



The Lab

-

Vagrant



VAGRANT

### Team

Dries De Houwer

Arno Heyvaert

Lenny Van de Winkel

# 1 Summary

<b>1 Summary</b>	<b>2</b>
<b>2 Project description</b>	<b>4</b>
2.1 Purpose	4
2.2 Corporate context	4
<b>3 What is Vagrant</b>	<b>5</b>
<b>4 Vagrant Installation</b>	<b>6</b>
4.1 Windows	6
4.2 Linux	6
4.3 MacOS	6
<b>5 How does Vagrant work?</b>	<b>7</b>
5.1 Plugins and Providers	7
5.2 Vagrant Boxes	7
5.3 Vagrantfile	8
5.4 Networking	10
5.5 Provisioning	10
<b>6 Vagrant and hypervisors</b>	<b>11</b>
6.1 Default provider and how to change it	11
6.2 Multiple providers and machines in one Vagrantfile	12
6.3 Multi-purpose Vagrantfile	13
<b>7 Vagrant and the cloud</b>	<b>15</b>
7.1 Google Cloud Platform	15
7.1.1 The plugin	15
7.1.1.1 Plugin installation	15
7.1.1.2 Setting up	15
7.1.1.3 Setting up a simple cloud instance	16
7.2 Microsoft Azure	19
7.2.1 The plugin	19
7.2.1.1 Installation	19
7.2.1.2 Setting up	19
7.2.2 Setting up a simple cloud instance	20
<b>8 Vagrant and provisioning tools</b>	<b>20</b>
<b>9 Vagrant for Developers</b>	<b>21</b>
<b>10 Vagrant for DevOps Engineers</b>	<b>23</b>
10.1 Use case	23
10.2 Creating a Vagrant box	23

<b>11 Vagrant vs Terraform</b>	<b>24</b>
<b>12 Conclusion</b>	<b>25</b>
<b>13 Sources</b>	<b>26</b>

## 2 Project description

The goal of this project is to explore the possibilities of Vagrant. We will explore the possibilities of this tool, how it can be used by developers and operators and what the possibilities of Vagrant are inside a corporate setting.

### 2.1 Purpose

The purpose of this project is to explore the full potential and capabilities of Vagrant in different contexts (Dev, DevOps, et cetera) and how it can be used with other (provisioning) tools and hypervisors.

Vagrant can deploy virtual environments locally and in the cloud. We will make a comparison between Vagrant and other IaC tools such as Terraform, AWS Cloudformation, ....

### 2.2 Corporate context

We will explore the possibilities of Vagrant in a corporate setting and will find out if it is beneficial if companies use it in their development, testing and production environments.

### 3 What is Vagrant

Vagrant is an open source tool created by Hashicorp. Vagrant can be used to create virtual machine environments and manage these environments. The tool can use multiple hypervisors and provisioning tools to create and manage virtual machines.

Vagrant uses "provisioners" and "providers". Provisioners are tools that allow the user to configure and customise virtual machines. Popular provisioners are Chef, Puppet, Ansible, Powershell & Bash scripts, .... Providers are services that are being used by Vagrant to create the virtual environments. Some examples of these providers are Virtualbox, Hyper-V, KVM, Docker, VMware, ....

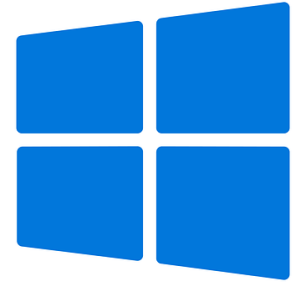


**HashiCorp**

## 4 Vagrant Installation

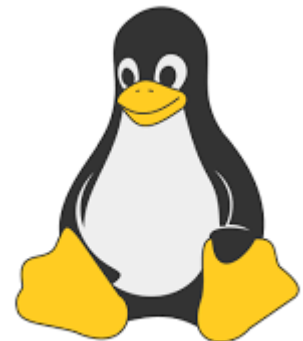
### 4.1 Windows

We can install Vagrant on Windows by browsing to the [official website](#), downloading and then running the Vagrant binary.



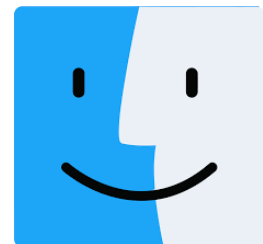
### 4.2 Linux

Vagrant can either be installed through using a binary or through a package manager (apt, yum, dnf, homebrew). The binary and the commands to install Vagrant are listed on the [official website](#).



