

In this manual there are a few applications that I use that might not be installed in your workstation. Please install with

```
sudo apt-get install tree
```

1.3 What's virtualization

The term *virtualization* refers to the act of creating a virtual version of something, including but not limited to a virtual computer hardware platform, operating system, storage device, or computer network resources. [Wikipedia⁶](https://en.wikipedia.org/wiki/Virtualization)

Basically with virtualization you can run virtual computers inside your own computer.

In this manual *virtualization* means *hardware virtualization* which is the creation of a virtual machine that acts like a real computer with its own operating system. The software executed in the virtual machines is separated from the underlying hardware resources and the whole virtual machine is stored in a few files located on the host system hard-drive.

1.4 Why to use virtualization

While developing an application it's a good practice to have at least a development environment and a production one.

The *development environment* is the environment used by the developer(s) to create and test the web application while the *production environment* is the final environment in which the application is published and lives.

In this case *environment* means pretty much everything, like:

- the operating system in use
- the software and services installed in the system and their versions
- the libraries used in the web application and their versions
- the tools used to deploy the application
- the tools used to test and/or monitor the application

Running an application in inhomogeneous environments can lead to strange bugs popping up in an environment and not in the other. This can be due to an amount of factors like different setups and configurations or different versions of the software in use.

1.4.1 Virtualization and environments

Virtualization provides fast and easy creation environments:

from a virtual machine template you can create as many clones as you like in minutes and you can have as many virtual machine templates as you need.

⁶<https://en.wikipedia.org/wiki/Virtualization>

Virtualization grants isolated environments:

this means that each software project can be run in isolation inside its own virtual machine and the developer's workstation is totally decoupled from the projects. To clarify consider this case: you have the application A requiring PHP 5.3 and the application B requiring PHP 5.6. Installing PHP 5.3 and 5.6 at the same time on your workstation is not easy but you can set up two different virtual machines each one with a different PHP version.

Virtualization can solve the problem of inhomogeneous environments:

with the help of some [automation tools](#) each environment of the same project can be configured with specific software and services accordingly to the application needs. In this way the developers can be sure that are using the same type and version of software in each environment

Virtualization helps working teams to work faster and better:

for at least these reasons:

- all the team members can develop in the same exact environment even when the team mates work remotely from different locations
- any change/update to the environment (like a new configuration or a new software) can be easily shared with the team members. It's just a matter of sharing a few configuration files

1.5 Virtualization tools

The most famous virtualization systems are probably VirtualBox, VMware and Docker and there are many others.

Each product has its own approach, features, pros and cons. Docker, for instance, has a very different approach while VirtualBox and VMware are quite similar.

Anyway this manual will focus only on VirtualBox (see the [Perspective](#)).

VirtualBox and two other applications called [Vagrant](#)⁷ and [Packer](#)⁸ provide what's necessary to create, clone, run and destroy your virtual machines (*a.k.a. boxes or vm or vms*) in a simple and predictable way.

In a while your workstation will have a VirtualBox virtual machine managed by Vagrant that you can use as *development environment* for your applications.

Your applications code will be stored in your workstation file-system and you will edit it locally using any editor.

The applications code will be run by the virtual machine and this explains why you don't need to install services (like Apache, Nginx or Mysql) on your workstation: they will run in the virtual machine.



If you don't have an editor you might to try to use [Sublime Text](#)⁹



It's very important that you always have the latest version of all the software listed below otherwise you can experience misbehaviors. For this reason I don't suggest to use distribution based packages but to download the code directly from the official websites.

1.6 VirtualBox

[VirtualBox](#)¹⁰ offers *full virtualization* which is a particular kind of virtualization that allows an unmodified operating system with all of its installed software to run in a special environment, on top of your existing operating system. Your existing operating system is called *host* while the virtual machines are called *guests*. This approach, often called *native virtualization*, is different from mere emulation which is typically quite slow.

VirtualBox is also different from the so-called *paravirtualization* solutions (such as Xen) which requires the guest operating system to be modified.

⁷<https://www.vagrantup.com/>

⁸<https://www.packer.io/>

⁹<http://phpbridge.org/installfest/linux#step5>

¹⁰<http://www.virtualbox.org/>

1.6.1 VirtualBox installation

Download from the [VirtualBox official website](#)¹¹ the latest package that suits at best your workstation operating system and install it:

Ex.:

```
cd /tmp
wget -c http://download.virtualbox.org/virtualbox/4.3.28/virtualbox-4.3_4.3.28-100309~Ubuntu~raring_amd64.deb
sudo dpkg -i virtualbox-4.3_4.3.28-100309~Ubuntu~raring_amd64.deb
```

1.7 Vagrant

[Vagrant](#)¹², created by [Mitchell Hashimoto](#)¹³, helps you to manage virtual boxes on top of VirtualBox (or VMWare etc.). Vagrant is considered as a wrapper for VirtualBox and it will configure the virtual machine(s) using a single configuration file named `Vagrantfile`.

Vagrant also takes care to create a directory shared between your host and the virtual machine so that you can create and edit your files locally and then run the code inside your virtual machine. More details later on.

1.7.1 Vagrant installation

Download from the [Vagrant official website](#)¹⁴ the latest package that suits at best your workstation operating system and install it:

Ex.:

```
cd /tmp
wget -c https://dl.bintray.com/mitchellh/vagrant/vagrant_1.7.2_x86_64.deb
sudo dpkg -i vagrant_1.7.2_x86_64.deb
```



Be sure that you are installing a Vagrant version > 1.5

¹¹https://www.virtualbox.org/wiki/Linux_Downloads

¹²<http://vagrantup.com/>

¹³<http://mitchellh.com/>

¹⁴<http://www.vagrantup.com/downloads.html>

1.8 Packer

Packer¹⁵, created by **Mitchell Hashimoto**¹⁶, is a tool for creating identical machine images for multiple platforms starting from a single configuration source. Packer automates the creation of any type of machine image: out of the box Packer comes with support to build images for Amazon EC2, DigitalOcean, Google Compute Engine, QEMU, VirtualBox, VMware and more.

This means that with Packer you write some configuration files (written in json format) and then you can programmatically create the same virtual machine for different environments like VirtualBox, VMware etc.

In this manual Packer will be used to create a Debian based virtual machine for VirtualBox which will be then used by Vagrant as a *template virtual machine* for the creation of your working environments. The customization of each environment will be done using an automation tool called Ansible (more details later on).

1.8.1 Packer installation

Differently from Vagrant and VirtualBox, Packer doesn't come with a debian installer so the easiest way to install it into your system is to run the `install_packer.sh` script provided in the manual extra.

Checkout the *installing_tools* branch from the *extra* repository

```
cd $pve
git checkout installing_tools
```

If you run the `tree` command you will see this:

```
damko@nitro ~/projects/pragmatic_virtualization_extra
$ tree
.
├── bin
│   └── install_packer.sh
```



Edit the `./bin/install_packer.sh` script before running it and change the variables listed at the top accordingly to your needs.

¹⁵<http://packer.io>

¹⁶<http://mitchellh.com/>

```
# environment variables: set here your preferences
tmp_dir='/tmp' # default '/tmp'
system='linux' # default 'linux'
architecture='amd64' # either 'amd64' or '386'
parent_dir='/home/damko/applications' # whatever you like
packer_dir='packer' # default 'packer'
packer_version='0.7.5' # default '0.7.5' @2015-05-20
bashrc_file='/home/damko/.bashrc_custom' # default $HOME'/.bashrc'
```

The *parent_dir* and the *bashrc_file* parameters and *packer_version* are the parameters you should take care of the most.

parent_dir

sets where you want to have it installed. I prefer to install it in my home directory. You can put any valid path

bashrc_file

sets which is the bash file in which the PATH variable should be altered so that your bash can find the Packer executable when you open a terminal

packer_version

this should be the latest packer version. You need to check it on the [Packer website](https://packer.io/downloads.html)¹⁷.

Once you are done, run the installation:

```
cd $pve
./bin/install_packer.sh
```

¹⁷<https://packer.io/downloads.html>

2. About Vagrant

2.1 How Vagrant works

You already know that Vagrant acts as a wrapper for VirtualBox (or any other supported virtualization system). This means that when you give a command to Vagrant it will route the command to the virtualization system.

For instance when you can ask Vagrant to create a new virtual machine it means that Vagrant will ask VirtualBox to create it.

The virtualization system can not create a virtual machine out of thin air because, as you can imagine, you need to install the operating system as much as you do when you want to install an operating system on an empty hard-drive.

To solve this problem Vagrant uses *boxes* which can be considered as *template virtual machines* that are given to VirtualBox so that it can clone “real” virtual machines from them.

If you are thinking that this solution just moves the problem from VirtualBox to Vagrant you are right. Something still has to create the *boxes*.

How are Vagrant “boxes” created?

In two ways:

1. they can be created from scratch by Packer, starting from an iso file (it can be any distribution you like)
2. they can be downloaded from the [Vagrant Cloud website](http://vagrantcloud.com)¹

Vagrant Cloud is a great solution: it’s simple, quick and saves you a lot of time. Basically Vagrant Cloud lists boxes created by other people who have decided to share them publicly.

You go on the Vagrant Cloud website, look for the operating system that you need, run few commands on the terminal and Vagrant takes care of everything (see section 2.4).

Perfect, isn’t it? Well, it depends.

What if you don’t trust the downloaded boxes? What if need a Linux distro not listed in Vagrant Cloud?

In these cases you have no other choice than creating your own box.

Section 2.3 will show how to create a Vagrant box using Packer starting from an iso file but, if you prefer to use Vagrant Cloud, you can move straight to section 2.4.

¹<http://vagrantcloud.com>

2.2 Conventions used in both methods

Before starting, there are some conventions used for both methods.

Ariadne will be the arbitrary *root name* used to label the Vagrant virtual machines generated starting from the Vagrant template.

This means that the development machine will be named *ariadne-dev* and the production one *ariadne-prod*.

In the both the *extra* branches there is a directory called *ariadne* having this structure:

```
|— orchestration
|   └─ vagrant
|— projects
```

This is an arbitrary skeleton and has this meaning:

the *orchestration* directory:

it's the "virtualization" directory and it contains what's needed for the management of the vms.

the *projects* directory:

it contains the source code shared with the virtual machine.

In the future you can change this structure as you like as far as you place somewhere a valid Vagrantfile and a directory to host the code but, for now, I strongly suggest you to keep it as it is.

2.3 Create a Vagrant template box from iso file

Checkout the *create_vagrant_box_from_iso* branch from the *extra* repository

```
cd $pve
git checkout create_vagrant_box_from_iso
cd packer-templates
```

You'll be following these steps:

1. creation of a box using Packer and starting from a Debian iso installer
2. importation of the Packer box in Vagrant as template box
3. creation, setup and configuration of a Vagrant virtual machine

2.3.1 Step 1: Creation of a box with Packer

Packer will download a net-install iso from the Debian servers and will use it to launch the installation process and create a virtual machine image compatible with VirtualBox and Vagrant.

Basically Packer will what you would do when you manually install Debian from a net-install cd and will adjust it for the virtualization environment.

```
cd $pve/packer-templates
tree
.
├── http
│   └── preseed.cfg
├── LICENCE
├── README.md
├── scripts
│   ├── base.sh
│   ├── cleanup.sh
│   ├── vagrant.sh
│   └── virtualbox.sh
└── ta-debian-8-jessie-virtualbox.json
```

The *ta-debian-8-jessie-virtualbox.json* file contains all the options to pass to the Debian installer to perform an unattended installation.

If you have a look at it you see that the Packer box is configured to have 1 core, 512MB ram and 10Gb hdd. That's enough for now: later on you will see how easy it is to add a core or some ram.

Before creating the vm, check if the json file is valid:

```
cd $pve/packer-templates
packer validate ta-debian-8-jessie-virtualbox.json
```

This should output *Template validated successfully*

Finally, this is the command to create the fresh vm:

```
cd $pve/packer-templates
packer build ta-debian-8-jessie-virtualbox.json
```

This last command will launch a VirtualBox window and you will be able to see the Debian installation running unattended.

You are not supposed to interact with the installation window unless you see the process stuck on a dialog window for long time and asking for user input.

When the installation process ends (it will take a while) you will see this message in the terminal:

```
==> Builds finished. The artifacts of successful builds are:
--> virtualbox-iso: 'virtualbox' provider box: debian-800-jessie.box
```

In my case, *debian-800-jessie.box* is a file about 829MB big which lives inside the packer-templates directory and contains the fresh virtual machine.

