VAGRANT

What is Vagrant?

Vagrant isolates dependencies and their configuration within a single disposable, consistent environment, without sacrificing any of your existing tools.

Prerequisites

- Install the latest version of Vagrant.
- Install a virtualization product such as; VirtualBox, VMware Fusion, or Hyper-V.

Your first virtual machine

With the two commands below, you will have a fully usable virtual machine with Ubuntu 18.04 LTS 64-bit.

Initialize Vagrant

```
vagrant init hashicorp/bionic64
```

Before you can continue to the next step, ensure that Vagrant has created a **Vagrantfile**.

Start the virtual machine

Now that you have a Vagrantfile that configures your deployment, you can start the virtual machine.

```
vagrant up
```

When the virtual machine is successfully deployed, there will be a message stating that it is booted and ready.

Destroy the virtual machine

When you are done, be sure to terminate the virtual machine. Confirm when the CLI prompts you by typing **y**.

```
vagrant destroy
```

Install Vagrant

To install Vagrant, first, find the appropriate package for your system and download it. Vagrant is packaged as an **operating-specific package**. Run the installer for your system. The installer will automatically *add vagrant to your system path so that it is available in terminals.*

We highly recommend using the official installers on the downloads page. There may be differences between system package managers and the official installers that can lead to inconsistent experiences.

Verify the Installation

After installing Vagrant, verify the installation worked by opening a new command prompt or console, and checking that vagrant is available.

```
vagrant -v
```

You have successfully downloaded and installed Vagrant!

Initialize a Project Directory

The first step to configure any Vagrant project is to create a Vagrantfile. The Vagrantfile allows you to:

- Mark the root directory of your project. Many of the **configuration options** in Vagrant are **relative to this root directory**.
- Describe the kind of machine and resources you need to run your project, as well as what **software to install** and how you want to access it.

Create a directory

Make a new directory for the project you will work on throughout these tutorials.

```
mkdir vagrant_getting_started
```

Move into your new directory.

```
cd vagrant_getting_started
```

Initialize the project

Vagrant has a built-in command for initializing a project, vagrant init, which can take a box name and URL as arguments. Initialize the directory and specify the hashicorp/bionic64 box.

```
vagrant init hashicorp/bionic64
```

You now have a Vagrantfile in your current directory. Open the Vagrantfile, which contains some pre-populated comments and examples.

Next steps

You should commit your Vagrantfile to version control, this will allow every person on the project to benefit from Vagrant without any upfront configuration work.

Install and Specify a Box

Instead of building a virtual machine from scratch, which would be a slow and tedious process, *Vagrant uses a base image to quickly clone a virtual machine*. These base images are known as "**boxes**" in Vagrant, and specifying the box to use for your Vagrant environment is always the first step after creating a new Vagrantfile.

Install a Box (Optional)

Sometimes you may want to install a box without creating a new Vagrantfile. For this, you would use the box add subcommand.
You can add a box to Vagrant with **vagrant box add**. This stores the box under a specific name so that multiple Vagrant environments can re-use it. If you have not added a box yet, do so now.

```
vagrant box add hashicorp/bionic64
```

This will **download the box** named hashicorp/bionic64 from HashiCorp's Vagrant Cloud box catalog, where you can find and host boxes.
Boxes are globally stored for the current user. *Each project uses a box as an initial image to clone from, and never modifies the actual base image*.
In the above command, you will notice that *boxes are namespaced. Boxes are broken down into two parts—the username and the box name—separated by a slash*. In the example above, the username is hashicorp, and the box is bionic64. You can also specify boxes via URLs or local file paths, but that will not be covered in the getting started guide.

Use a Box

Now you've added a box to Vagrant either by initializing or adding it explicitly, you need to configure your project to use it as a base. Open the Vagrantfile and replace the contents with the following.

