TÉCNICO LISBOA

RC 17/18	LAB ASSIGNMENT	Number:	3
Web and Email Scenarios		Issue Date:	06 Oct 2017
HTTP, SMTP, POP3		Due Date:	

Preliminary Notes

The instructions in this document are applicable to Computers at the IST Labs, which are already configured with the required tools. 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. The instructions are suitable for machines running the following Operating Systems:

- Microsoft Windows from version 7 up to 10
- Apple macOS from versions 10.8 'Mountain Lion' up to 11.13 'High Sierra'
- Debian-based Linux, such as Ubuntu (recommended) from versions 12.04 'Precise' to 16.04 'Xenial Xerus'.

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

1 Configuring Two Different Machines With Vagrant

In order to study the communication between systems we need at least two different machines. We will learn how to configure two different machines with Vagrant. This procedure is scalable so you can launch as many machines as you want (and your system allows).

We will need to have one `Vagrantfile` and as it is illustrated:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
Vagrant.configure(2) do |config|

  config.ssh.insert_key = false
  config.ssh.forward_x11 = true
  config.vbguest.auto_update = true

  #Server Configuration
  config.vm.define :webServer do |server_config|
    server_config.vm.box = "ubuntu/trusty64"
    server_config.vm.hostname = "webServer"
    server_config.vm.network :private_network, ip: "
      192.168.56.222"
    server_config.vm.synced_folder "data/core/configs", "/home/
      vagrant/.core/configs"
    server_config.vm.synced_folder "data/core/myservices", "/
      home/vagrant/.core/myservices"
    server_config.vm.provider "virtualbox" do |vb|
      vb.name = "webServer"
      opts = ["modifyvm", :id, "--natdnshostresolver1", "on"]
      vb.customize opts
      vb.memory = "2048"
    end
    server_config.vm.provision :shell, path: "bootstrap-core.sh"
  end

  #Client Configuration
  config.vm.define :webClient do |client_config|
    client_config.vm.box = "ubuntu/trusty64"
    client_config.vm.hostname = "webClient"
    client_config.vm.network :private_network, ip: "
      192.168.56.200"
    client_config.vm.synced_folder "data/core/configs", "/home/
      vagrant/.core/configs"
       client_config.vm.synced_folder "data/core/myservices", "
          /home/vagrant/.core/myservices"
    client_config.vm.provider "virtualbox" do |vb|
      vb.name = "webClient"
      opts = ["modifyvm", :id, "--natdnshostresolver1", "on"]
      vb.customize opts
      vb.memory = "2048"
    end
  client_config.vm.provision :shell, path: "bootstrap-core.sh"
  end
end # of config
```

The main objective is to create a code block for each machine, containing the specific provisioning commands. In this example the first machine will be named “webServer”, as seen in the code block that starts with `config.vm.define "webServer"do |server_config|`. The second one will be named “webClient”, starting with `config.vm.define "webClient"do |client_config|`. There are parameters that may be unique for each machine like IP addresses and hostnames.

You can now both the two machines ”webServer” and ”webClient” by issuing the following command just once:

```
:~$ vagrant up
```

You will observe that Vagrant will take a few minutes but when it finishes, there will be two machines running. To connect to each machine you should specify their name when using the `vagrant ssh` command as exemplified for the case of the webServser:

```
:~$ vagrant status
Current machine states:
webServer                   running (virtualbox)
webClient                   running (virtualbox)
:~$ vagrant ssh webServer
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@webServer:~$
```

Verify that you have connectivity between the host system and the virtual machines and that all system are inter-connectible [1]:

```
vagrant@webServer:~$  ping 192.168.1.200
PING 192.168.1.200 (192.168.1.200) 56(84) bytes of data.
64 bytes from 192.168.1.200: icmp_seq=1 ttl=63 time=0.443 ms
64 bytes from 192.168.1.200: 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.
```

2 HTTP Experiments

2.1 Starting a HTTP Server

HTTP is a client-server application layer protocol for distributed hypermedia information systems, and it is the basis of the **World Wide Web**. Now that we have two machines we need one to run a HTTP server and the other to run a HTTP client. We will use for this experiment a simple python implementation of a HTTP server by issuing the following command in order to run this server, listening in port 80:

```
vagrant@webServer:~$ sudo python -m  SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

[1]To discover what is your IP, open a new terminal and run "*ifconfig | grep inet*" for mac/linux or "*ipconfig /all*" for windows

2.2 Connecting to the server

In order to access to the ”webClient” machine, open a new command line window and with the `vagrant ssh` for that system you will get the following:

```
:~$ vagrant ssh webServer
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@webClient:~$
```

We will use the `telnet` application to connect to processes running in those systems. Try the following interaction with the webServer:

```
vagrant@webClient:~$ telnet webServer 80
```

You will observe that something is missing from the returned error. That missing piece is DNS, i.e., the Domain Name Service to “translate” the system name to its IP address.