### 4.3 MacOS

Vagrant can either be installed using the binary or Brew. The binary and the command to install Vagrant are listed on the [official website](#).



## 5 How does Vagrant work?

### 5.1 Plugins and Providers

Vagrant makes use of plugins to add additional functionality that the core installation does not provide. Plugins are an important feature of Vagrant and the chance that we will have to install and use Plugins while setting up an environment is very likely. Plugins can be installed with the `vagrant plugin install` command like shown below.

```
# Installing a plugin from a known gem source
vagrant plugin install my-plugin

# Installing a plugin from a local file source
vagrant plugin install /path/to/my-plugin.gem
```

When a plugin is successfully installed, it will automatically be loaded and ready for use.

A great example of the usage of plugins in Vagrant are providers. These providers are the platforms on which Vagrant can set up an environment. Fresh out of the box Vagrant supports three providers, these are Virtualbox, Hyper-V and Docker. When we plan to use Vagrant for production purposes, we probably want to use other providers than these three. Extra providers can then be installed with the help of plugins.

### 5.2 Vagrant Boxes

Vagrant uses boxes. A box is a base image of a virtual machine. Vagrant uses these boxes as a base image when creating a virtual environment. There is a wide range of boxes available on the [official website](#). We can compare these boxes to Docker images.

In the example below, we will define a virtual environment with an Ubuntu 18.04 VM in it. We can do this in 2 ways:

1. Using the CLI
2. Inside our Vagrantfile

CLI:

The command below creates a base Vagrantfile with comments and some example code. The name of this Ubuntu 18.04 box is "hashicorp/bionic64".

```
vagrant init hashicorp/bionic64
```

Vagrantfile:

We can easily change the box image when we already have a Vagrantfile. The only thing that needs to be changed is the "config.vm.box" property.

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
end
```

## 5.3 Vagrantfile

Vagrantfiles are used by Vagrant to determine which type of machine is needed and how to configure and provision these machines, the syntax is written in Ruby (but considering it's mostly simple variable assignment, knowledge of the Ruby programming language isn't necessary).

The actual filename for Vagrantfiles is "Vagrantfile" and only one Vagrantfile can be used per project. When using any vagrant command, Vagrant looks for the Vagrantfile in the directory that the command was used, climbing down the directory tree until it finds one. You can also set the "VAGRANT\_CWD" environment variable to specify a specific path.

An important concept with Vagrantfiles, is that Vagrant actually loads multiple Vagrantfiles and merges them into one. This allows us to set defaults or override prior settings. The order in which Vagrantfiles are loaded goes as follows:

1. Vagrantfile that is packaged with the [box](#).
2. Vagrantfile in the Vagrant home directory (defaults to ~/.vagrant.d).
3. Project directory.
4. Multi-machine overrides (see documentation).
5. Provider-specific overrides (see documentation).

More details about Vagrantfiles can be found in the [documentation](#). A simple example of a Vagrantfile can be found on the next page.



This Vagrantfile makes use of KVM to deploy an Ubuntu20.04 virtual machine. We use the inline shell to change the root password and set the timezone of the vm to Asia Ho Chi Minh. We also change the hostname to ubuntu20.04.

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

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.provider :libvirt do |libvirt|
    libvirt.driver = "kvm"
  end
  config.vm.box = "generic/ubuntu2004"
  config.vm.box_version = "3.1.16"
  config.vm.provision:shell, inline: <<-SHELL
    echo "root:rootroot" | sudo chpasswd
    sudo timedatectl set-timezone Asia/Ho_Chi_Minh
  SHELL

  config.vm.define "ubuntu20.04" do |ubuntu|
    ubuntu.vm.hostname = "ubuntu20.04"
  end

  config.vm.provision:shell, path: "bootstrap.sh"
end
```

## 5.4 Networking

The core installation of Vagrant comes with high-level networking options for port forwarding, connecting to a public network and the ability to create a private network. These network options are in a way that they work on multiple providers like VirtualBox and VMware.

If these high-level options are not specific enough for your use case there is no need to worry as all providers have their own networking options that you can configure which makes you able to fine tune your networking configuration based on your chosen provider(s).