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

The hashicorp/bionic64 in this case must match the name you used to add the box above. This is how Vagrant knows what box to use. *If the box was not added before, Vagrant will automatically download and add the box when it is run.*
You may specify an explicit **version** of a box by specifying **config.vm.box_version**

```
 config.vm.box_version = "1.0.282"
```

You may also specify the **URL** to a box directly using **config.vm.box_url**:

```
config.vm.box_url = "https://vagrantcloud.com/hashicorp/bionic64"
```

Find More Boxes

Once you're ready to start using Vagrant in your development workflow you will need to know how to discover other boxes.

The best place to find more boxes is HashiCorp's Vagrant Cloud box catalog. HashiCorp's Vagrant Cloud has a public directory of freely available boxes that run various platforms and technologies. *You can search Vagrant Cloud to find the box you care about*.
You can also host your own boxes on Vagrant Cloud. Paid accounts can host private boxes to share privately within an organization.

Boot an Environment

Now that you have initialized your project and configured a box for it to use, it is time to boot your first Vagrant environment.

Bring up a virtual machine

Run the following from your terminal:

```
vagrant up
```

In less than a minute, this command will finish and you will have a virtual machine running Ubuntu.

SSH into the machine

You will not actually see anything though, since Vagrant runs the virtual machine without a UI. To prove that it is running, you can SSH into the machine:

```
vagrant ssh
```

This command will drop you into a **full-fledged SSH session**. Go ahead and interact with the machine and do whatever you want. Although it may be tempting, be careful about rm -rf /, since Vagrant shares a directory at /vagrant with the directory on the host containing your Vagrantfile, and this can delete all those files.
Take a moment to think about what just happened: With just one line of configuration and one command in your terminal, we brought up a fully functional, SSH-accessible virtual machine. Cool.
Terminate the SSH session with **CTRL+D**, or by logging out.

```
logout
```

Destroy the machine

Once you're back on your host machine, stop the machine that Vagrant is managing and *remove all the resources created during the machine-creation process*. When prompted, confirm with a yes.

```
vagrant destroy
```

Remove the box

The vagrant destroys command does not remove the downloaded box file. List your box files.

```
vagrant box list
```

Remove the box file with the remove subcommand, providing the name of your box.

```
vagrant box remove hashicorp/bionic64
```

Synchronize Local and Guest Files

Virtual machines are convenient for development in, but not many people want to edit files using a *plain terminal-based editor over SSH*. *Vagrant automatically syncs files to and from the guest machine*. This way you can **edit files locally and run them in your virtual development environment**.
By default, *Vagrant shares your project directory* (the one containing the Vagrantfile) with the **/vagrant** directory in your guest machine.

Explore the synced folder

On the virtual machine, list the files in the vagrant directory.
Tip: When you vagrant **ssh** into your machine, you're in **/home/vagrant**, which is a **different** directory from the **synced /vagrant** directory.

Next Steps

With synced folders, you can continue to use your own editor on your host machine and have the files synced into the guest machine.

Share an Environment

Vagrant also makes it easy to share and collaborate on these environments. The primary feature to do this in Vagrant is called Vagrant Share.
Vagrant Share is a plugin that lets you share your Vagrant environment with anyone around the world with an Internet connection. It will give you a URL that will route directly to your Vagrant environment from any device in the world that is connected to the Internet.

Prerequisites

Vagrant share requires **ngrok**. If you don't have ngrok, follow their install documentation to install it.
Install the plugin

Install vagrant share.

vagrant plugin install vagrant-share

Share the environment

```
vagrant share
```

Open a web browser and visit the URL from your output.
If you modify the files in your shared folder and refresh the URL, you will see your update. The URL is routed directly into your Vagrant environment and works from any device in the world that is connected to the internet.

Next Steps

To learn more about Vagrant Share, see the complete Vagrant Share documentation.

Rebuild an Environment

When you are ready to come back to your project, whether it is tomorrow, a week from now, or a year from now, you can spin it up again.

Vagrant Up

To start a box, run vagrant up.

```
vagrant up
```

That's it! Since the Vagrant environment is already all configured via the Vagrantfile, you or any of your coworkers simply have to run vagrant up at any time and Vagrant will recreate your work environment.

Teardown an Environment

it's time to stop, shut down, and finally destroy the environment. When using Vagrant you will choose from these options depending on how long you are stopping your work for, and whether or not you want to be able to come back to it.