2.3 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:

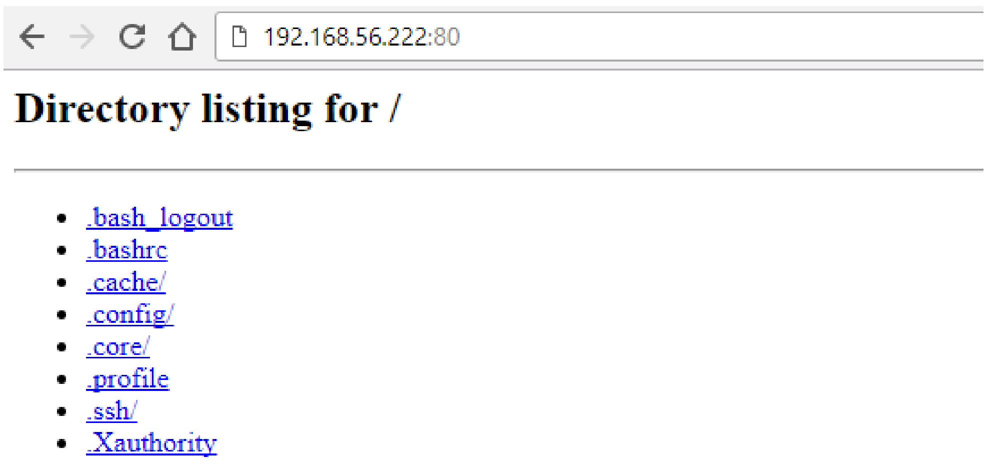
```
vagrant@webClient:~$ sudo sh -c "echo '192.168.56.222 webServer'
   >> /etc/hosts"
vagrant@webClient:~$ cat /etc/hosts
```

You should not have problems now, for the interaction and you should obtain a simple html page witha that command:

```
vagrant@webClient:~$ telnet webServer 80
Trying 192.168.56.222...
Connected to webServer.
Escape character is '^]'.
GET / HTTP/1.1
Host: WebServer
```

2.4 Connecting to the webServer with a browser

With a browser we can visualize this simple html file using the IP address of our “webServer” machine and the port where it server is running: 192.156.68.222:80


192.168.56.222:80

Directory listing for /

- .bash_logout
- .bashrc
- .cache/
- .config/
- .core/
- .profile
- .ssh/
- .Xauthority

Figure 1: Example of simple html page served by the webServer, in a browser window