2.3.2 Step 2: Import the Packer box in Vagrant

Now that the `debian-800-jessie.box` file is ready, the next step is to tell Vagrant to import the Packer box and to make it available in the system as a template:

```
cd $pve/packer-templates
vagrant box add debian_jessie_800_64bit ./debian-800-jessie.box
```

which will output something like:

```
==> box: Adding box 'debian_jessie_800_64bit' (v0) for provider:
      box: Downloading: file:///.../packer-templates/debian-800-jessie.box
==> box: Successfully added box 'debian_jessie_800_64bit' (v0) for 'virtualbox'!
```

`debian_jessie_800_64bit` will be the name of the Vagrant template and it must be unique in the system. Imagine this Vagrant box as a *template* for a fresh Debian Jessie 64bit based virtual machine that can be used to create as many Vagrant boxes as you want.

From this moment you will need to use Packer again only if you want to create a template virtual machine based on a distribution different from Debian Jessie 8.0 64 bit.



If you like you can now delete the `debian-800-jessie.box` file.

2.3.3 Step 3: Creation and configuration of a Vagrant vm

Finally it's time to create the development environment virtual machine called *ariadne-dev*.

```
cd $pve/ariadne/orchestration/vagrant/
```

The *Vagrantfile* contained in this directory is already configured and, for now, it looks like this:

```
1  VAGRANTFILE_API_VERSION = "2"
2
3  #####
4  # VARIABLES
5
6  # Vagrant template box name
7  VG_BOX_NAME = "debian_jessie_800_64bit"
8
9  # Local virtual machine for development
10 VB_DEV_VM_NAME = "ariadne-dev"
11
12 #####
13
14 Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
15
16     # Vagrant settings for the development vm
17     config.vm.define VB_DEV_VM_NAME, primary: true do |devel|
18
19         # Vagrant box name. This the Vagrant template (created by Packer)
20         # from which the vagrant virtual machine will be generated
21         devel.vm.box = VG_BOX_NAME
22
23         # Hostname for the generated Vagrant vm
24         devel.vm.hostname = VB_DEV_VM_NAME
25
26         # Folder sharing is now disabled
27         devel.vm.synced_folder ".", "/projects", disabled: true
28
29         # VirtualBox settings
30         devel.vm.provider "virtualbox" do |vb|
31             vb.name = VB_DEV_VM_NAME
32
33             # If true it does not boot in headless mode => shows virtualbox UI.
34             # Uncomment only for troubleshooting.
35             #vb.gui = true
36
37             # Uses VBoxManage to customize the VM
38             vb.customize ["modifyvm", :id, "--memory", "512"]
39             vb.customize ["modifyvm", :id, "--cpus", "1"]
40             vb.customize ["modifyvm", :id, "--ioapic", "on"]
41         end
42     end
```



```
43
44 end
```

The Vagrantfile contains all the needed instructions that Vagrant needs to build the new vm. Soon it will be also used to describe also the production virtual machine.

Everything is ready to launch the creation of the virtual machine (which will take a couple of minutes) with the command:

```
cd $pve/ariadne/orchestration/vagrant
vagrant up
```

See what's happening:

1. Vagrant clones the VirtualBox template box into a VirtualBox virtual machine
2. Vagrant arranges some changes to the virtual machine accordingly to what set in the Vagrantfile (like the number of cores, the amount of ram, the network card MAC address, the hostname etc.)
3. Vagrant starts the virtual machine

Output:

```
Bringing machine 'ariadne-dev' up with 'virtualbox' provider...
==> ariadne-dev: Clearing any previously set forwarded ports...
==> ariadne-dev: Clearing any previously set network interfaces...
==> ariadne-dev: Preparing network interfaces based on configuration...
    ariadne-dev: Adapter 1: nat
==> ariadne-dev: Forwarding ports...
    ariadne-dev: 22 => 2222 (adapter 1)
==> ariadne-dev: Running 'pre-boot' VM customizations...
==> ariadne-dev: Booting VM...
==> ariadne-dev: Waiting for machine to boot. This may take a few minutes...
    ariadne-dev: SSH address: 127.0.0.1:2222
    ariadne-dev: SSH username: vagrant
    ariadne-dev: SSH auth method: private key
    ariadne-dev: Warning: Connection timeout. Retrying...
    ariadne-dev:
    ariadne-dev: Vagrant insecure key detected. Vagrant will automatically repla\
ce
    ariadne-dev: this with a newly generated keypair for better security.
    ariadne-dev:
    ariadne-dev: Inserting generated public key within guest...
```

```
ariadne-dev: Removing insecure key from the guest if its present...
ariadne-dev: Key inserted! Disconnecting and reconnecting using new SSH key.\
..
==> ariadne-dev: Machine booted and ready!
```

After this you will eventually see some downloads starting and VirtualBox Guest Additions being updated but it will completely automatic.

Anyway the virtual machine is ready and running.

You can now login in the virtual machine via ssh with:

```
cd $pve/ariadne/orchestration/vagrant
vagrant ssh
```

2.4 Create a Vagrant template box from Vagrant Cloud

Checkout the `create_vagrant_box_from_cloud` branch from the *extra* repository

```
cd $pve
git checkout create_vagrant_box_from_cloud
```

What has been made with Packer can be done with [Vagrant Cloud](#)². If you look for “Debian Jessie” in the “Discover Boxes” section you will get many results: the first one that I have found claiming to produce a clean and minimal Debian Jessie 64 bit is the ARCO Research Group [deb/jessie-amd64](#)³

To use this box run:

```
cd $pve/ariadne/orchestration/vagrant
vagrant init deb/jessie-amd64
```

Output:

```
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant
```

This command creates a Vagrantfile which, leaving out the comments, looks like this:

```
1 VAGRANTFILE_API_VERSION = "2"
2
3 Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
4
5   config.vm.box = "xezpeleta/wheezy64"
6
7 end
```

If you know want to have the same virtual machine as the previous chapter (same RAM, cores, name etc.) you should replace the Vagrantfile with the one provided in the extra and be sure that there is no other ariadne-dev VirtualBox machine.



You can check your virtual machines list by running the command `virtualbox`

²<http://vagrantcloud.com/>

³<https://atlas.hashicorp.com/deb/boxes/jessie-amd64>

```
cd $pve/ariadne/orchestration/vagrant
mv Vagrantfile_custom Vagrantfile
```

The only extra step that you need to do is:

```
cd $pve/ariadne/orchestration/vagrant
vagrant up
```

Output:

```
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Box 'xezpeleta/wheezy64' could not be found. Attempting to find and\
install...
    default: Box Provider: virtualbox
    default: Box Version: >= 0
==> default: Loading metadata for box 'xezpeleta/wheezy64'
    default: URL: https://vagrantcloud.com/xezpeleta/wheezy64
==> default: Adding box 'xezpeleta/wheezy64' (v1.0.0) for provider: virtualbox
    default: Downloading: https://vagrantcloud.com/xezpeleta/boxes/wheezy64/vers\
ions/1/providers/virtualbox.box
    default: Progress: 1% (Rate: 495k/s, Estimated time remaining: 0:16:08)

[...]

==> default: Successfully added box 'xezpeleta/wheezy64' (v1.0.0) for 'virtualbo\
x'!
==> default: Importing base box 'xezpeleta/wheezy64'...

[...]

==> default: Machine booted and ready!
```

The `vagrant up` command downloads the box, stores it in your hard drive, makes it available for VirtualBox and then, finally, orders to VirtualBox to create a virtual machine by cloning the downloaded box and you end up with the working virtual machine. Brilliant!

That's it. The virtual machine is ready and running.

You can now login in the virtual machine via ssh with:

```
cd $pve/ariadne/orchestration/vagrant  
vagrant ssh
```

2.5 Some considerations about Vagrant

2.5.1 Same result

As you can see, it doesn't matter which way you choose (Packer or Vagrant Cloud) you end up with the same result.

2.5.2 Vagrantfile position

I want also to stress on something that you probably have already noticed: all the Vagrant commands are anticipated by this command:

```
cd $pve/ariadne/orchestration/vagrant
```

This is because Vagrant expects to find the Vagrantfile in the directory from which you are typing the commands.

2.5.3 The .vagrant directory

Now that the Vagrant virtual machine has been created you can see a new hidden directory \$pve/ariadne/orchestration/vagrant/.vagrant

```
tree $pve/ariadne/orchestration/vagrant/.vagrant
```

```
.
├── machines
│   └── ariadne-dev
│       └── virtualbox
│           ├── action_provision
│           ├── action_set_name
│           ├── id
│           ├── index_uuid
│           └── synced_folders
```

In this directory Vagrant saves important information about the newly created vm.



Don't delete the .vagrant folder!

2.5.4 start and stop a vm

If you want to stop or reboot the development virtual machine from your workstation terminal, be sure to be in the vagrant directory and then run:

```
vagrant halt
```

or

```
vagrant reload
```

To start the development virtual machine after you stop it:

```
vagrant up
```

3. Vagrant administration

3.1 Listing virtual machines

It should be clear by now that Vagrant is a virtual machine manager therefore there are a number of commands that you can run to handle your vagrant virtual machines:

```
vagrant -h
```

Output:

```
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
hostsupdater	
init	initializes a new Vagrant environment by creating a Vagrantfile
login	log in to HashiCorp's Atlas
package	packages a running vagrant environment into a box
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
rebuild	
reload	restarts vagrant machine, loads new Vagrantfile configuration
resume	resume a suspended vagrant machine
share	share your Vagrant environment with anyone in the world

machine	ssh	connects to machine via SSH
	ssh-config	outputs OpenSSH valid configuration to connect to the m\
	status	outputs status of the vagrant machine
	suspend	suspends the machine
	up	starts and provisions the vagrant environment
	vbguest	
	version	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``.

For instance you can list the Vagrant templates (or *boxes*) with this:

```
vagrant box list
```

You should see something like:

```
deb/jessie-amd64      (virtualbox, 2.0)
debian_jessie_800_64bit (virtualbox, 0)
```

You can also list the Vagrant vms managed by Vagrant with:

```
vagrant global-status
```

Output:

```
4c47439 ariadne-dev virtualbox    running /home/damko/projects/pragmatic_v\
irtualization_extra/ariadne/orchestration/vagrant
```

You can also ask VirtualBox to list the virtual machines present in its registry.

```
vboxmanage list vms
```

Output:

```
"ariadne-dev" {472a10a0-6958-4e62-9219-54d50bee4928}
```



The output of the last two commands (`vagrant global-status` and `vboxmanage list vms`) is the same (*ariadne-dev* for both) but this is just a coincidence. There is no equivalence between the two commands.

If you prefer you can use the VirtualBox GUI to list the register vms and to deal with them:

```
virtualbox
```



If you need to destroy a virtual machine made with Vagrant it's better to use the command `vagrant destroy` than to delete the virtual machine from the VirtualBox interface. The `vagrant destroy`, in fact, will deal with the `.vagrant` folder and it will update its content and keep it clean. VirtualBox obviously will not.

3.1.1 Where are your templates and vms?

The Vagrant template boxes are located on your workstation hard-disk

```
ls -lh ~/.vagrant.d/boxes
```

Output:

```
drwxr-xr-x 3 damko damko 4.0K 2014-11-17 20:27:33 debian_jessie_800_64bit
```

You can check the occupied space with:

```
du -sh ~/.vagrant.d/boxes/*
```

Output:

```
442M    /home/damko/.vagrant.d/boxes/debian_jessie_800_64bit
```

The Vagrant virtual machines will be saved in the VirtualBox default directory (in my case `~/vms/vbox/`)

```
ls -lh ~/vms/vbox
```

Output:

```
drwx----- 3 damko damko 4.0K 2014-11-26 16:56:12 ariadne-dev
```



you can change the VirtualBox default directory by editing the “~/Virtualbox/VirtualBox.xml” file. The value is contained in the “defaultMachineFolder” parameter of the “SystemProperties” tag.

Occupied space:

```
du -sh ~/vms/vbox/*
```

Output:

```
1.2G    /home/damko/vms/vbox/ariadne-dev
```

3.2 Improving the Vagrantfile

The Vagrantfile used till now is quite simple and can be improved to do more.

The Vagrantfile can be written to configure several virtual machines and to host many different configurations (like network settings, host name etc.) for each one.

Checkout the *create_vagrant_box_from_iso* branch from the *extra* repository:

```
cd $pve
git checkout improving_vagrantfile
```

This branch provides an improved version of the previous Vagrantfile. It looks like this:

```
1  VAGRANTFILE_API_VERSION = "2"
2
3  #####
4  # VARIABLES
5
6  # Vagrant template box name
7  VG_BOX_NAME = "debian_jessie_800_64bit"
8
9  # Local virtual machine for development
10 VB_DEV_VM_NAME = "ariadne-dev"
11 VB_DEV_VM_IP = "192.168.51.10"
12
13 #####
14
15 Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
16
17     # Vagrant settings for the development vm
18     config.vm.define VB_DEV_VM_NAME, primary: true do |devel|
19
20         # Vagrant box name. This the Vagrant template (created by Packer) from w\
21 hich the vagrant virtual machine will be generated
22         devel.vm.box = VG_BOX_NAME
23
24         # Hostname for the generated Vagrant vm
25         devel.vm.hostname = VB_DEV_VM_NAME
26
27         # Sets the IP
28         devel.vm.network "private_network", ip: VB_DEV_VM_IP
29
```

