TÉCNICO LISBOA

AGISIT 20/21	LAB ASSIGNMENT	Guide:	04
Multi-Node Scenarios		Issue Date:	20 Oct 2020
Configuration Management of a Web System - part 3		Report Due:	26 Oct 2020
Author: Prof. Rui Cruz		Due Date:	27 Oct 2020

1 Introduction

This lab experiment is the third of a series in which progressive levels of **Configuration Management and Deployment** will be tried. Vagrant will be used to create the virtual environment (see Figure 1), including one virtual machine with Ansible on it to act as the Management Node. Several additional Vagrant virtual machines will be created, as Infrastructure nodes to be managed by Ansible.

Ansible is used to automate many IT needs such as Cloud provisioning, Configuration management, Application deployment, Intra-service Orchestration, etc. **Ansible** models an IT infrastructure by describing how all systems inter-relate. Ansible is "agentless", meaning that it does not uses agents in the infrastructure nodes, making it easy to deploy – and most importantly, it uses a very simple language called **YAML** (Yet Another Markup Language), in the form of **Ansible Playbooks**. Ansible Playbooks are special text files describing the automation jobs in plain English.

Ansible works by connecting to the Infrastructure Nodes and pushing out small programs, called "Ansible Modules" to them. These programs are written to be *resource models* of the **desired state** of the system. Ansible executes those modules (over SSH by default), and automatically removes them when finished.

Ansible is typically installed on a "management machine", such as the one you have already used and configured since Part 1 and Part 2 of this Lab experiment, and that you will continue to use in this lab.

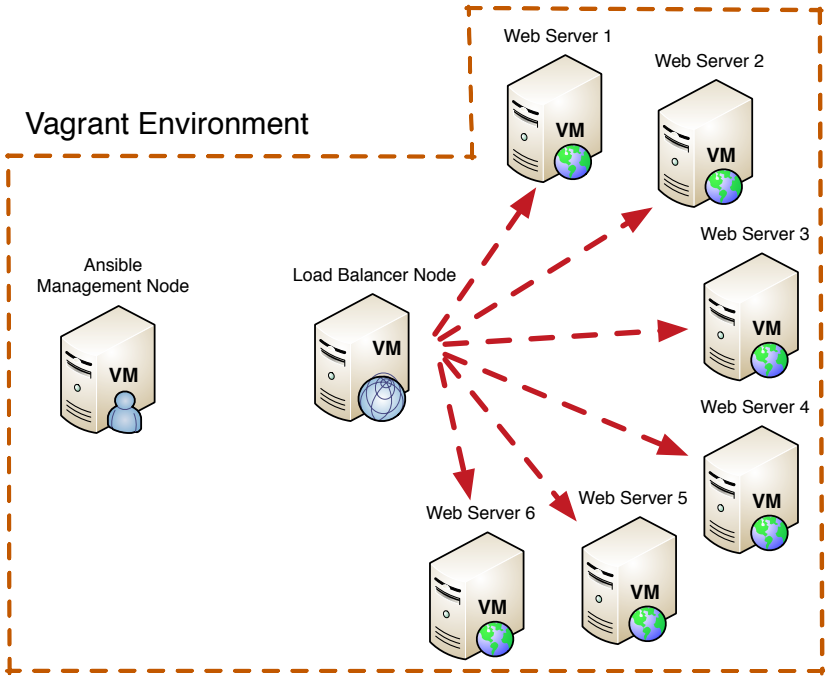

Figure 1: The Webfront Environment

AGISIT 20/21	LAB ASSIGNMENT	Guide:	04
Multi-Node Scenarios		Issue Date:	20 Oct 2020
Configuration Management of a Web System - part 3		Report Due:	26 Oct 2020
Author: Prof. Rui Cruz		Due Date:	27 Oct 2020

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.

Before proceeding you should verify if you have a "clean" environment, i.e., no Virtual Machine "instances" running (using precious resources in your system), or inconsistent instances in Vagrant and Virtualbox.

