

Cloud Computing Applications and Services

University of Minho

Guide 0



Warm-up

2024

Welcome to the practical classes of the Cloud Computing Applications and Services course! The goal of this first guide is to set up your lab environment, in particular, it will focus on how to deploy and configure virtual machines (VMs) in an automatic and repeatable fashion. The configurations and tools used in this guide will be helpful throughout the entire course.

Key for understanding the guides

Practical classes are intended to help you apply the content learned in the theoretical classes. The main goal is to understand how applications and services are configured, deployed, and served from Cloud Infrastructures and, to this end, we will be working from the perspectives of two major actors. The *cloud provider*, who is concerned with how to provide services to their clients, and the *application developer*, who is concerned with configuring and deploying their application correctly. Along these guides, each task or group of tasks will be marked with the following symbols to help you understand each role:

-  Tasks performed by the *cloud provider*;
-  Tasks performed by the *application developer*;

VMware

VMware is a virtualization software that helps you to run multiple operating systems in a single host. In this guide, you will use VMware to set up several virtual machines.

VMware documentation is available at: <https://docs.vmware.com>

Vagrant

Vagrant is a software for managing portable virtual software development environments (e.g., for VMware, VirtualBox, etc). You will use Vagrant to automatically deploy and configure a set of VMs.

Useful documentation on Vagrant is available at:

- Command line utilities - <https://www.vagrantup.com/docs/cli/>
- Configuration file - <https://www.vagrantup.com/docs/vagrantfile/>
- Multiple VMs setup - <https://www.vagrantup.com/docs/multi-machine/>
- Network configuration - <https://www.vagrantup.com/docs/networking/>
- Shell Provisioner - <https://www.vagrantup.com/docs/provisioning/shell.html>.

1 Setup

1.1 VMware installation

Start by installing the VMware tool on your machine, with the following steps:

1. Create a Broadcom account at <https://profile.broadcom.com/web/registration>. After filling the basic profile information, you can skip the "Build my profile" section. You can now log in to the website.
2. Download the appropriate version for your OS (choose the most recent version):
 - (a) *Linux & Windows*: VMware Workstation Pro 17.0 for **Personal Use**:
<https://support.broadcom.com/group/ecx/productdownloads?subfamily=VMware+Workstation+Pro>
 - (b) *Mac*: VMware Fusion 13 Pro for **Personal Use**:
<https://support.broadcom.com/group/ecx/productdownloads?subfamily=VMware+Fusion>

Note: After choosing the release version of the software, you must agree to the *Terms & Conditions*, which is located below the search bar, in order to unlock the download.

3. Verify the installation:
 - (a) *Linux & Windows*: Open VMware Workstation Pro > Help > About VMware Workstation. Check if License Information Type is "Personal Use Only"
 - (b) *Mac*: Open VMware Fusion Pro > VMware Fusion > About VMware Fusion. Check if "Licensed for Personal Use only" is under the version number

1.2 Vagrant installation

Now, install the Vagrant tool and the necessary plugins to work with VMware with the following steps:

1. Download and install Vagrant at:
https://developer.hashicorp.com/vagrant/install?product_intent=vagrant
2. After verifying the Vagrant installation with `vagrant --version`, install the plugins:
 - (a) *Vagrant VMware Utility* at <https://developer.hashicorp.com/vagrant/install/vmware>
 - (b) *Vagrant Plugin for VMware* at
<https://developer.hashicorp.com/vagrant/docs/providers/vmware/installation>
3. Verify the installation with `vagrant plugin list` and check if `vagrant-vmware-desktop` is listed.

2 Tasks

2.1 Vagrantfile

Vagrant uses a configuration file (*Vagrantfile*) to describe the type of machines required for a project and how to configure and provision them. Inspect the Vagrantfile provided with this guide to understand the following aspects.

1. The Vagrantfile describes how to configure and provision five VMs of two different types:
 - **Type 1**: VMs with 1 GiB of RAM and 2 virtual CPUs (*myVM*, *node1* and *node2*).
 - **Type 2**: VMs with 2 GiB of RAM and 2 virtual CPUs (*controlplane* and *monitorVM*).
2. In the beginning of the file, several variables are defined:

- (a) the local path for the user's SSH public and private keys, and the content of the public key. SSH keys are used to authenticate a user or process that wants to access a remote system through the SSH protocol. We need to share our public key with the VMs so we can access them from our computer.

```
PUBLIC_KEY_PATH = "~/ssh/id_rsa.pub"
PRIVATE_KEY_PATH = "~/ssh/id_rsa"
READ_PUBLIC_KEY = File.read(File.expand_path(PUBLIC_KEY_PATH)).strip
```

- (b) the IP address/range of each VM's private network.

```
MY_VM_IP = "192.168.56.10"
CONTROL_PLANE_IP = "192.168.56.100"
NODES_IP_RANGE = "192.168.56"
MONITOR_VM_IP = "192.168.56.200"
```

- (c) the number of node VMs to be created.

```
Number_Nodes = 2
```

Note 1: These variables may need to be adapted for your deployment.