```

30     # NFS settings
31     devel.vm.synced_folder "../../projects", "/vagrant", type: "nfs"
32
33     # VirtualBox settings
34     devel.vm.provider "virtualbox" do |vb|
35         vb.name = VB_DEV_VM_NAME
36
37         ## If true it does not boot in headless mode => shows virtualbox UI.\
38     Uncomment only for troubleshooting.
39         #vb.gui = true
40
41         # Uses VBoxManage to customize the VM
42         vb.customize ["modifyvm", :id, "--memory", "512"]
43         vb.customize ["modifyvm", :id, "--cpus", "1"]
44         vb.customize ["modifyvm", :id, "--ioapic", "on"]
45     end
46 end
47
48 end

```

You can see the differences with the previous branch using your favorite `diff` tool.

If you don't have a favorite diff tool yet I suggest [meld](http://meldmerge.org/)¹. Once `meld` has been installed you can configure your git preferences to use `meld` with these commands:

```

git config --global diff.guitool meld
git config --global difftool.prompt false

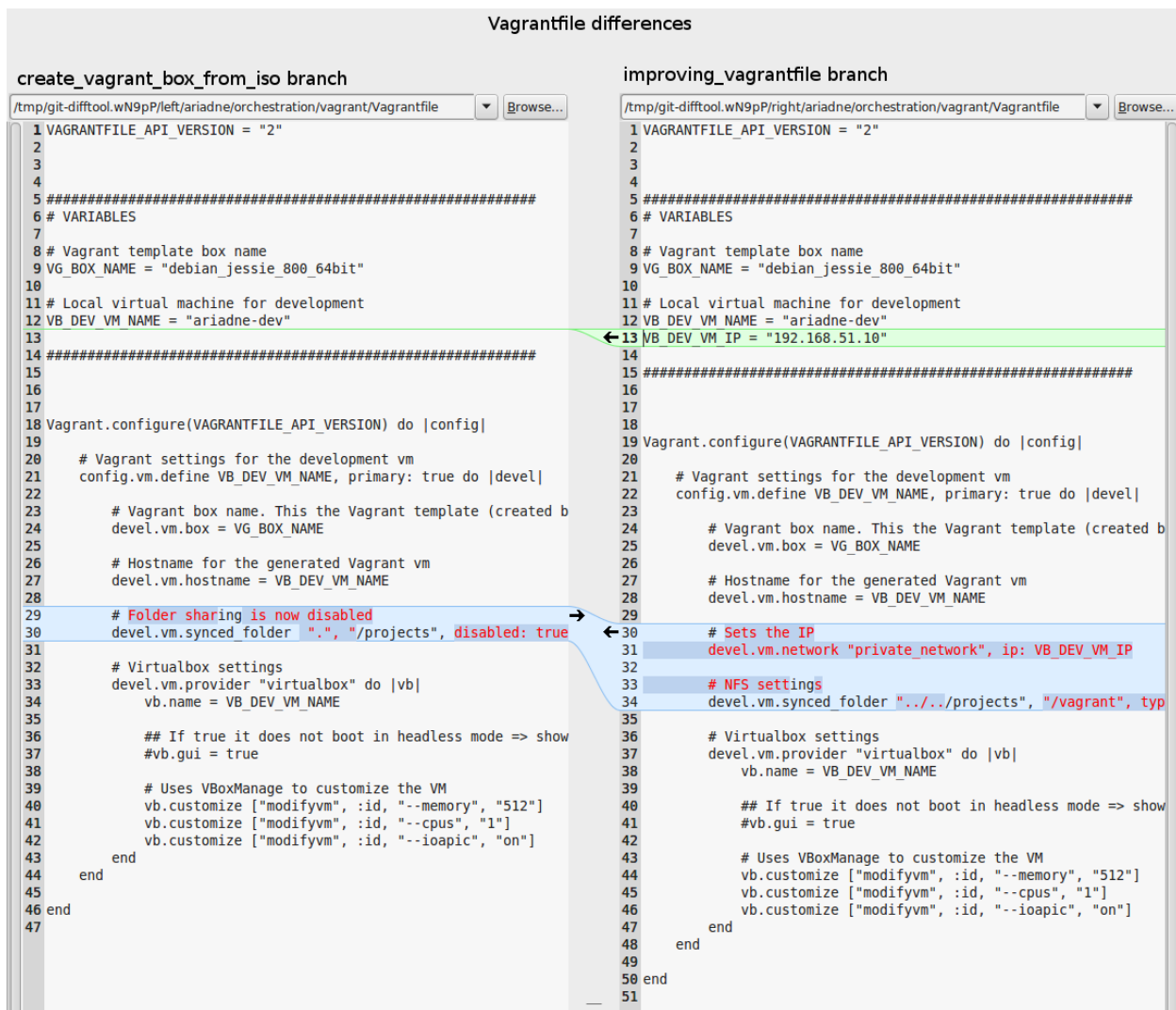
```

Then you can see the differences between the previous and the current branch:

```
git difftool -g -d create_vagrant_box_from_iso
```

which look like this:

¹<http://meldmerge.org/>



Vagrantfile differences

3.2.1 Conventions in use

At the beginning of the *Vagrantfile* there are some variable declarations that can be freely declared and that are used later on in the Vagrantfile.

Declaring variables is not mandatory, meaning that you can hardcode the values, but I think it's highly suggested because of re-usability.

I also used some conventions that you can definitely drop if you don't like them:

- variables starting with VG = Vagrant
- variables starting with VB = VirtualBox
- the virtual machine hostname and VirtualBox virtual machine name match

- the virtual machine name is also used to define each section of the Vagrantfile, like this
`config.vm.define VB_DEV_VM_NAME, primary: true do |devel|`

3.2.2 Virtual machine network settings

Especially if you don't have a DHCP server running in your network, it can be convenient to automatically set the IP for the virtual machine.

Vagrant can take care of this and to do so I declare the variable `VB_DEV_VM_IP`:

```
VB_DEV_VM_IP = "192.168.51.10"
```

and then I use that variable in line 28

```
# Sets the IP
devel.vm.network "private_network", ip: VB_DEV_VM_IP
```



if there is already a machine using the IP address 192.168.51.10 in your network (i.e. another vm) then you must change the “`VB_DEV_VM_IP`” variable and set an IP that **does not belong** to your workstation network class.

The same thing happens for the host name. Variable:

```
VB_DEV_VM_NAME = "ariadne-dev"
```

and used in the configuration line number 25:

```
# Hostname for the generated Vagrant vm
devel.vm.hostname = VB_DEV_VM_NAME
```

With this configuration I expect that the virtual machine hostname will be *ariadne-dev* and it will take the IP *192.168.51.10*.



Avoid “dots” in the hostname²

It's better to add this information to the workstation hosts file `/etc/hosts`:

²<http://serverfault.com/questions/229331/can-i-have-dots-in-a-hostname>

```
sudo echo "192.168.51.10    ariadne-dev" >> /etc/hosts
```

3.2.3 Directory sharing

As already said, the code will be stored on the workstation file-system but it will run on the virtual machine. This means that the virtual machine needs to connect to the workstation file-system through a directory sharing service. The most comfortable way to do so in Linux is by using the NFS service.

The NFS service is the only service that you really need to install on the workstation. To install NFS:

```
sudo apt-get install nfs-kernel-server
```

This configuration line (31)

```
devel.vm.synced_folder "../../projects", "/vagrant", type: "nfs"
```

tells Vagrant which is the directory that should be shared with the virtual machine right each time the virtual machine is booted. Vagrant will automatically take care of the configuration of the NFS shares (which are set in the `/etc/exports` file of your workstation).

In this case the `$pve/ariadne/projects` directory will be mounted on the `/vagrant` directory of the virtual machine file-system. As you can see the source directory path is relative to the directory containing the Vagrantfile (`../../projects`).

3.2.4 How to apply the changes

To apply the changes added to the Vagrantfile, stop the virtual machine and restart it:

```
cd $pve/ariadne/orchestration/vagrant
vagrant halt
vagrant up
```

Output:


```

Bringing machine 'ariadne-dev' up with 'virtualbox' provider...
==> ariadne-dev: Clearing any previously set forwarded ports...
==> ariadne-dev: Clearing any previously set network interfaces...
==> ariadne-dev: Preparing network interfaces based on configuration...
    ariadne-dev: Adapter 1: nat
    ariadne-dev: Adapter 2: hostonly
==> ariadne-dev: Forwarding ports...
    ariadne-dev: 22 => 2222 (adapter 1)
==> ariadne-dev: Running 'pre-boot' VM customizations...
==> ariadne-dev: Booting VM...
==> ariadne-dev: Waiting for machine to boot. This may take a few minutes...
    ariadne-dev: SSH address: 127.0.0.1:2222
    ariadne-dev: SSH username: vagrant
    ariadne-dev: SSH auth method: private key
    ariadne-dev: Warning: Connection timeout. Retrying...
==> ariadne-dev: Machine booted and ready!
GuestAdditions 4.3.16 running --- OK.
==> ariadne-dev: Checking for guest additions in VM...
==> ariadne-dev: Setting hostname...
==> ariadne-dev: Configuring and enabling network interfaces...
==> ariadne-dev: Exporting NFS shared folders...
==> ariadne-dev: Preparing to edit /etc/exports. Administrator privileges will be required...
[sudo] password for damko:

```

As you can see NFS requires root privileges to modify the `/etc/exports` file so Vagrant will pause the booting process and ask for your user's system password



your user is supposed to be in the sudoers group

Then Vagrant will continue:

```

nfsd running
==> ariadne-dev: Mounting NFS shared folders...
==> ariadne-dev: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> ariadne-dev: to force provisioning. Provisioners marked to run always will still run.

```

3.2.5 How to test the changes

Login to the virtual machine via SSH with:

```
vagrant ssh
```

or using the virtual machine hostname:

```
ssh vagrant@ariadne-dev  
#provide "vagrant" as password
```

and check if the changes have been applied correctly.

3.2.5.1 Checking NFS

```
vagrant@ariadne-dev:~$ cat /vagrant/testfile
```

Output:

```
I'm a test file
```

This looks good. The *projects* directory contains *testfile* and it's also visible inside the virtual machine.

3.2.5.2 Checking the hostname

```
vagrant@ariadne-dev:~$ cat /etc/hostname
```

Output:

```
ariadne-dev
```

This looks good.

3.2.5.3 Ethernet

```
vagrant@ariadne-dev:~$ sudo ifconfig
```

Output:

```
eth0      Link encap:Ethernet  HWaddr 08:00:27:2e:98:c9
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe2e:98c9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:916 errors:0 dropped:0 overruns:0 frame:0
          TX packets:602 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:87745 (85.6 KiB)  TX bytes:71271 (69.6 KiB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:06:d4:da
          inet addr:192.168.51.10  Bcast:192.168.51.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe06:d4da/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:74 errors:0 dropped:0 overruns:0 frame:0
          TX packets:22 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9996 (9.7 KiB)  TX bytes:2100 (2.0 KiB)
```

This is not exactly what was expected because there are 2 network interfaces. This is due to the fact that when the first `vagrant up` was run there was no ethernet configuration set in the Vagrantfile and so an `eth0` device was created by default.

I decided to start with a very minimalistic Vagrantfile, for educational purposes, to show that virtualization is not a magic thing: bad configuration can often lead to unexpected results that you need to sort out step by step.

To solve the little issue with the two network interfaces you can either manually fix the `/etc/network/interfaces` file or you can be lazy and destroy and rebuild the virtual machine using the new Vagrantfile.

3.3 Destroy your virtual machine

I told you that virtualization is a great thing because you can easily create, destroy and rebuild a virtual machine with basically no effort.

Let's prove it by destroying and then rebuilding the virtual machine.

3.3.1 How to identify the correct virtual machine

Before destroying a virtual machine it's important to know its identification name so that you can be sure that you are deleting the right one.

Currently we have the Vagrant/VirtualBox virtual machine and the Vagrant template box and they have different names that are written in the Vagrantfile:

```
# Vagrant template box name
VG_BOX_NAME = "debian_jessie_800_64bit"

# Local virtual machine for development
VB_DEV_VM_NAME = "ariadne-dev"
```

3.3.2 Destroy the Vagrant virtual machine

Vagrant can be used to destroy the newly created *ariadne-dev*



The virtual machine to destroy must be defined in the Vagrantfile present in the current directory

```
cd $pve/ariadne/orchestration/vagrant/
vagrant destroy ariadne-dev
```

Output:

```
ariadne-dev: Are you sure you want to destroy the 'ariadne-dev' VM? [y/N]
==> ariadne-dev: Forcing shutdown of VM...
==> ariadne-dev: Destroying VM and associated drives...
```

If you run again:

```
vboxmanage list vms
```

or

```
vagrant global-status
```

you should not see *ariadne-dev* anymore.

If you try to destroy a virtual machine that is not listed in the Vagrantfile, like this

```
vagrant destroy precise64_default_1396556373306_80830
```

you get

The machine with the name 'precise64_default_1396556373306_80830' was not found \ configured for this Vagrant environment.

3.3.3 Destroy the Vagrant template box

One day the Vagrant template box will become outdated and you might want to delete it to replace it with a new one.

