TÉCNICO LISBOA

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	1.1
Application Layer		**Issue Date:**	23 Sep 2019
Web applications and HTTP		**Due Date:**	
Author: Prof. Rui Cruz		**Revision:**	4.1

1 Introduction

One of the *key tools* used for launching virtual development and test environments, as scaled "replicas" of real production environments, is **Vagrant**. Vagrant allows to create Infrastructures (Hosts, Networks) on a desktop/laptop machine. Vagrant acts as wrapper around virtualization software, speeding up many of the tasks associated with setting up, tearing down, and sharing Virtual Machines.

Vagrant is an invaluable tool for managing local *sandboxes* because it creates disposable multi-node environments, as self contained systems, used for development, testing and gaining experience with versions of Operating Systems, Applications, Tools or configurations, Networked Systems, and then throw them away. These sandbox systems can also be used to reproduce Network connectivity and protocols, performance problems, or test client-server interactions.

Vagrant uses a *definition file* (named **Vagrantfile**) to define computers (one or more Virtual Machines or Containers), networking, and storage for an environment that can be created and run with a local virtualization tool. Because the definition is contained in an externalized file, a Vagrant environment is reproducible. The entire environment can be torn down (even destroyed), and then rebuilt with a single command (using that **Vagrantfile**).

The objective of the experiments with Vagrant in this Lab is learning how to configure a multi-node environment (servers, network, applications) and to experiment with Application Layer services, such as a **Web Server** and the **Hypertext Transport Protocol (HTTP)**.

Preliminary Notes

The instructions in this document are applicable to Computers at the IST Labs.

Nevertheless, one nice feature of the software stack we are going to use is that it is portable to many platforms including **YOUR OWN** personal computers, running the following Operating Systems:

- Microsoft Windows from version 10 up
- Apple macOS from versions 10.13 'High Sierra' up
- Debian-based Linux, such as Ubuntu (recommended) from versions 12.04 'Precise' up.

It is not recommended to apply this setup to a virtual machine (nested virtualization), although possible, as the configuration requires access to a hypervisor environment (recommended Virtualbox) in the host system.

Note: Avoid copying text strings from the command line examples or configurations in this document, as pasting them into your system or files may introduce/modify some characters, leading to errors or inadequate results.

2 Multi-Node spin off with Vagrant

For each Vagrant environment, or set of machines to play around with, it is always good to create a **specific project directory** in your system (because each project has a Vagrantfile, but with different code for each project).

To start this new project, create a directory, and name it, for example, **weblab**. Download from the course website the file `RC_19_20_LAB_01_P1_support_files.zip` and uncompress the content to the **weblab** project folder. A `Vagrantfile`, a file named `bootstrap_web.sh` and a folder named `html` will be created (the folder html contains a file named `index.html`).

In order to ensure adequate updating and upgrading for Vagrant environments using the Virtualbox provider, and in case you have not already done it, add the `vagrant-vbguest` Plugin with the command:

```
:~$ vagrant plugin install vagrant-vbguest
# or, if already installed
:~$ vagrant plugin update vagrant-vbguest
```

2.1 First Step

We will use a new box with a Ubuntu OS, named "**ubuntu/trusty64**", for all the virtual machines in this experiment.

Open the `Vagrantfile` with an editor and verify that its content is similar to the one illustrated below. Note the comment lines 1 and 2 (Ruby directives) before the first line of the main code block `Vagrant.configure(2)do |config|`).

The main objective is to create a code block for each machine , containing the specific provisioning commands. The first machine will be named "webserver", as seen in the code block that starts with `config.vm.define "webserver"do |web_config|`.

The IP address of the machine is an example that can typically be used, but you may need to adapt the value to the private network address of Virtualbox in your system.

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

Vagrant.configure(2) do |config|

  # Global configurations
  config.ssh.insert_key = false
  config.ssh.forward_x11 = true
  config.vbguest.auto_update = true
  config.vm.box_check_update = false
```

```
  # Definitions for a VM
  config.vm.define "webserver" do |web_config|
    web_config.vm.box = "ubuntu/trusty64"
    web_config.vm.hostname = "webserver"
    web_config.vm.network "private_network", ip: "192.168.56.21"
    web_config.vm.network "forwarded_port", guest: 80, host: 8080
    web_config.vm.provider "virtualbox" do |vb|
      vb.name = "webserver"
      opts = ["modifyvm", :id, "--natdnshostresolver1", "on"]
      vb.customize opts
      vb.memory = "256"
    end # of vb
   end # of web_config
end # of config
```

Now boot the “webserver” server with the command:

```
:~$ vagrant up
```

You will observe that Vagrant will start by downloading the Ubuntu box and when the whole process is finished a new Virtual Machine will be running. When ready, establish a session with the “webserver” system using the following command:

```
:~$ vagrant ssh
```

The session is established and we will get the machine prompt, similar to:

```
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-13-generic x86_64)

Last login: Fri Oct 1 01:31:44 2017 from 10.0.2.2
vagrant@webserver:~$
```

Now, you need to discover what is the IP address of your host system machine by opening a new terminal window and running `ifconfig | grep~inet` for mac/linux or `ipconfig /all` for windows (or look at the network properties icon).

From the Terminal window of the “webserver” verify that you have connectivity to your host system as follows (use your own host IP address, not the one in the example):

```
vagrant@webserver:~$  ping 192.168.1.216
PING 192.168.1.216 (192.168.1.216) 56(84) bytes of data.
64 bytes from 192.168.1.216: icmp_seq=1 ttl=63 time=0.443 ms
64 bytes from 192.168.1.216: icmp_seq=2 ttl=63 time=0.611 ms
--- 192.168.1.216 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.443/0.567/0.611/0.077 ms
```

```
vagrant@webserver:~$ exit
logout
Connection to 127.0.0.1 closed.
```

And now, from your host system to the “webserver”:

```
:~$ ping 192.168.56.21
PING 192.168.56.21 (192.168.56.21): 56 data bytes
64 bytes from 192.168.56.21: icmp_seq=0 ttl=64 time=0.625 ms
64 bytes from 192.168.56.21: icmp_seq=1 ttl=64 time=0.610 ms
--- 192.168.56.21 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.610/0.654/0.728/0.052 ms
:~$
```

You will now shutdown the “webserver” and verify that it is not running, with the following commands:

```
:~$ vagrant halt
==> webserver: Attempting graceful shutdown of VM...
:~$
:~$  vagrant status webserver
Current machine states:

webserver                         poweroff (virtualbox)

The VM is powered off. To restart the VM, simply run `vagrant up`
:~$
```

2.2 Second Step

Open again the Vagrantfile to create a replica of the block that defines the “webserver” server, and change in that code block all instances of “webserver” to “client”.

Also change the IP address to a different value but in the same subnet, for example 192.168.56.11. **For this server remove the line that forwards the port 80**.

Now boot both the “webserver” and “client” systems with the command:

```
:~$ vagrant up
```

You will observe that the “webserver” system will boot up rapidly but the “client” system will take a little more time as it needs to be instantiated as a new Virtual Machine.

When ready establish sessions with the “webserver” and the “client” systems using the following commands (you can open a terminal window for each) and verifying the addresses of interfaces, as well as if the servers can reach each other:

```
:~$ vagrant ssh  webserver
vagrant@webserver:~$ ping 192.168.56.11
...
vagrant@webserver:~$ ip addr show
```

```
:~$ vagrant ssh  client
vagrant@client:~$ ping 192.168.56.21
...
vagrant@client:~$ ip addr show
```

Did you notice that you have a loopback interface plus two network interfaces (**eth0** and **eth1**) with different IP addressing schemes?

You can now shutdown both the "webserver" and the "client" systems and verify that they are not running, with the following commands:

```
:~$ vagrant halt
==> client: Attempting graceful shutdown of VM...
==> webserver: Attempting graceful shutdown of VM...
:~$
:~$  vagrant status
Current machine states:
webserver                 poweroff (virtualbox)
client                    poweroff (virtualbox)

This environment represents multiple VMs. The VMs are all listed
   above with their current state. For more information about a
   specific
VM, run `vagrant status NAME`.
:~$
```

2.3 Third Step

Open the Vagrantfile to insert a "shell provision" instruction in the code block of the "webserver" system, similar to the line illustrated in this example:

```
  config.vm.define "webserver" do |web_config|
    web_config.vm.box = "ubuntu/trusty64"
    web_config.vm.hostname = "webserver"
    web_config.vm.network "private_network", ip: "192.168.56.21"
```

```
    web_config.vm.network "forwarded_port", guest: 80, host: 8080
    web_config.vm.provider "virtualbox" do |vb|
      vb.name = "webserver"
      opts = ["modifyvm", :id, "--natdnshostresolver1", "on"]
      vb.customize opts
      vb.memory = "256"
    end # of vb
    web_config.vm.provision "shell", path: "bootstrap_web.sh"
  end # of web_config
```

This instruction will upload and execute the `bootstrap_web.sh` script within the guest machine. The script will install the "apache" web server application (`http://httpd.apache.org`) in the machine, and start it.

In order to execute the provision and then verify the results, do the following:

```
:~$ vagrant up --provision webserver
```

You will start to see the machine booting, and then being provisioned (starting with packages update) with the "apache" web server.

When finished, open the following URL: `http://localhost:8080` in a browser of your host machine. If everything went well, you will be greeted with a **funny message**.

Shutdown the "webserver" system with :