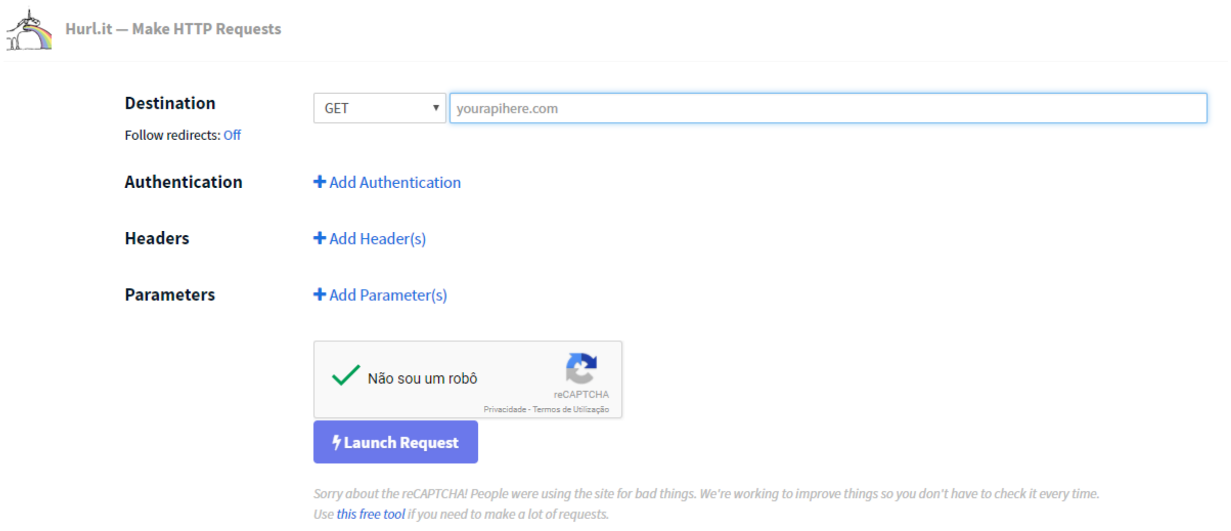
2.5 Using C.O.R.E

You can also use the C.O.R.E. emulator to execute this same experiment. Boot the machine and open a window for the "core-gui" (look at the LAB#1) and create a simple network with a router and two PCs. Run the experiment, annotaing theIP addresses of the PCs. In each PC open a terminal (bash) and in one of them a wireshark window. Start the HTTP server as explained earlier (Section 2.1) in one of the PCs and connect with telnet (Section 2.2) from the other one, and analyze wireshark capture.

2.6 Analysing HTTP messages with Web based tools

There are also some web based tools to study HTTP protocol messages. One of them is accessible at `www.hurl.it`. Analyze the responses to the request of the page with the following url: `http://help.websiteos.com/websiteos/example_of_a_simple_html_page.htm`.

RC 17/18	**LAB ASSIGNMENT**	**Number:**	3
Web and Email Scenarios		**Issue Date:**	06 Oct 2017
HTTP, SMTP, POP3		**Due Date:**	


Hurl.it — Make HTTP Requests

Destination: GET youraapihere.com
Follow redirects: Off

Authentication: + Add Authentication

Headers: + Add Header(s)

Parameters: + Add Parameter(s)

✓ Não sou um robô

reCAPTCHA
Privacidade - Termos de Utilização

Launch Request

Sorry about the reCAPTCHA! People were using the site for bad things. We're working to improve things so you don't have to check it every time.
Use this free tool if you need to make a lot of requests.

Figure 2: Example of the Hurl.it interface

3 SMTP Experiments

3.1 Connecting to a SMTP server

SMTP is the universally used protocol for email transfer. For our SMTP experiment we will use an external SMTP test server provided for study purposes, the “smtp.mailtrap.io”. So from any machine try the following interaction (note that the user credentials to be used are those in the example), where your inputs are represented in orange:

```
vagrant@webClient:~$ telnet smtp.mailtrap.io 2525
Trying 54.85.222.127...
Connected to mailtrap.io.
Escape character is '^]'.
220 mailtrap.io ESMTP ready
EHLO smtp.mailtrap.io
250-mailtrap.io
250-SIZE 5242880
250-PIPELINING
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-DSN
250-AUTH PLAIN LOGIN CRAM-MD5
250 STARTTLS
```

```
AUTH LOGIN
334 VXNlcm5hbWU6
ZTM5OTNkZDg1ZWVhNDU=
334 UGFzc3dvcmQ6
MDU1ODgxNTAxOTQ4YWY=
235 2.0.0 OK
MAIL FROM: <from@smtp.mailtrap.io>
250 2.1.0 Ok
RCPT TO: <to@smtp.mailtrap.io>
250 2.1.0 Ok
DATA
354 Go ahead
To: to@smtp.mailtrap.io
From: from@smtp.mailtrap.io
Subject: Hello world!
This is the test message...
.
250 2.0.0 Ok: queued
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

4 POP3 Experiments

POP3 is an email access client application-layer Internet standard protocol used by local e-mail clients to retrieve e-mail from a remote server over a TCP/IP connection. Despite this protocol being less and less used, it is simple and is one of the most adequate ones to illustrate the mechanisms used in the email service. So from any machine try the following interaction (note that the user credentials to be used are those in the example), where your inputs are represented in orange, and use wireshark once again to inspect the packets exchanged:

```
vagrant@webClient:~$ telnet smtp.mailtrap.io 1100
Trying 54.85.222.127...
Connected to mailtrap.io.
Escape character is '^]'.
+OK POP3 ready
USER e3993dd85eea45
+OK
PASS 055881501948af
+OK maildrop locked and ready
STAT
+OK 0 0
```

RC 17/18	LAB ASSIGNMENT	Number:	3
Web and Email Scenarios		Issue Date:	06 Oct 2017
HTTP, SMTP, POP3		Due Date:	

```
LIST
+OK 0 messages (0 octets)
.
QUIT
+OK Bye
Connection closed by foreign host.
```

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

Confirm that the statuses of the VMs is ’powered off’.

In Lab computers, also do the following (answering “y”), in order to ensure the system is cleaned from your experiments. Note however that the configuration files remain stored in your private area in AFS:

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

Confirm that there are no VMs listed.