```
vagrant box remove debian_jessie_800_64bit
```

This command can be executed from any directory because it doesn't need the Vagrantfile.

Output:

```
Removing box 'debian_jessie_800_64bit' (v0) with provider 'virtualbox'...
```

Run again:

```
vagrant box list
```

Now you should not see `debian_jessie_800_64bit` in the list.



After deleting a Vagrant template box don't forget to go through every single Vagrantfile and update the `VG_BOX_NAME` variable with the new template name.

3.4 Rebuild the Vagrant virtual machine

Once you have a well configured Vagrantfile and a ready Vagrant template box all you need to do to rebuild you Vagrant virtual machine is to run

```
vagrant up
```

3.5 How to build another copy

If you want to create a clone of *ariadne* you just need to copy the Vagrantfile in another directory and modify its variables

```
VB_DEV_VM_NAME = "newvm-dev"  
VB_DEV_VM_IP   = "192.168.51.xx"
```

then run `vagrant up`. That's it.

4. Automation tools

4.1 Automation tools

Besides VirtualBox, Vagrant and Packer there are additional tools that can help setting up your virtual environment.

After a fresh virtual machine has been created the additional system configurations can be done manually or with an *automation tool*.

The most famous automation tools (also called *provisioning software*) currently available are [Puppet](http://www.puppetlabs.com)¹, [Chef](http://www.opscode.com)² and [Ansible](http://www.ansible.com)³.

These tools are specifically designed to install any kind of software package, service, script and configuration file. They can be used both on virtual machine and regular computers.

Provisioning virtual machines grants that any other virtual machine created using the same configuration files will be identical to the others and it's the way to programmatically create virtual machines with the same quality.

This approach is very useful when you work in a team in which each developer has his/her own development virtual machine but also when you strive to have a development environment homogeneous with the production one.

A well configured provisioning tool also gives you the freedom to destroy and rebuild your boxes in minutes or to painlessly switch your hosting provider at any time.

4.1.1 The Suggestotron requirements

The Suggestotron web application, as shown in the [requirements](#)⁴ needs PHP, a webserver and a mysql database to run plus some other accessories.

This manual will use Ansible to automatically install and configure the required software both in the development and in the production environment.

¹<http://www.puppetlabs.com>

²<http://www.opscode.com>

³<http://www.ansible.com>

⁴<http://phpbridge.org/intro-to-php/>

4.2 Ansible

Ansible is a powerful automation tool that doesn't require custom scripting or custom code or software agents pre-installed in the virtual machine.

Its simple configuration syntax, the ability to execute commands via ssh without the need of any software agent make Ansible very appealing.

It also doesn't need a database or daemons running and, once installed on your workstation, you can manage an entire fleet of remote machines from that central point.

4.2.1 Ansible installation

This time it's safe to use the distro installer:

```
sudo apt-get install software-properties-common
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible
```

Ansible, by default, decorates its output with the funny “cow” ASCII art.

```
-----
      \   ^__^
      \  (oo)\_______
         (__)\\       )\/\
            ||----w |
            ||     ||
```

It's funny but it's also “noisy” so it's better to disable it with this command:

```
sudo sed 's/#nocows/nocows/g' -i /etc/ansible/ansible.cfg
```

4.2.2 Ansible and Vagrant

Checkout the *ansible_apache_php* branch from the *extra* repository

```
cd $pve
git checkout ansible_apache_php
```

Vagrant can be configured to be a wrapper also for Ansible.

To do so the Vagrantfile has to contain these instructions (have a look at your Vagrantfile):


```
# Use Ansible as automation tool
devel.vm.provision "ansible" do |ansible|

  ansible.playbook = "../ansible/ariadne-dev-playbook.yml"

  # Run commands as root
  ansible.sudo = true
end
```

The only variable to take care of is *ansible.playbook* which says where Ansible can find its own configuration file.

4.3 Ansible configuration

I assume that you have followed step by step the manual and that you destroyed and rebuild the *ariadne-dev* virtual machine.

If you run `vagrant up` the virtual machine will start and the output will not be different from the one you are used to, which looks like this:

```
vagrant up
```

Output:

```
Bringing machine 'ariadne-dev' up with 'virtualbox' provider...
==> ariadne-dev: Clearing any previously set forwarded ports...
==> ariadne-dev: Clearing any previously set network interfaces...
==> ariadne-dev: Preparing network interfaces based on configuration...
    ariadne-dev: Adapter 1: nat
    ariadne-dev: Adapter 2: hostonly
==> ariadne-dev: Forwarding ports...
    ariadne-dev: 22 => 2222 (adapter 1)
==> ariadne-dev: Running 'pre-boot' VM customizations...
==> ariadne-dev: Booting VM...
==> ariadne-dev: Waiting for machine to boot. This may take a few minutes...
    ariadne-dev: SSH address: 127.0.0.1:2222
    ariadne-dev: SSH username: vagrant
    ariadne-dev: SSH auth method: private key
    ariadne-dev: Warning: Connection timeout. Retrying...
==> ariadne-dev: Machine booted and ready!
GuestAdditions 4.3.16 running --- OK.
==> ariadne-dev: Checking for guest additions in VM...
==> ariadne-dev: Setting hostname...
==> ariadne-dev: Configuring and enabling network interfaces...
==> ariadne-dev: Exporting NFS shared folders...
==> ariadne-dev: Preparing to edit /etc/exports. Administrator privileges will be required...
[sudo] password for damko:
nfsd running
==> ariadne-dev: Mounting NFS shared folders...
==> ariadne-dev: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> ariadne-dev: to force provisioning. Provisioners marked to run always will still run.
```



If you destroyed the virtual machine but you didn't rebuild it yet you will see a different result because Vagrant automatically starts Ansible at the first `vagrant up`.



If this is your case I suggest to go back to the previous branch (*improving_vagrantfile*) and rebuild the machine so that you won't see anything difference from what explained.

The *ansible* directory in the *extra* contains the scripts and the configuration files for Ansible required to install the following softwares:

- Apache
- PHP5
- Composer
- Psysh
- Xdebug

which is pretty much all the software you need to run and develop Suggestotron.

```
tree $pve/ariadne/orchestration/
```

```
├─ ansible
│   ├── ariadne-dev-playbook.yml
│   ├── tasks
│   │   ├── apache2.yml
│   │   ├── composer.yml
│   │   ├── init.yml
│   │   ├── php5.yml
│   │   ├── psysh.yml
│   │   └─ xdebug.yml
│   └─ vars.yml
└─ vagrant
    └─ Vagrantfile
```

ariadne-dev-playbook.yml is the file name written in the Vagrantfile, remember?

```
ansible.playbook = "../ansible/ariadne-dev-playbook.yml"
```

This is the Ansible main file, also known as *playbook*, it's written in [YAML format](http://en.wikipedia.org/wiki/YAML)⁵ and it looks like this:

⁵<http://en.wikipedia.org/wiki/YAML>

```
1  ---
2  - hosts: all
3
4  # Tells Ansible to run the commands with sudo
5  sudo: true
6
7  # File containing the variables used in Ansible scripts
8  vars_files:
9    - vars.yml
10
11 # List of actions to execute before any task
12 pre_tasks:
13   - name: playbook.yml | Run apt-get update if it was run last time more than \
14 12 hours ago
15     apt: update_cache=yes cache_valid_time=43200
16
17 # List of tasks that will be executed sequentially
18 tasks:
19
20 # initial setup: it installs common packages like 'curl', 'vim' ...
21 - include: tasks/init.yml
22
23 # installs PHP5
24 - include: tasks/php5.yml
25
26 # installs the Apache2 web server
27 - include: tasks/apache2.yml
28
29 # install Xdebug that can be used to debug the PHP code in your editor
30 - include: tasks/xdebug.yml
31
32 # installs Composer, a tool for dependency management in PHP
33 - {
34   include: tasks/composer.yml,
35   # these variable declarations override the variables declared in vars.yml
36   composer_path: "/usr/local/bin/composer",
37   composer_keep_updated: true
38 }
39
40 # installs Psysh a runtime console for PHP used in the PHPBridge course
41 - include: tasks/psysh.yml
42
```

```
43  # Handlers are used in Ansible tasks with this syntax example
44  # notify: restart apache2
45  # which means "after running this task execute
46  # `service: name=apache2 state=restarted enabled=yes`"
47  handlers:
48  - name: restart apache2
49    service: name=apache2 state=restarted enabled=yes
50
51  - name: restart php5-fpm
52    service: name=php5-fpm state=restarted enabled=yes
53
54  - name: restart mysql
55    service: name=mysql state=restarted enabled=yes
```

Ansible will be executed via Vagrant with the command `vagrant provision`

Ansible will start executing the tasks listed below the *tasks:* line in sequential order.

In order to keep things clean, each Ansible task is kept in a separated file stored in the `tasks` subdirectory and included in the playbook.

Please have a close look at the *playbook:* the YAML syntax is human readable and I tried to add comments to show what happens.

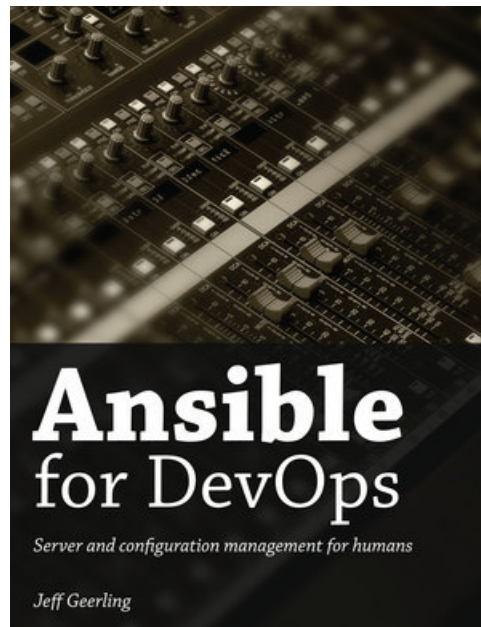


Deepening Ansible is beyond the purpose of this manual because it's a vary huge topic. If you want to know more (I know you want) I suggest to read the awesome [Ansible online documentation](http://docs.ansible.com/)⁶ and to purchase the excellent ebook [Ansible for DevOps](https://leanpub.com/ansible-for-devops)⁷ written by [Jeff Geerling](https://twitter.com/geerlingguy)⁸ (yes even if it's currently not finished, 94% at the time of writing).

⁶<http://docs.ansible.com/>

⁷<https://leanpub.com/ansible-for-devops>

⁸<https://twitter.com/geerlingguy>



Ansible for devops

4.3.1 Run Ansible

To run Ansible simply let Vagrant do it for you:

```
vagrant provision
```

Ansible output:

```
PLAY [all] *****

GATHERING FACTS *****
ok: [ariadne-dev]

TASK: [playbook.yml | Run apt-get update if it was run last time more than 12 hours ago] ***
ok: [ariadne-dev]

TASK: [init.yml | Install Sys Packages] *****
changed: [ariadne-dev] => (item=curl,vim,python-pycurl,python-apt,aptitude,multi\
tail,tree)

TASK: [init.yml | run Security Upgrade] *****
changed: [ariadne-dev]

TASK: [init.yml | Ensure NTP is installed] *****
ok: [ariadne-dev]

TASK: [init.yml | Ensure NTP is running] *****
changed: [ariadne-dev]

TASK: [php5.yml | Check if php5-fpm is already installed] *****
failed: [ariadne-dev] => {"changed": true, "cmd": ["dpkg", "-s", "php5-fpm"], "delta": "0:00:00.010319", "end": "2014-11-29 14:44:39.151516", "failed": true, "failed_when_result": true, "rc": 1, "start": "2014-11-29 14:44:39.141197", "stdout_lines": []}
stderr: dpkg-query: package 'php5-fpm' is not installed and no information is available
Use dpkg --info (= dpkg-deb --info) to examine archive files,
and dpkg --contents (= dpkg-deb --contents) to list their contents.
...ignoring