Note 2: Further information regarding the networking and SSH topics can be found in the slides that accompany this guide.

Note 3: Vagrant allows calling ruby code. The `READ_PUBLIC_KEY` variable stores the content of the public key found at `PUBLIC_KEY_PATH`.

3. The Linux distribution (*i.e.*, VM base image), the base configuration (*i.e.*, 1 GiB RAM and 2 vCPUs) and the provision tasks that should execute after starting the VM are defined as global configurations:

```
config.vm.box = "bento/ubuntu-24.04"
config.vm.provider "vmware_desktop" do |vb|
  vb.force_vmware_license = "workstation"
  vb.memory = 1024
  vb.cpus = 2
end
config.vm.provision :shell, privileged: true, inline: $provision_all
```

4. The `provision_all` script used in the `config.vm.provision` parameter specifies a *shell* environment to execute the commands necessary for configuring the SSH Public Key authentication.

```
$provision_all = <<-SHELL
echo "[ALL|Task 1] Configure SSH Public Key authentication"
echo "#{READ_PUBLIC_KEY}" >> /home/vagrant/.ssh/authorized_keys
(...)
apt update
SHELL
```

Note 1: The first command copies the content of the public SSH key (stored at the `READ_PUBLIC_KEY` variable) to the `authorized_keys` file at the VM's file system.

Note 2: All the previous commands execute inside the VM!

5. Specific configurations for each VM are then specified independently.

- (a) The example below uses a loop to change the hostname and the IP address for each *node* VM:

```
(1..Number_Nodes).each do |i|
  config.vm.define "node#{i}" do |node|
    node.vm.hostname = "node#{i}"
    node.vm.network :private_network, ip: "#{NODES_IP_RANGE}.#{100+i}"
  end
end
```

- (b) In *myvm*'s case, it is also necessary to copy the private key to the “*~/ssh*” directory inside the VM and to run another provision script (*provision_myvm*). We need our private key on *myvm* so that we can later connect to the other VMs from inside *myvm*.

```
config.vm.define :myvm do |mv|
  mv.vm.hostname = "myvm"
  mv.vm.network :private_network, ip: MY_VM_IP
  mv.vm.provision :file, source: PRIVATE_KEY_PATH, destination: "~/ssh/"
  mv.vm.provision :shell, inline: $provision_myvm
end
```

- (c) In the case of the *controlplane* and *monitor* VMs, we need to increase the base memory configuration from 1 GiB to 2 GiB:

```
config.vm.define :controlplane do |cp|
  cp.vm.hostname = "controlPlane"
  cp.vm.provider "vmware_desktop" do |v|
    v.memory = 2048
  end
  cp.vm.network :private_network, ip: CONTROL_PLANE_IP
end
```


Note 1: A private network is only accessible for the host and other VMs running at the host. A NAT public network is used for communication between VMs and the internet.

Note 2: Inspect the configuration for *node* VMs. What IP addresses are being created for these VMs?

2.2 Testing

1. Deploy the five VMs by running `vagrant up` (in the same folder where the Vagrantfile is stored).
2. Check if the VMs were created with `vagrant status`.
3. Connect to each VM by using SSH (the default username is *vagrant*). You should use the standard `ssh` command in your shell. **Do not use the `vagrant ssh` command!**
4. Check (with the `ip add` command) that the VMs have a public (NAT network created automatically by Vagrant) and private network interface configured.
5. Check if the VMs can communicate through the private network (e.g., `ping` command).
6. Connect (with SSH) to *myvm* and, from it, try establishing an SSH connection to the other VMs.
7. Re-run the provision step, without shutting down the VMs, with:

```
vagrant provision --provision-with shell
```

Why is this command useful? 

8. Power off all the VMs and restart only *node1*:

```
vagrant halt  
vagrant up node1
```

9. Stop and destroy the VMs with `vagrant destroy`

2.3 Explore

Explore the following commands and check their usage and/or output [↩]:

- `man` - manual pages for commands, library functions, system calls, etc.
- `vim` / `nano` - text editors;
- `cat` / `more` / `less` - file content visualization;
- `mkdir` / `rmdir` - directory manipulation;
- `ls` / `pwd` / `cd` - file system navigation;
- `mv` / `cp` / `rm` / `scp` - file manipulation;

Learning outcomes

Experiment with Linux virtual machines' automatic deployment using VMware and Vagrant. Assess how Vagrant helps simplify the configuration and deployment of virtual machines. Revise Vagrant configuration parameters, scopes, and deployment/management commands.