Networks are configured within a [Vagrantfile](#). We can configure network settings by using the `config.vm.network` option. Every network has an identifier followed by configuration arguments. Below is an example of a basic network configuration where we port forward port 80 on the guest OS to port 8080 on our host.

```
Vagrant.configure("2") do |config|
  # ...
  config.vm.network "forwarded_port", guest: 80, host: 8080
end
```

It is possible to use multiple networks inside of our virtual environment. We can achieve this by defining multiple `config.vm.network` options in our Vagrantfile. Most providers use the same order in which the networks are defined inside a Vagrantfile.

## 5.5 Provisioning

Vagrant makes use of provisioners. Provisioners allow users to automatically install additional software onto a machine. This is a very useful feature of Vagrant since standard boxes are most likely not fit perfectly for certain use cases. By making use of provisioners like Ansible, Chef, Puppet ... we can automate this process so that we don't have to do this extra configuration and installation manually every time we spin up an environment.

## 6 Vagrant and hypervisors

Vagrant makes use of providers, which are services that are being used by Vagrant to create virtual environments. Vagrants uses hypervisors to accomplish this. There are 3 default hypervisors that are supported: Virtualbox, Hyper-V and Docker. We can also use other providers by installing plugins. More information about plugins can be found at [chapter 5.1](#) of this paper or in the [official documentation](#).

Different providers offer different purposes. While Virtualbox might do everything we desire when testing our application, it is recommended to use VMware in production environments because it's more stable and performant then Virtualbox.

### 6.1 Default provider and how to change it

Vagrant uses Virtualbox as default provider. This can be changed by specifying the "VAGRANT\_DEFAULT\_PROVIDER" environment variable.

The example below shows how we can change the default provider to VMware Desktop on Windows.

```
$env:VAGRANT_DEFAULT_PROVIDER = "vmware_desktop"
```

It is also possible to specify a provider when creating the virtual environment. This can be done by using the "-- provider=[provider]" parameter when running the "vagrant up" command. The example below sets the provider to Virtualbox.

```
vagrant up --provider=virtualbox
```

Another way to specify a different provider is inside a Vagrantfile. The example below shows the configuration that allows us to use Virtualbox as a provider.

```
ENV['VAGRANT_DEFAULT_PROVIDER'] = 'virtualbox'
```

It is also possible to override default provider settings. The example below changes the specified that is used to a box specifically created for VMware.

```
Vagrant.configure("2") do |config|
  config.vm.box = "bionic64"

  config.vm.provider "vmware_fusion" do |v, override|
    override.vm.box = "bionic64_fusion"
  end
end
```

## 6.2 Multiple providers and machines in one Vagrantfile

Vagrant can use multiple providers simultaneously. All of this can be achieved by specifying the different providers in a Vagrantfile. An example of such a configuration can be found below. This example will create one VM called "web". We will use Virtualbox to create this VM. A second VM will also be created. This VM is called "db" and this VM will use VMware desktop as provider.

```
# ...
Vagrant.configure("2") do |config|
  config.vm.define "web" do |web|
    # ...
    web.vm.provider : "virtualbox" do |v|
      # ...
      v.customize ["modifyvm", :id, "--memory", 512]
    end
  end

  config.vm.define "db" do |db|
    # ...
    db.vm.provider : "vmware_desktop" do |v|
      # ...
      v.customize ["modifyvm", :id, "--memory", 512]
    end
  end
end
```

## 6.3 Multi-purpose Vagrantfile

Certain providers have certain benefits. Some people prefer to use Virtualbox, others prefer VMware, KVM or other providers. Some hypervisors only support one or two operating systems (KVM, Hyper-V). While all the devs might use Virtualbox, you and your team could use VMware. This makes it hard when multiple people need to use the same Vagrantfile. This is where multi-purpose Vagrantfiles come into play.