TASK: [php5.yml | Add php5 repository] *****
```

changed: [ariadne-dev]

TASK: [php5.yml | Add php5 repository key] *****
changed: [ariadne-dev]

TASK: [php5.yml | Update apt] *****
ok: [ariadne-dev]

TASK: [php5.yml | Install PHP Packages] *****
changed: [ariadne-dev] => (item=php5-fpm,php5-curl,php5-cli)

TASK: [apache2.yml | Install Apache Packages] *****
changed: [ariadne-dev] => (item=apache2,libapache2-mod-php5,libapache2-mod-macro)

TASK: [apache2.yml | Enable Apache rewrite module] *****
changed: [ariadne-dev]

TASK: [xdebug.yml | Install xdebug] *****
changed: [ariadne-dev] => (item=php5-xdebug)

TASK: [Ensure curl is installed (Debian).] *****
ok: [ariadne-dev]

TASK: [Install Composer into the current directory.] *****
changed: [ariadne-dev]

TASK: [Move Composer into globally-accessible location.] *****
changed: [ariadne-dev]

TASK: [Update Composer to latest version (if configured).] *****
ok: [ariadne-dev]

TASK: [psysh.yml | Install psysh for the vagrant user] *****
changed: [ariadne-dev]

TASK: [psysh.yml | Ensure that ~/.composer/vendor/bin is in PATH] *****
changed: [ariadne-dev]

PLAY RECAP *****
ariadne-dev : ok=20 changed=14 unreachable=0 failed=0

During the execution Ansible outputs the exit status (changed, Ok, skipped ...) of every single task so that you can see if something goes wrong.

If a task fails Ansible throws an error and the execution stops.

At this point *ariadne-dev* has *almost* all the required software installed.

You can check the list of the running services in this way:

```
cd $pve/ariadne/orchestration/vagrant
#logging into the virtual machine
vagrant ssh
```

```
vagrant@ariadne-dev:~$ sudo service --status-all
```

Output:

```
[ + ] acpid
[ + ] apache2
[ + ] atd
[ - ] bootlogs
[ ? ] bootmisc.sh
[ ? ] checkfs.sh
[ ? ] checkroot-bootclean.sh
[ - ] checkroot.sh
[ - ] console-setup
[ + ] cron
[ + ] exim4
[ - ] hostname.sh
[ ? ] hwclock.sh
[ - ] kbd
[ - ] keyboard-setup
[ ? ] killprocs
[ ? ] kmod
[ - ] lvm2
[ - ] motd
[ ? ] mountall-bootclean.sh
[ ? ] mountall.sh
[ ? ] mountdevsubfs.sh
[ ? ] mountkernfs.sh
[ ? ] mountnfs-bootclean.sh
[ ? ] mountnfs.sh
[ ? ] mtab.sh
[ ? ] networking
[ + ] nfs-common
```

```
[ + ] ntp
[ + ] php5-fpm
[ - ] procps
[ ? ] rc.local
[ - ] rmnologin
[ + ] rpcbind
[ - ] rsync
[ + ] rsyslog
[ ? ] sendsigs
[ + ] ssh
[ - ] sudo
[ + ] udev
[ ? ] udev-mtab
[ ? ] umountfs
[ ? ] umountnfs.sh
[ ? ] umountroot
[ - ] urandom
[ + ] vboxadd
[ + ] vboxadd-service
[ - ] vboxadd-x11
```

You have probably noticed that Mysql is missing because the playbook doesn't have any instructions about that. See the next chapter.

4.4 Ansible tools

Configuring Ansible can be hard mostly because configuring a system in a proper way is a hard job. Fortunately there are very useful tools available for free coming in help.

4.4.1 Ansible-galaxy

[Ansible Galaxy](https://galaxy.ansible.com)⁹ is a hub for finding, reusing, and sharing the best Ansible content. Basically it's a public repository for *Ansible roles*.

An *Ansible Role* is a way to organize tasks, variables and templates in a specific way with the aim to reuse it in many playbooks.

You are now going to install Mysql using [Benno Joy's Mysql role](#)¹⁰.

The role page provides a lot of information about how to use the role but the basics behind any Ansible Galaxy role is simple:

1. install the role locally
2. add it to your playbook

To install the role on your workstation

```
sudo ansible-galaxy install bennojoy.mysql
```

The role will be copied in `/etc/ansible/roles/`, path where you can also store your own Ansible roles.

Checkout the `ansible_apache_php_mysql` branch from the *extra* repository

```
cd $pve
git checkout ansible_apache_php_mysql
```

The `ariadne-dev-playbook.yml` now contains these additional lines

⁹<https://galaxy.ansible.com>

¹⁰<https://galaxy.ansible.com/list#/roles/1>

```

roles:
  # installs Mysql using Benno Joy's role
  - {
      role: bennojoy.mysql,

      # mysql root password
      mysql_root_db_pass: root,

      # mysql port open only for IP 127.0.0.1
      mysql_bind_address: 127.0.0.1,

      # do not create any db
      mysql_db: [],

      # no additional mysql user
      mysql_users: []
    }

```

They are telling Ansible to install Mysql using the variables listed below the `role: bennojoy.mysql` line.

Now run once again:

```

$ pve/ariadne/orchestration/vagrant/
vagrant provision

```

Output:

```

==> ariadne-dev: Running provisioner: ansible...

```

```

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

```

```

[...]
```

```

TASK: [bennojoy.mysql | Add the OS specific variables] *****\
***

```

```

ok: [ariadne-dev]

```

```

TASK: [bennojoy.mysql | Install the mysql packages in Redhat derivatives] **\
***

```

```

skipping: [ariadne-dev]

```

```
TASK: [bennojoy.mysql | Install the mysql packages in Debian derivatives] **\
***
changed: [ariadne-dev] => (item=python-selinux,mysql-server,python-mysqldb)

TASK: [bennojoy.mysql | Copy the my.cnf file] *****\
***
changed: [ariadne-dev]

TASK: [bennojoy.mysql | Create the directory /etc/mysql/conf.d] *****\
***
ok: [ariadne-dev]

TASK: [bennojoy.mysql | Start the mysql services Redhat] *****\
***
ok: [ariadne-dev]

TASK: [bennojoy.mysql | update mysql root password for all root accounts] **\
***
changed: [ariadne-dev] => (item=ariadne-dev)
changed: [ariadne-dev] => (item=127.0.0.1)
changed: [ariadne-dev] => (item=:1)
changed: [ariadne-dev] => (item=localhost)

TASK: [bennojoy.mysql | update mysql root password for all root accounts] **\
***
skipping: [ariadne-dev] => (item=127.0.0.1)
skipping: [ariadne-dev] => (item=:1)
skipping: [ariadne-dev] => (item=localhost)

TASK: [bennojoy.mysql | copy .my.cnf file with root password credentials] **\
***
changed: [ariadne-dev]

TASK: [bennojoy.mysql | ensure anonymous users are not in the database] ****\
***
ok: [ariadne-dev] => (item=localhost)
ok: [ariadne-dev] => (item=ariadne-dev)

TASK: [bennojoy.mysql | remove the test database] *****\
***
ok: [ariadne-dev]
```

```

TASK: [bennojoy.mysql | Create the database's] *****\
***
    skipping: [ariadne-dev]

TASK: [bennojoy.mysql | Create the database users] *****\
***
    skipping: [ariadne-dev]

TASK: [bennojoy.mysql | Create the replication users] *****\
***
    changed: [ariadne-dev] => (item={'name': 'repl', 'pass': 'foobar'})

TASK: [bennojoy.mysql | Check if slave is already configured for replication\
] ***
    skipping: [ariadne-dev]

TASK: [bennojoy.mysql | Ensure the hostname entry for master is available fo\
r the client.] ***
    skipping: [ariadne-dev]

TASK: [bennojoy.mysql | Get the current master servers replication status] *\
***
    skipping: [ariadne-dev -> {{ mysql_repl_master }}]

TASK: [bennojoy.mysql | Change the master in slave to start the replication]\
***
    skipping: [ariadne-dev]

[...]

PLAY RECAP *****\
***
    ariadne-dev                : ok=27   changed=10   unreachable=0   failed=0

```



Ansible roles are prioritized and they run before any other task. This happens even if the role configuration lines are written after the tasks.

If you now check the services installed on *ariadne-dev* you will see the mysql service listed there.

```
cd $pve/ariadne/orchestration/vagrant  
vagrant ssh
```

```
vagrant@ariadne-dev:~$ sudo service --status-all
```

4.4.2 Phansible

[Phansible](http://phansible.com/)¹¹ is another interesting tool. It comes in help providing a simple web interface that generates Ansible Playbooks for PHP based projects. A few clicks on the services you want to run on the vm and you get a zip file with all the Ansible configuration files.

Phansible is great but I suggest to always check that given configuration matches your needs: very likely there will be something that you have to add or modify.

¹¹<http://phansible.com/>

5. The Suggestotron application

Ariadne-dev is now ready to host the Suggestotron application. You just need to

- create the database
- populate the database
- provide the application code
- add a virtualhost for Apache

5.1 Create the database

In the Suggestotron guide, [chapter 5¹](#), there are detailed instructions about how to create by hand the Mysql database necessary to store the Suggestotron data.

With Ansible that can be automated, no need for manual configuration.

Checkout the *suggestotron_database_creation* branch from the *extra* repository

```
cd $pve
git checkout suggestotron_database_creation
```

You can see that the playbook is now telling Mysql to create the *suggestotron* database

```
roles:
# installs Mysql using Benno Joy's role
- {
    role: bennojoy.mysql,

    # mysql root password
    mysql_root_db_pass: root,

    # mysql port open only for IP 127.0.0.1
    mysql_bind_address: 127.0.0.1,

    # do not create any db
    mysql_db: [
```

¹http://phpbridge.org/intro-to-php/creating_a_database


```
        name: suggestotron
    ],

    # no additional mysql user
    mysql_users: [],
}
```

Run:

```
vagrant provision
```

and you will see in the output:

```
[...]
TASK: [bennojoy.mysql | Create the database's] *****
changed: [ariadne-dev] => (item={'name': 'suggestotron'})
[...]
```

Ansible claims that the database has been created. To check it:

```
ssh vagrant@ariadne-dev 'echo "show databases" | mysql -uroot -proot'
#provide "vagrant" as password
```

Output:

```
information_schema
mysql
performance_schema
suggestotron
```

5.2 Populate the database

Ansible can also populate the database.

One thing to keep in mind is that the database will be populated only if it's empty, i.e. right after its creation, otherwise every time `vagrant provision` is run the database will be populated with duplicated records.

Checkout the *suggestotron_database_creation* branch from the *extra* repository

```
cd $pve
git checkout suggestotron_populate_db
```

Now you see some more files

```
tree $pve/ariadne/orchestration/ansible
```

Output:

```
/
├─ ariadne-dev-playbook.yml
├─ sql
│   └─ suggestotron_init.sql
├─ tasks
│   ├─ apache2.yml
│   ├─ composer.yml
│   ├─ init.yml
│   ├─ php5.yml
│   ├─ populate_suggestotron_db.yml
│   ├─ psysh.yml
│   └─ xdebug.yml
└─ vars.yml
```

and the playbook has this additional line:

```
# populates the suggestotron db right after its creation
- include: tasks/populate_suggestotron_db.yml
```

This is the content of the populate_suggestotron_db.yml file:

```
1  ---
2  # This populates the suggestotron database
3
4  - name: populating suggestotron database
5    shell: echo "show tables" | mysql -uroot -proot suggestotron | grep "topics"
6    ignore_errors: yes
7    register: populated
8    failed_when: "'topics' not in populated.stdout"
9
10 ## debug: var=populated
11
```

```

12 # Copies the sql file containing the database data to /tmp
13 - copy: src=sql/suggestotron_init.sql dest=/tmp
14   when: populated|failed
15
16 # Imports the data in the database
17 - mysql_db: name=suggestotron state=import target=/tmp/suggestotron_init.sql
18   when: populated|failed

```

When the `populate_suggestotron_db` task runs, Ansible uses this command

```
echo "show tables" | mysql -uroot -proot suggestotron | grep "topics"
```

to check if the *suggestotron* database is populated.

The `mysql_db Ansible command`² is one of the many `Ansible modules`³ available and they are created to interact with services like Mysql, Apache etc.

As you can see the `mysql_db` module can be used to create, drop, dump and import a database.

Run again the provisioning:

```
vagrant provision
```

Output:

```
[...]
```

```

TASK: [populating suggestotron database] *****
failed: [ariadne-dev] => {"changed": true, "cmd": "echo \"show tables\" | mysql \
-uroot -proot suggestotron | grep \"topics\\\", \"delta\": \"0:00:00.007963\", \"end\":\
\"2014-12-01 17:51:01.546503\", \"failed\": true, \"failed_when_result\": true, \"rc\":\
1, \"start\": \"2014-12-01 17:51:01.538540\", \"stdout_lines\": []}
...ignoring

```

```

TASK: [copy src=sql/suggestotron_init.sql dest=/tmp] *****
ok: [ariadne-dev]

```

```

TASK: [mysql_db name=suggestotron state=import target=/tmp/suggestotron_init.sql\
] ***
changed: [ariadne-dev]

```

Ansible says that the database has been populated (see the “changed” status in last task).

Let’s see if it’s true with this command:

²http://docs.ansible.com/mysql_db_module.html

³http://docs.ansible.com/list_of_all_modules.html

```
ssh vagrant@ariadne-dev 'echo "select * from topics" | mysql -uroot -proot suggestotron'\n#provide "vagrant" as password
```

Output:

```
id  title  description\n1   Make Rainbow ElePHPants Create an elePHPant with rainbow fur\n2   Make Giant Kittens Like kittens, but larger\n3   Complete PHPBridge Because I am awesome
```

5.3 Get the Suggestotron code

The [Suggestotron repository](#)⁴ uses tags in the same way this manual uses branches therefore it's convenient to clone the git repository in the *projects* folder so that you can switch between tags to follow the step by step Suggestotron guide.