```
:~$ vagrant halt  webserver
```

3 HTTP Experiments

Now that you have the configuration for the two machines, and that you have already instantiated both, it is time for some experiments with the Web Server and HTTP.

3.1 Starting and connecting to the machines

Open a Terminal window and go to the **multinode** project folder. From there execute the command:

```
:~$ vagrant up
```

Confirm that you have both the "client" and the "webserver" running using the command `vagrant global-status`.

In order to access to the "client" machine, open a new Terminal window and using the command `vagrant ssh` for that system you will get something similar to the following:

```
:~$ vagrant ssh client
Welcome to Ubuntu 14.04.5 (GNU/Linux 3.13.0-132-generic x86_64)

Last login: Fri Oct 1 01:31:44 2017 from 10.0.2.2
vagrant@client:~$
```

You will now use the `telnet` application to connect to processes running in those systems.

Try the following interaction with the "webserver":

```
vagrant@client:~$ telnet webserver 80
```

You get an error, right? From the error you will observe that something is missing. That missing piece is DNS, i.e., the Domain Name Service to "translate" the system name to its IP address. So now, what can we do? See next section...

3.2 Using an alternative to DNS

As we did not implement a DNS server, we will use the special system file "/**etc**/**hosts**", that is used in nix* systems (Linux/Unix) as a static translator from hostnames to IP addresses. The command to populate the "/etc/hosts" file is as follows, and you can confirm with the command "cat" that the file now has a translation for the IP address of the "webserver" name:

```
vagrant@client:~$ sudo sh -c "echo '192.168.56.21 webserver' >> /
   etc/hosts"
vagrant@client:~$ cat /etc/hosts
```

You should not have problems now for the interaction and you should obtain a funny HTML page with that command.

3.3 Connecting to the webserver with a browser

With a browser in your Host machine you can visualize that simple HTML file using the IP address of the "webserver" machine and the port where the web service is listening: `http://192.168.56.21:80`

4 Finishing your Experiments

In order to stop the Virtual Machines and to verify the global state of all active Vagrant environments on the system, we can issue the following commands:

```
:~$ vagrant halt
==> default: Attempting graceful shutdown of VM...
:~$ vagrant global-status
```

Confirm that the statuses of the VMs is 'powered off'. In order to prevent those instantiated machines to use resources, you can destroy them, as they can now be recreated with that simple command `vagrant up`. Confirm that there are no VMs listed.

```
:~$ vagrant destroy
default: Are you sure you want to destroy the 'default' VM? [y/N]
==> default: Destroying VM and associated drives...
:~$ vagrant global-status
```

5 Submitting the Results of the Experiments

The experiments that you execute in this LAB will either produce results that you need to report or from which you will be asked questions about the execution. In order to report the results you achieved, proceed as follows:

5.1 General Procedure

The procedure for submission is quite simple:

1. In your Google Classroom for this course, you will find an Assignment for reporting this specific Lab experiment (with a Due Date);
2. The Assignment provides a Link to the TSUGI CLOUD Web Form with Exercise Questions;
3. When you are prepared with the requested materials (screen-shots, command line outputs, developed code, etc.) you will submit the items into the respective Exercise Form questions of the Assignment;
4. In the same Exercise Form of the Assignment you may be asked to comment your submissions;
5. Please note that in some types of Exercises you may also be asked to provide feedback (anonymously) to the answers from some of your classmates, classifying with "points" the level of accomplishment of your classmate answers.
6. When finished answering the Exercise Form, click the button **Done** in the top left of the Form; You will be returned to the Google Classroom Assignment;

7. In the Assignment, do not forget to confirm (click the button) **MARK AS DONE** or **TURN IN** when the assignment is completed;

5.2 Specific Procedure

For this LAB Assignment you will save the output of the command `vagrant up` into a file named "**vagrant.out**", after your changes are done to the Vagrantfile as explained in Section 2.3, and, of course, after ensuring that the whole experiment executed well.

One way of capturing the output of a command can be done as shown in the listing below:

```
# make sure the VMs are halted
:~$ vagrant halt
# start the VMs and save the output into a file
:~$ vagrant up > vagrant.out
```

You should see no output as it is being redirected to a file. However, when the command finishes, the cursor will return to a new prompt. The *vagrant.out* file created can be opened by any text editor.

Summarizing:

1. Produce the "**vagrant.out**" as explained above, and submit in the TSUGI Form where asked;
2. Submit the Vagrantfile produced as explained in Section 2.3 in the TSUGI Form where asked;
3. Capture a screenshot of the telnet interaction with the "webserver" after correcting the DNS, to submit in the TSUGI Form where asked;
4. Capture a screenshot of your browser opening the URL for the webserver, to submit in the TSUGI Form where asked;

WARNING. Submissions MUST BE MADE in the TSUGI CLOUD Web Form through Classroom. No other type of submission will be considered.