An example of a multi-purpose Vagrantfile can be found below. We use the "generic/ubuntu2004" box and the specific box url for Virtualbox, Hyper-V and VMware Desktop. The provider can be set by using the "vagrant up --provider=..." command or by uncommenting one of the ENV lines in the code.

```
# Uncomment the next line to set the default provider to 'VMware
Desktop'
# ENV['VAGRANT_DEFAULT_PROVIDER'] = "vmware_desktop"

# Uncomment the next line to set the default provider to 'Hyper-V'
# ENV['VAGRANT_DEFAULT_PROVIDER'] = "hyperv"

Vagrant.configure("2") do |config|
  config.vm.box = "generic/ubuntu2004"

  config.vm.provider 'virtualbox' do |virtualbox|
    config.vm.box_url =
"https://app.vagrantup.com/generic/boxes/ubuntu2010/versions/3.6.8/provi
ders/virtualbox.box"
  end

  config.vm.provider 'vmware_desktop' do |vmware|
    config.vm.box_url =
"https://app.vagrantup.com/generic/boxes/ubuntu2010/versions/3.6.8/provi
ders/vmware_desktop.box"
  end if Vagrant.has_plugin?('vagrant-vmware-desktop')

  config.vm.provider 'hyperv' do |hyperv|
    config.vm.box_url =
"https://app.vagrantup.com/generic/boxes/ubuntu2010/versions/3.6.8/provi
ders/hyperv.box"
  end if Vagrant.has_plugin?('vagrant-vmware-desktop')
end
```

Note: there are some things that we need to keep in mind when using a multi-purpose Vagrantfile. One of the most important things is that not all boxes have optimised versions for certain providers. You can check the supported providers on the [box webpage on the official website](#). Another important thing to keep in mind is that this example doesn't check which plugin is installed and doesn't use the corresponding hypervisor. Vagrant will use the default provider and will show an error message when the default provider or plugin isn't installed.

## 7 Vagrant and the cloud

We can use Vagrant with a lot of local providers (hypervisors) but in certain use cases it might be useful to deploy in the cloud. Unfortunately, Vagrant doesn't natively support cloud providers, however there are several plugins available that make this possible. For this example, we're going to try two different cloud providers: Google Cloud Platform and Microsoft Azure.

### 7.1 Google Cloud Platform

#### 7.1.1 The plugin

Vagrant doesn't support Google Cloud Platform (GCP) natively (or any Cloud provider for that matter), so we'll need to use a plugin that will allow Vagrant to work with GCP. All the necessary documentation [can be found here](#).

##### 7.1.1.1 Plugin installation

The readme page of this plugin is very clear, and the installation of this plugin is easy. All we have to do to install this plugin is to execute the command below.

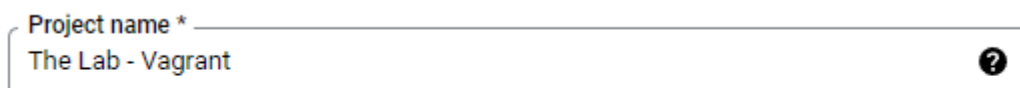
```
vagrant plugin install vagrant-google
```

##### 7.1.1.2 Setting up

Using the GCP plugin is straight-forward. We need three things to get things going:

- The Google Cloud Project ID
- A service file with access
- SSH key set up in the project metadata

The project ID is displayed upon creation of the project:



Project ID: **the-lab-vagrant**. It cannot be changed later. [EDIT](#)

We can also use already created projects. We can find the project ID by clicking on the list on the top left in the GCP console:



Getting a service file can be done by going to the GCP console: *IAM and admin > Service accounts > Create service account > Follow the steps > Click on your created service account > Keys > Add key > JSON*.

In order to set up a local SSH key so we can SSH into our instances and provision them, we'll need to add our public SSH key to the project's metadata. Info about the creation of SSH keys can be found [here](#). We will add our public key to the GCE metadata after we've created our key pair. We can do this by using the console and following the next steps: *Compute Engine > Metadata > SSH Keys > Edit*

There are a lot of configuration options. We won't go over all of the options but more info can be found in the [README file](#) of the plugins Github repository.

### 7.1.1.3 Setting up a simple cloud instance

Because of the way the plugin works, you cannot use Vagrant Boxes. However, Google Cloud provides you with a load of images you can use for different operating systems. Provisioning is still possible.