```
cd $pve/ariadne/projects\n git clone https://github.com/dshafik/suggestotron.git
```

What has been done till now is roughly the equivalent of the first 6 chapters of the Suggestron guide so you should checkout the Suggestotron code at chapter 7 level.



Maybe you need to have a look at the Creating A Data Class chapter first to understand how the Class is made

```
cd suggestotron\n git checkout Chapter_07_Adding_Topics
```

5.4 Add a virtual host for Apache

Checkout the *suggestotron_apache_vhost* branch from the *extra* repository

```
cd $pve\n git checkout suggestotron_apache_vhost
```

Have a look at the differences between the current branch and the previous one to have a clear idea of what happened:

⁴<https://github.com/dshafik/suggestotron>

```
git difftool -g -d suggestotron_populate_db..suggestotron_apache_vhost
```

▼ ariadne	4.1 kB	Tue 02 Dec 2014 19:44:32	▼ ariadne	4.1 kB	Tue 02 Dec 2014 19:44:32
▼ orchestration	4.1 kB	Tue 02 Dec 2014 19:44:32	▼ orchestration	4.1 kB	Tue 02 Dec 2014 19:44:32
▼ ansible	4.1 kB	Tue 02 Dec 2014 19:44:32	▼ ansible	4.1 kB	Tue 02 Dec 2014 19:44:32
▼ tasks			▼ tasks	4.1 kB	Tue 02 Dec 2014 19:44:32
suggestotron.yml			suggestotron.yml	947 B	Tue 02 Dec 2014 19:44:32
▼ templates			▼ templates	4.1 kB	Tue 02 Dec 2014 19:44:32
suggestotron.vhost			suggestotron.vhost	733 B	Tue 02 Dec 2014 19:44:32
ariadne-dev-playbook.yml	2.2 kB	Tue 02 Dec 2014 19:44:32	ariadne-dev-playbook.yml	2.4 kB	Tue 02 Dec 2014 19:44:32

Differences between branches

The playbook now includes this additional task file

```
# creates the Apache2 vhost for http://suggestotron
- {
    include: tasks/suggestotron.yml,
    application_name: "suggestotron",
    domain: "{{ application_name }}.ariadne-dev"
}
```

which describes these tasks:

```
1  ---
2  # This task takes care of the Apache vhost
3
4  - name: suggestotron.yml | Create symlink /var/www/"{{ domain }}" pointing to /v\
5  agrant/"{{ application_name }}"/
6    file: src=/vagrant/"{{ application_name }}"/ dest=/var/www/"{{ domain }}" owne\
7  r=www-data group=www-data state="link"
8
9  - name: suggestotron.yml | Add host to /etc/hosts
10    lineinfile: dest=/etc/hosts regexp='^127\.0\.0\.1' line='127.0.0.1 localhost {\
11  { domain }} www.{{ domain }}' owner=root group=root mode=0644
12
13  - name: suggestotron.yml | Copy Apache vhost for "{{ domain }}"
14    template: src="{{ application_name }}.vhost" dest=/etc/apache2/sites-available\
15  / owner=root group=root mode=0644
16
17  - name: suggestotron.yml | Enable Apache vhost "{{ domain }}"
18    file: src=/etc/apache2/sites-available/"{{ application_name }}.vhost" dest=/et\
19  c/apache2/sites-enabled/"{{ application_name }}.vhost" owner=root group=root sta\
```

```
20 te="link"
21   notify: restart apache2
```

As you can see, Ansible accepts variables.

They can be defined in the `vars.yml` file (included at the beginning of the playbook) or in the playbook file at the *include* level (as it happened already for the Mysql role, remember?) and they can be output using the double curly brackets notation “`{{ }}`”.

To apply the changes made in this branch run:

```
vagrant provision
```

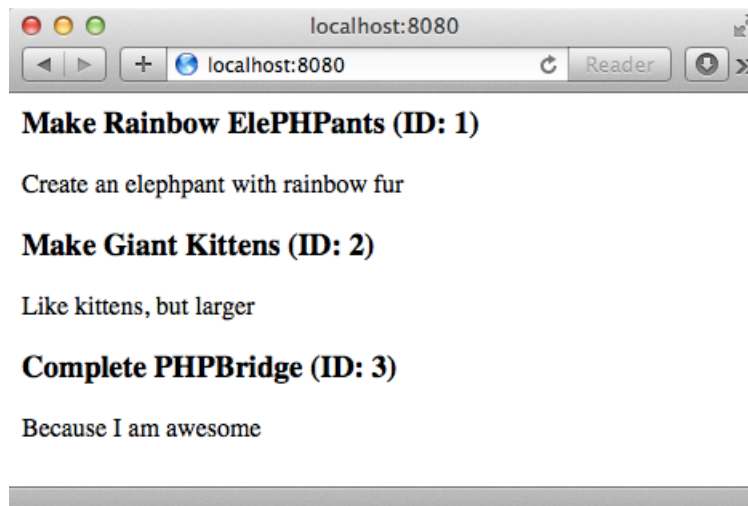
5.5 Run Suggestotron

Now you are ready to run the Suggestotron application for the first time.

Just add this line to the `/etc/hosts` file in your workstation

```
192.168.51.10    suggestotron.ariadne-dev www.suggestotron.ariadne-dev
```

Now open this link <http://suggestotron.ariadne-dev>⁵ and you should see this:



Index page

⁵<http://suggestotron.ariadne-dev>

5.6 Play with Suggestotron

If you are a beginner in PHP you can now dive in the [Intro to PHP guide](#)⁶ and play with the Suggestotron application just checking out the right tag every time you move to a new chapter.



The only difference with the Suggestotron guide will be the URLs: just replace `http://localhost:8080` with `http://suggestotron.ariadne-dev`.

For instance, you can roll back to chapter 2 and play a little with *psysh*, in this way:

Log in the virtual machine:

```
cd $pve/ariadne/orchestration/vagrant/  
vagrant ssh
```

and run:

```
psysh
```

now copy and paste, line by line, in the terminal these commands:

```
$my_variable = 5; # This assigns the value 5 to the name $my_variable.
```

```
$my_variable + 2;
```

```
$my_variable * 3;
```

```
echo $my_variable; # This shows the value of the variable
```

```
echo "Programming is easy!"; # This shows "Programming is easy"
```

```
echo "13 * 8";
```

```
echo 13 * 8;
```

```
echo "My variable is: $my_variable";
```



You can quit *psysh* by pressing *q* and *enter*

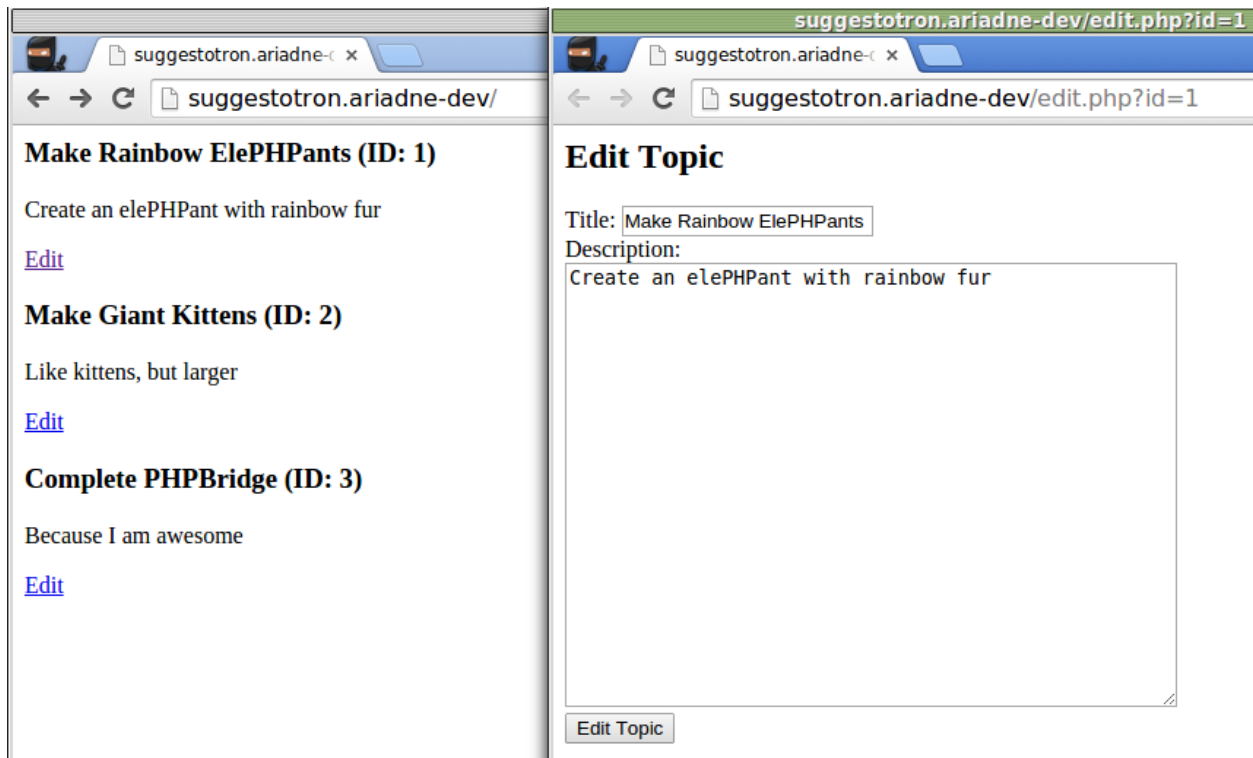
Then you can move to chapter 8. Switch the tag first

⁶<http://phpbridge.org/intro-to-php/>

```
cd $pve/ariadne/projects/suggestotron/  
git checkout Chapter_08_Editing_Topics
```

and then refresh your browser at the page `http://suggestotron.ariadne-dev/` and click the first *Edit* link.

You should see something like this:



Suggestotron chapter 8

6. The production environment

If you are a beginner you can probably postpone this chapter because I think you already have enough on your plate. Come back when you feel ready to publish your application.

Virtualization is cool because it creates as many isolated environments as you like and they can be programmatically configured. There is a specific environment which is usually out of the control of the developer: **production**.

The *production environment* is the space hosting the public version of your web application. Many applications run in shared hosting or in dedicated virtual machines created by the provider and then configured by hand. In these cases the developer can not be sure that the production environment is compatible with the development one until the code is published. Very likely the development and the production environment aren't equal and what's compatible today might become all of the sudden incompatible because of some differences in software versions. Troubles can be one apt-get upgrade far away.

Fortunately there are several cool [providers](#)¹ offering virtualized systems that can be handled exactly like the development one via Vagrant and you are going to see how to use [Digital Ocean](#)² as the production environment for the Suggestotron application.

I didn't pick Digital Ocean for any particular reason if not the fact that it's the only one I know and use.

Digital Ocean is inexpensive but definitely not for free so, if you want to try it, get ready to pay their [fees](#)³. I suggest to look around (or ask [them](#)⁴) for a *Promo Code*.

6.1 Vagrant and Digital Ocean

Vagrant has several [official plugins](#)⁵ to extend its own functionalities.

Digital Ocean is a well known and very good hosting provider which, among other services, provides virtual machines (a.k.a. droplets) in the cloud.

Basically you can purchase a virtual machine from Digital Ocean and use it for your production environment.

¹<https://github.com/mitchellh/vagrant/wiki/Available-Vagrant-Plugins#providers>

²<https://www.digitalocean.com/>

³<https://www.digitalocean.com/pricing/>

⁴<https://twitter.com/digitalocean>

⁵<https://github.com/mitchellh/vagrant/wiki/Available-Vagrant-Plugins>



I'm not associated in any way with Digital Ocean and, of course, there are other good competitors offering similar services.

How cool would it be if you could handle a Digital Ocean droplet in the say way you do with your local virtual machine? Well, you can!

The [vagrant-digitalocean](https://github.com/smdahlen/vagrant-digitalocean)⁶ plugin is a *provider plugin* that supports the management of [Digital Ocean](https://www.digitalocean.com)⁷ droplets, i.e. allows your Vagrant to talk to the Digital Ocean API.

Brief summary of what's going to happen:

1. installation of the Digital Ocean plugin
2. registration of the Digital Ocean droplet in Vagrant templates
3. you sign up at Digital Ocean
4. you generate a Digital Ocean token
5. you update your Vagrantfile with some additional information
6. Vagrant creates the Digital Ocean virtual machine for you

6.1.1 Install the digital-ocean plugin

This command installs the *digital-ocean* plugin globally in your system therefore it's a one time operation.



You can install any other plugin in the same way you are doing here.

