TÉCNICO LISBOA

RC 20/21	LAB ASSIGNMENT	Guide	01
Application Layer		Issue Date:	28 Sep 2020
Web applications and HTTP		Report Due:	02 Oct 2020
Author: Prof. Rui Cruz		Due Date:	06 Oct 2020

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 if not needed anymore. 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**), which is a Ruby (`www.ruby-lang.org`) script to define networked computers (one or more Virtual Machines or Containers) and storage for an environment that can be created and run with a local virtualization Hypervisor. 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 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

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 the Hypervisor environment (mandatory Virtualbox) in the host system.

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

RC 20/21	LAB ASSIGNMENT	Guide	01
Application Layer		**Issue Date:**	28 Sep 2020
Web applications and HTTP		**Report Due:**	02 Oct 2020
Author: Prof. Rui Cruz		**Due Date:**	06 Oct 2020

2 Submitting the Results of the Experiments

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

2.1 General Procedure

On your system, it is advisable to **create a working folder for this lab** (not the experiment folder!), that will contain the necessary files for reporting. These files may be screenshots, code snippets, text documents, system logs, etc.

It is advisable to give specific and good identifying names to those files, following what will be asked to report. For example, you may be asked to take several screenshots during the procedures, and so, rename those screenshots to the specific items being requested in the assignment. It is also advisable to collect results in a text file or document file while doing the experiment, so that you will have those results when you will need to submit them in the online assignment form.

The procedure for submission is quite simple:

1. This Lab assignment MUST be reported up to the **Report Due** date;
2. In your Google Classroom for this course, you will find an **Assignment** for reporting the results from this specific Lab experiment.
3. The Assignment provides a Link to the **TSUGI CLOUD Web Form** containing the Questions to be answered;
4. The Assignment **Due Date** corresponds to the date it MUST be considered completed. For that you will (click the button) **Turn in** or **Mark as Done** in order for the Assignment to be assessed by the Teachers;
5. 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;
6. In the same Exercise Form of the Assignment you may be asked to comment your submissions;
7. 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;
8. Please note that this process involves a **Peer Review** mechanism, in which you will be asked to provide **feedback** (anonymously) to the Reports of your classmates. So, after submitting your own assignment, get back to it to see the Reviews that the system have distributed automatically to you;

2.2 Specific Procedure

For this Lab Experiment you will need to submit the following items (explained in Section 3 and Section 4.:

1. Interpret the original "**Vagrantfile**" you have downloaded, explaining, in your own words, what you think the "instructions" in it are supposed to do. Submit your answer in the TSUGI Form where asked;
2. Post the result of the command `ifconfig` for mac/linux or `ipconfig` for windows, that allow you to know the network interfaces in your host. Submit your answer in the TSUGI Form where asked;
3. Interpret briefly the results of the command `ifconfig` for mac/linux or `ipconfig` for windows, indicating the IPv4 network addresses of the "active" interfaces in your host. Submit your answer in the TSUGI Form where asked;
4. Submit the modified "**Vagrantfile**" produced as explained in Section 3.2 in the TSUGI Form where asked;
5. Post the result of the command `ip addr show`, as explained in Section 3.2, taken from the "**webserver**" machine in the TSUGI Form where asked;
6. Post the result of the command `ip addr show`, as explained in Section 3.2, taken from the "**client**" machine in the TSUGI Form where asked;
7. Capture a screenshot of the leftmost side of the Browser in your host system, **showing clearly** the URL used and the page answered by the "**webserver**" machine. The content of the screenshot MUST be readable! Submit your answer in the TSUGI Form where asked;
8. Post the result of the first telnet interaction with the "webserver", as explained in section 4.1. Submit that result in the TSUGI Form where asked;
9. Post the result of the telnet interaction with the "webserver" **after correcting the DNS**, as explained in section 4.1. Submit that result in the TSUGI Form where asked;
10. Post the result of requesting a page from the "webserver" using **curl**, as explained in Section 4.2, 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.

3 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 **project directory** in your system (because each project has a specific **Vagrantfile**, but with different instructions 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_20_21-LAB01_support_files.zip` and **uncompress the content** into the **weblab** project folder. The zip archive contains a `Vagrantfile`, a file named `bootstrap_web.sh` and a folder named `html` (that contains in it 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 so, add the `vagrant-vbguest` Plugin with the command:

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

3.1 First Step

For this project we will use a box with a Ubuntu OS, named "**ubuntu/xenial64**", for all the virtual machines in the experiment.

Open the `Vagrantfile` **with an code 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|`).

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

Vagrant.configure(2) do |config|
 # 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"
     vb.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
     vb.memory = "256"
   end # of vb
  end # of web_config
end # of config
```

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

The assignment of IP addresses to the machines can be made, as illustrated is this example, to a **private** network address of Virtualbox in your system. Please note that you may need to adapt the values to the specific environment in your host system in order to avoid conflicting addresses.

It is always a good practice to verify if your Vagrantfile is syntactically and semantically correct (because the Vagrantfile is a "program"). For that purpose use the following command:

```
:~$ vagrant validate
```

If there is no validation error, you may 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 either `ipconfig /all` or `ipconfig -all` for windows, depending if you are using Powershell or a Bash-like command-line interface.

From the SSH session of the "webserver" verify that you have connectivity to your host system, using the Ping command as illustrated (use your own host IP address, not the one in this 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`
:~$
```

3.2 Second Step

Open again the Vagrantfile with your code editor. You will now create a new block of code to define a new Virtual Machine. For that purpose, you may start by replicating the block that defines the "webserver" server, and then change in that new code block all instances of "web_config" to "client_config".

Also must change the IP address of the new machine to a different value but within the same subnetwork, for example 192.168.56.11. **For this new server, remove from the code block you replicated, the instruction that forwards the port 80, all the instructions that configure shared folders, and the instruction that "provisions" the system via a "shell' script**.

Validate your modified Vagrantfile before proceeding, with the command:

```
:~$ vagrant validate
```

If the validation succeeded, you can now boot both "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 a new Virtual Machine is being created.

When all ready, establish independent 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 the network interfaces, as well as if both 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 with different IP addressing schemes in each machine?

4 HTTP Experiments

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

To start, open in YOUR HOST a browser of your choice and get the page with URL: `http://localhost:8080` or `http://127.0.0.1:8080`. If everything went well, you will be greeted with a **funny message**.

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

Try the following interaction with the "webserver" from the "client" machine:

```
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...

4.1 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 ao, run again the command:

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

4.2 Requesting a web page without a browser

There are other methods to request web pages, or transfer data from Web servers, without the need of a browser.

One of the methods is with a tool named **curl** (`https://curl.haxx.se`), used for transferring data with URLs, which comes already installed in most Operating Systems.

So, in the "client" machine use the following command, to get the `index.html` page from the "webserver":

```
vagrant@client:~$ curl http://webserver:80
```

5 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
==> webserver: Attempting graceful shutdown of VM...
```

Confirm that the statuses of the VMs is 'powered off' in order to prevent those machines from using resources. You can, of course, also destroy them if not needed anymore:

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