We then need to configure a Vagrantfile per the plugin:

```
Vagrant.configure("2") do |config|
  config.vm.box = "google/gce"

  config.vm.provider :google do |google, override|
    google.google_project_id = "GOOGLE_CLOUD_PROJECT_ID"
    google.google_json_key_location = "/path/to/private-key.json"

    google.image_family = 'ubuntu-2004-lts'

    override.ssh.username = "USERNAME"
    override.ssh.private_key_path = "~/.ssh/id_rsa"
  end
end
```



The data needed is self-explanatory. We're going to deploy a Ubuntu 20.04 LTS machine and try to run a few basic SSH commands on it in the **europe-west1-b** zone. This leaves us with the following Vagrantfile:

```
Vagrant.configure("2") do |config|
  config.vm.box = "google/gce"

  config.vm.provider :google do |google, override|
    google.google_project_id = "the-lab-vagrant"
    google.google_json_key_location =
"servicefile/the-lab-vagrant-a41e33c1ab2d.json"

    google.image_family = "ubuntu-2004-lts"

    google.zone = "europe-west1-b"

    override.ssh.username = "DESKTOP-ENGK677\vdwle"
    override.ssh.private_key_path = "C:/Users/vdwle/.ssh/google_compute_engine"
  end
end
```

We can then use the `vagrant up --provider=google` to deploy this instance.

After a minute or so, the Compute Engine is set up:

<input type="checkbox"/>	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP
<input type="checkbox"/>	✓	i-2022021915-cef07ed1	europe-west1-b			10.132.0.4 (nic0)	34.140.236.31

We can also provision this instance. As a minimal example, we're going to update the instance and install Apache2. We end up with this Vagrantfile:

```
Vagrant.configure("2") do |config|
  config.vm.box = "google/gce"

  config.vm.provider :google do |google, override|
    google.google_project_id = "the-lab-vagrant"
    google.google_json_key_location =
"servicefile/the-lab-vagrant-a41e33c1ab2d.json"

    google.image_family = "ubuntu-2004-lts"

    google.zone = "europe-west1-b"

    override.ssh.username = "vagrantlenny"
    override.ssh.private_key_path =
"C:/Users/vdwle/.ssh/google_compute_engine"
  end

  config.vm.provision:shell, inline: <<-SHELL
    apt update
    apt upgrade -y
    apt install apache2 -y
  SHELL
end
```

As long as our SSH is configured properly, this should run seamlessly and we should have an Ubuntu 20.04 LTS machine that has Apache2 running.

## 7.2 Microsoft Azure

### 7.2.1 The plugin

Like its Google Cloud counterpart, there's a simple Vagrant plugin for Microsoft Azure that works in the same way. The documentation of this plugin [can be found here](#).

#### 7.2.1.1 Installation

Installing this plugin is as straightforward as with GCE. All we have to do is run this command:

```
vagrant plugin install vagrant-azure
```

#### 7.2.1.2 Setting up

Using Microsoft Azure is a little more complex than GCP. You need the following:

- The [Azure CLI](#)
- Tenant ID
- Subscription ID
- Client ID
- Client Secret

The ID's and Secret can be found by logging into your Azure account > Azure Active Directory > App Registrations and creating a new app. The ID's will appear there. You can then create a secret by clicking on the link next to "Client credentials".

^ Essentials

Display name : [Vagrant](#)
Client credentials : [0 certificate, 1 secret](#)

Application (client) ID : e69f5554-356a-4403-b0bd-1dc90760a119
Redirect URIs : [Add a Redirect URI](#)

Object ID : 85c87cad-6a89-487e-a089-ae297c44a091
Application ID URI : [Add an Application ID URI](#)

Directory (tenant) ID : ed1fc57f-8a97-47e7-9de1-9302dfd786ae
Managed application in l... : [Vagrant](#)

Supported account types : [My organization only](#)

Certificates (0)
Client secrets (1)
Federated credentials (0)

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

+ New client secret

Description	Expires	Value ⓘ	Secret ID
vagrant	8/20/2022	<div></div>	<div></div>

## 7.2.2 Setting up a simple cloud instance

After we have found our ID's and the secret, we can fill in a Vagrantfile template that the plugin developers provide us:

```
Vagrant.configure('2') do |config|
  config.vm.box = 'azure'

  # use local ssh key to connect to remote vagrant box
  config.ssh.private_key_path = '~/.ssh/id_rsa'
  config.vm.provider :azure do |azure, override|

    # each of the below values will default to use the env vars named as
    # below if not specified explicitly
    azure.tenant_id = "ed1fc57f-8a97-47e7-9de1-9302dfd786ae"
    azure.client_id = "e69f5554-356a-4403-b0bd-1dc90760a119"
    azure.client_secret = ENV['AZURE_CLIENT_SECRET']
    azure.subscription_id = "8aa7351f-46dd-4891-949b-5d05d73b9b15"
  end
end
```

We don't want our client secret accidentally pushed to version control, so it's a safe bet to place that in an environment variable. For Windows, you can do "setx AZURE\_CLIENT\_SECRET secret" and for Linux it's "export AZURE\_CLIENT\_SECRET=secret".