```
cd $pve
vagrant plugin install vagrant-digitalocean
```

Output:

```
Installing the 'vagrant-digitalocean' plugin. This can take a few minutes...
Installed the plugin 'vagrant-digitalocean (0.7.4)'
```

The plugin can:

- create and destroy your droplets
- power on and off your droplets
- rebuild a droplet
- provision a droplet with the shell or Chef provisioners
- setup a SSH public key for authentication
- create a new user account during the droplet creation

⁶<https://github.com/smdahlen/vagrant-digitalocean>

⁷<https://www.digitalocean.com>

6.1.2 Digital Ocean droplets

Digital Ocean refers to a virtual machine as a “droplet” and offers many different linux distributions such as Ubuntu, Debian, CoreOS, Fedora and CentOS.

Each droplet runs a fresh operating system with no additional packages and configurations installed than the ones present in the system right after the installation from a distro iso.

Also, thanks to the vagrant plugin, a droplet acts as a Vagrant template box not very differently from the one previously created in this manual using Packer.

It would be nice to use a customized Packer box instead of a pre-built droplet but, as far as I know, this is not possible.

Exactly as it happened after the creation of the Packer box you need to add the Digital Ocean droplet to your Vagrant boxes. This is done with this command:

```
vagrant box add digital_ocean https://github.com/smdahlen/vagrant-digitalocean/raw/master/box/digital_ocean.box
```

This is a one time command and it doesn't do what you might expect: it doesn't clone a box in your system (it would make no sense) but it creates a simple json file in the Vagrant boxes directory:

```
tree ~/.vagrant.d/boxes/digital_ocean
```

Output:

```
/home/damko/.vagrant.d/boxes/digital_ocean
├── 0
│   └── digital_ocean
│       └── metadata.json
```

The content of the *metadata.json* file is just

```
{
  "provider": "digital_ocean"
}
```

This *provider* key will be used in the Vagrantfile to refer to this Vagrant box (see later).

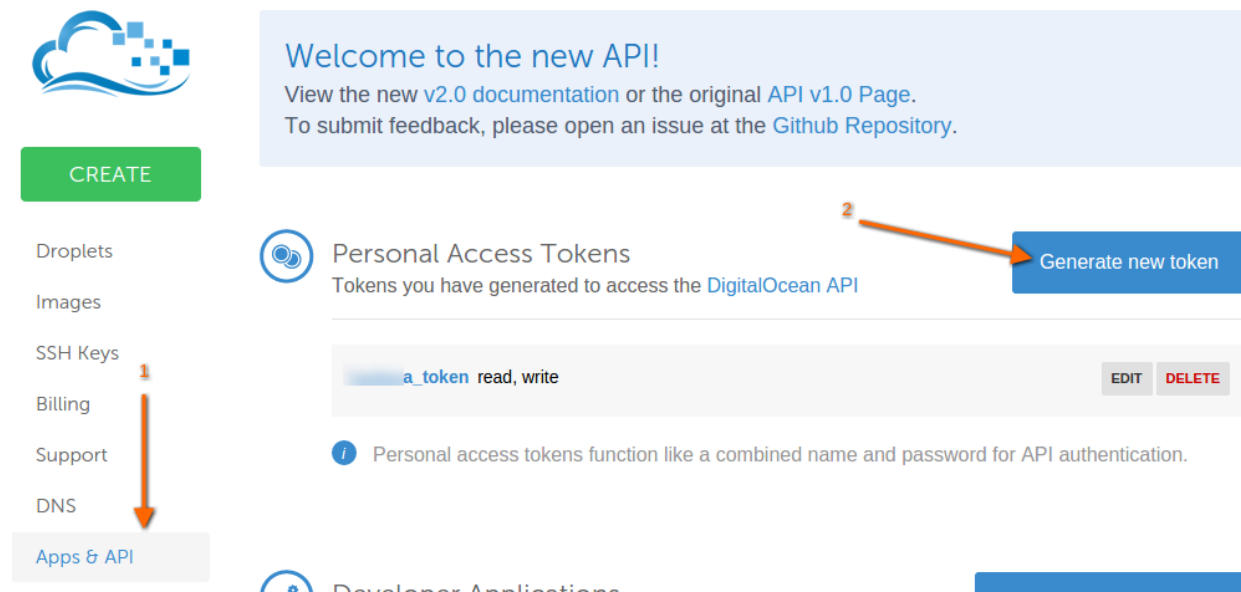
6.1.3 Generate the Digital Ocean token

You can create and administrate your droplets from the Digital Ocean administration panel but this will not be necessary for you because you are going to use Vagrant instead.

The only thing you are required to do through the administration panel is the creation of a *token*.


A token is a very long text string which acts as an authorization key for Vagrant (see later).

First of all you have to [sign in](#)⁸. Once you are logged in just follow the screenshots:



Administration panel

⁸<https://cloud.digitalocean.com/registrations/new>



New Personal Access Token

[Go back to Apps & API Page](#)

CREATE

- Droplets
- Images
- SSH Keys
- Billing
- Support
- DNS
- Apps & API

ariadne_prod


Select Scopes
Scopes limit access for personal tokens.

Default: ☒ Read Optional: ☒ Write

Generate Token

i Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for DigitalOcean over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Token creation



Please copy your new personal access token now. It won't be shown again for your security. ×

CREATE

- Droplets
- Images
- SSH Keys
- Billing
- Support
- DNS
- Apps & API

Welcome to the new API!
View the new [v2.0 documentation](#) or the original [API v1.0 Page](#).
To submit feedback, please open an issue at the [Github Repository](#).

Personal Access Tokens
Tokens you have generated to access the [DigitalOcean API](#)

Generate new token

a_token read, write	EDIT DELETE
ariadne_prod read, write (647675865bb75263e484b50d4cff17fca9ae3a9ae1df380f1d8cfc473fff5077)	EDIT DELETE

i Personal access tokens function like a combined name and password for API authentication.

Copy the token

Don't forget to copy and paste the token somewhere before closing the browser otherwise you have

to recreate it again. Also, keep the token private.

6.1.4 Adjust the Vagrantfile

Checkout the **production_environment* branch from the *extra* repository

```
cd $pve
git checkout production_environment
```

and have a look at the Vagrantfile.

There are new variables declared at the top of the file, one of which is DO_TOKEN.



you need to update the DO_TOKEN variable with the token you just created and copied

```
# Digital Ocean virtual machine for production
DO_VM_NAME = "ariadne-prod"
DO_TOKEN = "647675865bb75263e484b50d4c ff17 fca9ae3a9ae1df380f1d8c fc473ff5077"
```

The *DO_VM_NAME* variable sets *ariadne-prod* as the name for the Digital Ocean droplet.

In the bottom part of the Vagrantfile there are these additional lines:

```
1  #DIGITAL OCEAN
2  config.vm.define DO_VM_NAME do |production|
3
4    # this is the provider
5    production.vm.box = "digital_ocean"
6    production.vm.box_url = "https://github.com/smdahlen/vagrant-digitalocean/raw/\
7  master/box/digital_ocean.box"
8
9    # this disables the NFS support
10   production.vm.synced_folder ".", "/vagrant", disabled: true
11
12   # your public ssh key path
13   production.ssh.private_key_path = "/home/damko/.ssh/id_rsa"
14
15   production.vm.provider :digital_ocean do |provider|
16
17     # your token
```

```

18     provider.token = DO_TOKEN
19
20     # Droplet type.
21     # To get the list of the available droplets you can use this command
22     # curl -X GET -H 'Content-Type: application/json' -H 'Authorization: Bear\
23 er <your_token>' "https://api.digitalocean.com/v2/droplets"
24     provider.image = "Debian 7.0 x64"
25
26     # Droplet location
27     # To get the list of the available regions you can use this command
28     # curl -X GET -H 'Content-Type: application/json' -H 'Authorization: Bear\
29 er <your_token>' "https://api.digitalocean.com/v2/regions"
30     provider.region = "ams2"
31
32     # How much ram for the droplet. Careful here! The monthly cost of the dro\
33 plet depends on these parameters
34     # Check the costs here https://cloud.digitalocean.com/droplets/new before\
35 changing them
36     provider.size = "512MB"
37     provider.backups_enabled = false
38
39     # A string representing the name to use when creating a DigitalOcean SSH \
40 key for VPS authentication. It defaults to Vagrant.
41     provider.ssh_key_name = "damko@nitro"
42 end
43
44 production.vm.provision "ansible" do |ansible|
45
46     ansible.playbook = "../ansible/ariadne-dev-playbook.yml"
47
48 end
49 end

```

The first thing that you should notice in this Vagrantfile is this line:

```
production.vm.synced_folder ".", "/vagrant", disabled: true
```

NFS is obviously disabled for the production machine.

Read the comments I made and carefully update the next variables accordingly to your situation:

- `production.ssh.private_key_path`

- provider.region
- provider.size
- provider.backups_enabled

The *production.ssh.private_key_path* is absolutely mandatory and you have to set there the path to your SSH public key which usually lays in this file (in your workstation file-system):

```
~/.ssh/id_rsa.pub
```

If you don't have the `id_rsa.pub` file you can generate one by running:

```
ssh-keygen
```



just hit enter when asked for a password

This will generate both a private and a public key. They will be respectively stored in `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub`.



Note that the *ansible.playbook* is now called *ariadne-prod-playbook.yml*.

Usually each specific environment has its own playbook. This is because usually the development and the production environment differs a little: for instance you don't want to have the debugging options and tools in both the environments.

In this case, for instance, I removed Xdebug and Psysh from *ariadne-prod-playbook.yml*.

Also the Apache vhost configuration must be modified:

```
# creates the Apache2 vhost for http://suggestotron
- {
    include: tasks/suggestotron.yml,
    application_name: "suggestotron",
    domain: "{{ application_name }}.adriadne.prod"
}
```

And *adriadne.prod* should be changed with your public domain.

At the top of the playbook there is now this line:


```
gather_facts: yes
```

which forces Ansible to retrieve information about the running virtual machine. This information will be used in the Ansible tasks.

If you edit the *suggestotron.yml* file you will see that it has this additional line: `when: ansible_hostname == "ariadne-dev"`

```
- name: suggestotron.yml | Create symlink /var/www/"{{ domain }}" pointing to /v\
  agrant/"{{ application_name }}"/
  file: src=/vagrant/"{{ application_name }}"/ dest=/var/www/"{{ domain }}" owner\
  r=www-data group=www-data state="link"
  when: ansible_hostname == "ariadne-dev"
```

The *ansible_hostname* variable gets populated by Ansible because of the `gather_facts: yes` instruction and it can be used as a parameter.

In this case Ansible tries to create a symlink of the `/vagrant/suggestotron` directory to `/var/www/` only if the virtual machine is *ariadne-dev* because it uses, differently from *ariadne-prod*, the NFS shared directory.

6.1.5 Create the production vm

It's all set and you can create your droplet with this command:

```
vagrant up --provider=digital_ocean ariadne-prod
```

The `--provider=digital_ocean` parameter is required only when you are creating the droplet. After that you can forget about it.



the *vagrant up* command is now followed by the vm name. This is because the Vagrantfile now contains two virtual machines: *ariadne-dev* and *ariadne-prod*.

Look at this line in the Vagrantfile:

```
config.vm.define VB_DEV_VM_NAME, primary: true do |devel|
```

The *primary: true* part means that *ariadne-dev* is the primary vm and its alias inside the Vagrantfile is *devel*.

So this is the `vagrant up --provider=digital_ocean ariadne-prod` output:

Bringing machine 'ariadne-prod' up with 'digital_ocean' provider...

==> ariadne-prod: Using existing SSH key: damko@nitro

==> ariadne-prod: Creating a new droplet...

==> ariadne-prod: Assigned IP address: 178.62.173.1xx

==> ariadne-prod: Running provisioner: ansible...

PLAY [all] *****

GATHERING FACTS *****

ok: [ariadne-prod]

TASK: [playbook.yml | Run apt-get update if it was run last time more than 12 hours ago] ***

ok: [ariadne-prod]

TASK: [bennojoy.mysql | Add the OS specific variables] *****

ok: [ariadne-prod]

TASK: [bennojoy.mysql | Install the mysql packages in Redhat derivatives] *****
skipping: [ariadne-prod]

TASK: [bennojoy.mysql | Install the mysql packages in Debian derivatives] *****
changed: [ariadne-prod] => (item=python-selinux,mysql-server,python-mysqldb)

TASK: [bennojoy.mysql | Copy the my.cnf file] *****
changed: [ariadne-prod]

TASK: [bennojoy.mysql | Create the directory /etc/mysql/conf.d] *****
ok: [ariadne-prod]

TASK: [bennojoy.mysql | Start the mysql services Redhat] *****
ok: [ariadne-prod]

TASK: [bennojoy.mysql | update mysql root password for all root accounts] *****
changed: [ariadne-prod] => (item=ariadne-prod)
changed: [ariadne-prod] => (item=127.0.0.1)
changed: [ariadne-prod] => (item=:1)
changed: [ariadne-prod] => (item=localhost)

TASK: [bennojoy.mysql | update mysql root password for all root accounts] *****
skipping: [ariadne-prod] => (item=127.0.0.1)
skipping: [ariadne-prod] => (item=:1)