Suspend the machine

Suspending the virtual machine will *stop it and save its current running state*.

```
vagrant suspend
```

When you begin working again bring the machine back up and its state **resumes from where you left off**. Start the machine again.

```
vagrant up
```

Suspending your machine is intended for **stopping and starting work quickly**. The downside is that *the virtual machine will still use disk space while suspended, and requires additional disk space to store the state of the virtual machine RAM.*

Halt the machine

Halting the virtual machine will **gracefully shut down the guest operating system and power down the guest machine**. Halt your machine now.

```
vagrant halt
```

Halting your machine will cleanly shut it down, *preserving the contents of the disk and allowing you to cleanly start it again.* Restart your machine.

```
vagrant up
```

A halted guest machine will take more time to start from a cold boot and will still consume disk space.

Destroy the machine

Destroying the virtual machine will **remove all traces of the guest machine from your system**. It'll stop the guest machine, power it down, and reclaim its disk space and RAM. Destroy the machine now, and confirm with a yes when prompted.

```
vagrant destroy
```

Again, when you are ready to work again, just issue a vagrant up.
It takes even longer to bring a machine up once you destroy it, and the machine's state will not be saved.

```
vagrant up
```

Networking and Provisioning Environments

This process was manual and would need to be repeated every time you shut down and brought this environment back up. **Vagrant automatically installs software** when you use vagrant up so that the guest machine can be repeatably created and ready to use.

Write a provisioning script

Set up Apache with a shell script. Create the following shell script and save it as **bootstrap.sh** in the same directory as your Vagrantfile.

```
#!/usr/bin/env bash
apt-get update
apt-get install -y apache2
if ! [ -L /var/www ]; then
  rm -rf /var/www
  ln -fs /vagrant /var/www
fi
```

This script downloads and starts Apache, and creates a symlink between your synced files directory and the location where Apache will look for content to serve.

Configure Vagrant

Next, edit the Vagrantfile to use the script when you provision the environment.

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

The provision line configures Vagrant to use the shell provisioner to set up the machine with the bootstrap.sh file. *The file path is relative to the location of the project root which should be the location of the Vagrantfile.*

Verify deployment

After Vagrant completes provisioning, the web server will be active. You cannot see the website from your own browser yet, but you can verify that the provisioning works by loading a file from within the machine.
SSH into the guest machine.
Now get the HTML file that was created during provisioning.

```
wget -qO- 127.0.0.1
```

This works because in the shell script above you installed Apache and set up the default DocumentRoot of Apache to point to your /vagrant directory, which is the default synced folder setup by Vagrant.

Next Steps

For complex provisioning scripts, it may be more efficient to package a custom Vagrant box with those packages pre-installed instead of building them each time. Learn more about the packaging custom boxes documentation.

Configure the Network

You will use Vagrant's networking features to provide access to the guest machine from your host machine.

Configure port forwarding

Port forwarding allows you to specify ports on the guest machine to share via a port on the host machine. This allows you to access a port on your own machine, but actually have all the network traffic forwarded to a specific port on the guest machine.

```
config.vm.network :forwarded_port, guest: 80, host: 4567
```

Reload so that these changes can take effect.

```
vagrant reload
```

Access the served files

Once the machine has loaded, you can access http://127.0.0.1:4567 in your browser. You will find a web page that is being served from the guest's virtual machine.

Next Steps

Vagrant also has other forms of networking, allowing you to assign a static IP address to the guest machine, or to bridge the guest machine onto an existing network. If you are interested in other options, read the networking page.

Explore Other Providers

Your project was backed with VirtualBox. But Vagrant can work with a wide variety of backend providers, such as VMware, Hyper-V, and more.

Install a new provider

Read the documentation of each provider for more information on how to set it them up.

Boot with the new provider

Once you have a provider installed, you do not need to make any modifications to your Vagrantfile; just run vagrant up with the proper provider and Vagrant will do the rest.

```
vagrant up --provider=vmware_desktop
```

For more information on providers, read the full documentation on providers.