## 8 Vagrant and provisioning tools

## 9 Vagrant for Developers

Vagrant supports both local providers and cloud providers which makes it an ideal tool for first testing your environments locally before pushing it to the cloud. In this part we are going to focus on how to use Vagrant to deploy an environment locally using VMware.

More specifically we are going to deploy two virtual machines, called "db" and "webapp" respectively. The "db" VM will host a MySQL 5.7 database while the "webapp" VM will host a .NET web application. We will make use of the 'generic/ubuntu1804' box for both our virtual machines. This box can be found on the Vagrant cloud.

Since the order in which we define our Virtual machines matters in Vagrant, we will first start configuring our db VM. The reason being that the .NET application on our web app will need to connect to the database. The code snippet below will deploy the db VM with an IP address on a Host-Only network.

```
config.vm.define "db" do |db|
  db.vm.boot_timeout= 600
  db.vm.box = "generic/ubuntu1804"
  db.vm.network "private_network", ip: "192.168.75.50"

  db.vm.provider "vmware_workstation" do |vmware|
    vmware.vmx["memsize"] = '2048'
    vmware.vmx["displayname"] = "db"
  end
end
```

In order to put a MySQL database on this virtual machine we will have to make use of provisioners. In this case we will make use of Docker. We do this by adding the code block shown below.

```
db.vm.provision "docker" do |d|
  d.run "mysql:5.7",
  args: "-e MYSQL_ROOT_PASSWORD=secretPass -e MYSQL_USER=testuser -e
  MYSQL_PASSWORD=testPass -e MYSQL_DATABASE=appdb -p 3306:3306"
end
```

Next up is our webapp VM. As you can see the hardware- and network configuration is very similar to the db VM.

```
config.vm.define "webapp" do |webapp|
  webapp.vm.boot_timeout= 600
  webapp.vm.box = "generic/ubuntu1804"
  webapp.vm.network "private_network", ip: "192.168.75.10"
  webapp.vm.synced_folder "./playbooks", "/vagrant/playbooks"

  webapp.vm.provider "vmware_workstation" do |vmware|
    vmware.vmx["memsize"] = '2048'
    vmware.vmx["displayname"] = "webapp"
  end
end
```

The only change is the `.vm.synced_folder` option which allows us to mount a directory from our host OS to our Virtual machine. We make use of this feature to get our Ansible playbooks which we will use to deploy our .NET app on our VM and because we are using the `ansible_local` provisioner.

Since our host OS is Windows we have to use the `ansible_local` provisioner. The reason being that Ansible can't be installed on a Windows OS. An alternative for this is to set up another VM using Linux on which you install Ansible. You can then use this VM to deploy your playbooks in your environment. However for the sake of this use case we are going to stick with the `ansible_local` provisioner option.

```
webapp.vm.provision "shell",
  inline: "sudo add-apt-repository -y ppa:deadsnakes/ppa && sudo apt
update -y && sudo apt install -y python3.9 && sudo useradd -s /bin/bash -d
/home/ansible/ -m -G sudo ansible"

webapp.vm.provision "ansible_local" do |ansible|
  ansible.playbook = "./playbooks/dotnetapp_playbook.yml"
end
```

As you can see in the code above we first use the shell provisioner to execute some bash commands that are necessary to run the Ansible playbook.

## 10 Vagrant for DevOps Engineers

Vagrant can be useful for DevOps Engineers too because it can work in many different ways, such as with different hypervisors, locally, in the cloud, or even a hybrid approach. In this section of this paper, we're going to explore what Vagrant can mean for DevOps Engineers with a common use case.

### 10.1 Use case

A DevOps Engineer wants to deploy an application in the cloud.

### 10.2 Creating a Vagrant box

## 11 Vagrant vs Terraform



## 12 Conclusion

## 13 Sources

[Documentation | Vagrant by HashiCorp \(vagrantup.com\)](#)