```
skipping: [ariadne-prod] => (item=localhost)
```

```
TASK: [bennojoy.mysql | copy .my.cnf file with root password credentials] *****  
changed: [ariadne-prod]
```

```
TASK: [bennojoy.mysql | ensure anonymous users are not in the database] *****  
ok: [ariadne-prod] => (item=localhost)  
ok: [ariadne-prod] => (item=ariadne-prod)
```

```
TASK: [bennojoy.mysql | remove the test database] *****  
ok: [ariadne-prod]
```

```
TASK: [bennojoy.mysql | Create the database's] *****  
changed: [ariadne-prod] => (item={'name': 'suggestotron'})
```

```
TASK: [bennojoy.mysql | Create the database users] *****  
skipping: [ariadne-prod]
```

```
TASK: [bennojoy.mysql | Create the replication users] *****  
changed: [ariadne-prod] => (item={'name': 'repl', 'pass': 'foobar'})
```

```
TASK: [bennojoy.mysql | Check if slave is already configured for replication] ***  
skipping: [ariadne-prod]
```

```
TASK: [bennojoy.mysql | Ensure the hostname entry for master is available for the  
client.] ***  
skipping: [ariadne-prod]
```

```
TASK: [bennojoy.mysql | Get the current master servers replication status] ****  
skipping: [ariadne-prod -> {{ mysql_repl_master }}]
```

```
TASK: [bennojoy.mysql | Change the master in slave to start the replication] ***  
skipping: [ariadne-prod]
```

```
TASK: [init.yml | Install Sys Packages] *****  
changed: [ariadne-prod] => (item=curl,vim,python-pycurl,python-apt,aptitude,mult  
itail,tree)
```

```
TASK: [init.yml | run Security Upgrade] *****  
changed: [ariadne-prod]
```

```
TASK: [init.yml | Ensure NTP is installed] *****
```

changed: [ariadne-prod]

TASK: [init.yml | Ensure NTP is running] *****

changed: [ariadne-prod]

TASK: [apache2.yml | Install Apache Packages] *****

changed: [ariadne-prod] => (item=apache2,libapache2-mod-php5)

TASK: [apache2.yml | Enable Apache rewrite module] *****

changed: [ariadne-prod]

TASK: [php5.yml | Check if php5-fpm is already installed] *****

failed: [ariadne-prod] => {"changed": true, "cmd": ["dpkg", "-s", "php5-fpm"], "\delta": "0:00:00.013664", "end": "2014-12-05 17:15:28.401623", "failed": true, "\failed_when_result": true, "rc": 1, "start": "2014-12-05 17:15:28.387959", "stdo\ut_lines": []}

stderr: dpkg-query: package 'php5-fpm' is not installed and no information is available

Use dpkg --info (= dpkg-deb --info) to examine archive files,
and dpkg --contents (= dpkg-deb --contents) to list their contents.

...ignoring

TASK: [php5.yml | Install PHP Packages] *****

changed: [ariadne-prod] => (item=php5-fpm,php5-curl,php5-cli,php5-mysql)

TASK: [php5.yml | enable Apache2 PHP module] *****

changed: [ariadne-prod]

TASK: [Ensure curl is installed (Debian).] *****

ok: [ariadne-prod]

TASK: [Install Composer into the current directory.] *****

changed: [ariadne-prod]

TASK: [Move Composer into globally-accessible location.] *****

changed: [ariadne-prod]

TASK: [Update Composer to latest version (if configured).] *****

ok: [ariadne-prod]

TASK: [populating suggestotron database] *****

failed: [ariadne-prod] => {"changed": true, "cmd": "echo \"show tables\" | mysql\

```
-uroot -proot suggestotron | grep "\"topics\\\"", "delta": "0:00:00.026047", "end"\\
: "2014-12-05 17:16:01.495320", "failed": true, "failed_when_result": true, "rc"\\
: 1, "start": "2014-12-05 17:16:01.469273", "stdout_lines": []}
...ignoring
```

```
TASK: [copy src=sql/suggestotron_init.sql dest=/tmp] *****
changed: [ariadne-prod]
```

```
TASK: [mysql_db name=suggestotron state=import target=/tmp/suggestotron_init.sql\\
] ***
changed: [ariadne-prod]
```

```
TASK: [suggestotron.yml | Create symlink /var/www/"suggestotron.adriadne.prod" p\\
ointing to /vagrant/"suggestotron"/] ***
skipping: [ariadne-prod]
```

```
TASK: [suggestotron.yml | Add host to /etc/hosts] *****
changed: [ariadne-prod]
```

```
TASK: [suggestotron.yml | Copy Apache vhost for "suggestotron.adriadne.prod"] ***
changed: [ariadne-prod]
```

```
TASK: [suggestotron.yml | Enable Apache vhost "suggestotron.adriadne.prod"] *****
changed: [ariadne-prod]
```

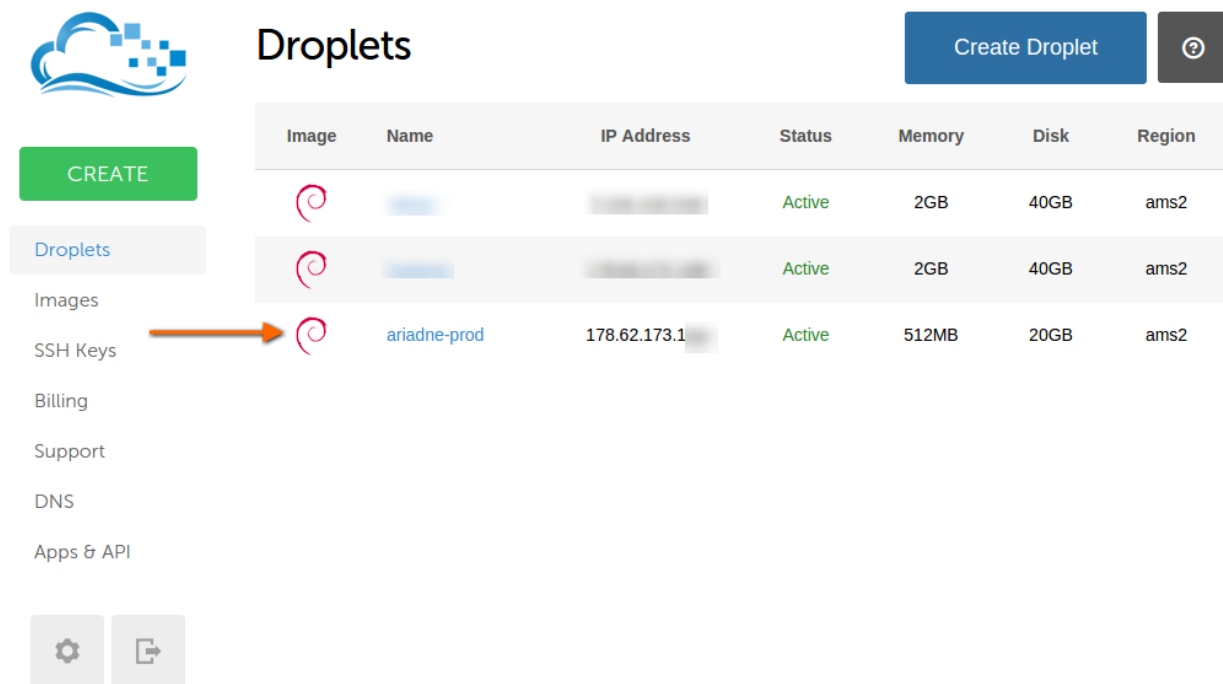
```
TASK: [suggestotron.yml | force Apache2 restart] *****
changed: [ariadne-prod]
```

```
NOTIFIED: [bennojoy.mysql | restart mysql] *****
changed: [ariadne-prod]
```

```
NOTIFIED: [restart apache2] *****
changed: [ariadne-prod]
```

```
PLAY RECAP *****
ariadne-prod          : ok=36   changed=26   unreachable=0   failed=0
```

Done! Your production environment is ready and in your Digital Ocean administration interface you now see the *ariadne-prod* droplet and its public ip address.



The screenshot shows the DigitalOcean 'Droplets' management page. On the left is a sidebar with navigation links: 'CREATE' (green button), 'Droplets' (selected), 'Images', 'SSH Keys', 'Billing', 'Support', 'DNS', and 'Apps & API'. At the top right are 'Create Droplet' and a help icon. The main area displays a table of droplets:

Image	Name	IP Address	Status	Memory	Disk	Region
			Active	2GB	40GB	ams2
			Active	2GB	40GB	ams2
	ariadne-prod	178.62.173.1	Active	512MB	20GB	ams2

An orange arrow points from the 'SSH Keys' link in the sidebar to the 'ariadne-prod' droplet row.

Droplet list

You have probably noticed that *ariadne-prod* is now missing the suggestotron code, right?

You can copy it in the Apache vhost root folder via ssh:

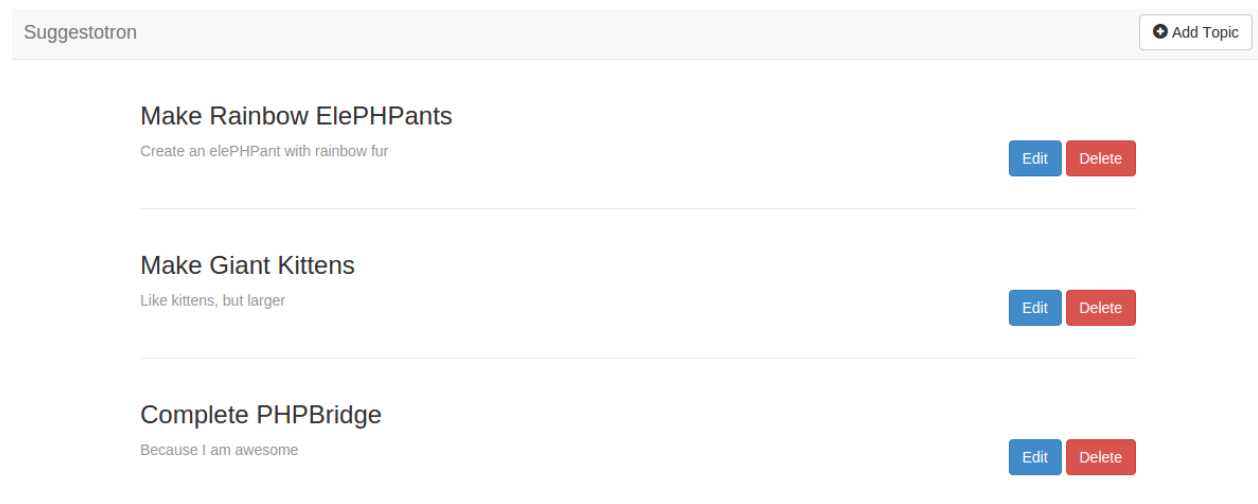
```
cd $pve/ariadne/projects
scp -r suggestotron root@<droplet_ip_address>:/var/www/
ssh root@<droplet_ip_address> 'chown www-data:www-data -fR /var/www/suggestotron\
.ariadne.prod'
```

Now you can configure your hosts file

```
sudo echo "<droplet_ip_address>    suggestotron.ariadne.prod" >> /etc/hosts
```

and you can see it at <http://suggestotron.ariadne.prod>⁹.

⁹<http://suggestotron.ariadne.prod>



Suggestotron application

6.2 Dive deeper into virtualization

The manual is almost over. I tried to share all I could and, by now, it should be pretty clear that virtualization is a huge and charming change in the computing world.

It's such a vast field that a new professional figure, the *devops engineer*, has born and the job offers are growing rapidly.

If you want to learn more about virtualization there are many books available for many different virtualization technologies.

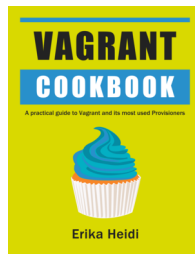
I highly suggest to start with [Vagrant CookBook](https://leanpub.com/vagrantcookbook)¹⁰ written by [Erika Heidi](https://twitter.com/erikaheidi)¹¹, honestly, the clearest book I have ever read.

She efficiently explains what are the differences between Ansible, Puppet and Chef and this helped me a lot in choosing Ansible as my favorite automation tool.

She also provides some valuable tricks and many recipes for all the three provisioning systems that will help you to speed up your learning process.

¹⁰<https://leanpub.com/vagrantcookbook>

¹¹<https://twitter.com/erikaheidi>



Vagrant Cookbook