For that purpose run the `vagrant global-status` command and observe the results (as in the following example):

```
:~$ vagrant global-status
id       name      provider   state    directory
-----------------------------------------------------------------
28fb48a  mininet   virtualbox poweroff /Users/x/Projects/mininet
f0ccec2  web1      virtualbox running  /Users/x/Projects/multinode
f09c279  web2      virtualbox running  /Users/x/Projects/multinode
```

In the above example, you can observe that there are three Virtual Machines, being the first "mininet", which is **powered off**, but two "web" servers **still running**. It is **advisable to halt those VMs** if running, and then eventually **clean and destroy the VMs** if not needed anymore.

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.

AGISIT 20/21	LAB ASSIGNMENT	Guide:	04
Multi-Node Scenarios		Issue Date:	20 Oct 2020
Configuration Management of a Web System - part 3		Report Due:	26 Oct 2020
Author: Prof. Rui Cruz		Due Date:	27 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 the Moodle (`https://moodle.dei.tecnico.ulisboa.pt/login/index.php`) for the AGISIT course, you will find an **Assignment** for reporting the results from this specific Lab experiment.
3. The Assignment Link opens a **TSUGI CLOUD Web Form** containing the Questions to be answered;
4. The Assignment **Due Date** corresponds to the date it MUST be considered completed.
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 question boxes of the Assignment;
6. At the end of the Exercise Form you may comment your submission, or respond to feedbacks received (quoting them);
7. When finished answering the Exercise Form, click the button **Done** in the top left of the Form; You will be returned to the Moodle Assignment Link;
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:

1. (Exp.1) Report the Outputs of the command `ansible all -m shell -a "ntpq -p"` before and after the correction of the BUG, including your interpretation and description of the BUG. Write your report in the TSUGI Form where asked;

2. (Exp.2) Interpret the `site.yml` Playbook, briefly describing the purpose of the plays and the tasks in them. Write the report in the TSUGI Form where asked;

3. (Exp.2) Interpret briefly the configuration template files for the Web Service to figure out the meaning of some of the variables. Write your report in the TSUGI Form where asked;

4. (Exp.2) Captute the Output of the Playbook `site.yml` and include your interpretation of the results in the TSUGI Form where asked;

5. (Exp.3) Capture a Screenshot of your browser showing the page of URL `http://127.0.0.1:8080`, and submit in the TSUGI Form where asked;

6. (Exp.3) Capture a Screenshot of your browser showing the page of URL `http://127.0.0.1:8080/haproxy?stats`, and submit in the TSUGI Form where asked;

7. (Exp.3) Interpret some of the key metrics shown in the Load Balancer Statistics page after refreshing the page several times, reporting what have changed. Submit the report in the TSUGI Form where asked;

8. (Exp.3) Capture the output of the Benchmarking tool for requests -n = 1000 and for concurrency -c = 2 , and submit in the TSUGI Form where asked;

9. (Exp.3) Report the output of the Benchmarking tool with a different concurrency (-cparameter) and different requests value (-n parameter), refreshing the Load-Balancer Statistics page, and interpreting the changes observed, to submit inthe TSUGI Form where asked;

10. (Exp.4) Report and interpret the output of each execution (with the timer results) of the playbooks **parallel.yml** and **serial.yml**, in the TSUGI Form where asked;

WARNING. Submissions MUST BE MADE in the TSUGI CLOUD Web Form you can find in Moodle for this course. No other type of submission will be considered.

3 Setting up the Vagrant-Ansible Environment

To start the experiments, go to the directory created in parts 1 and 2 of this lab experiment, named **webfront**. In the **webfront** folder you already have the `Vagrantfile` that you created/modified.

We will use for this environment the same box with a Ubuntu OS, named "ubuntu/trusty64", which you should already have in your system, and so there is no need to download it again.

This experiment will make use of a **Management Node** named **mgmt**, a **Load Balancer Node** named **balancer** and several **Web Server Nodes** named **web1**, **web2**, etc., as illustrated in Figure 1.

Download the file `AGISIT_20_21_LAB_GUIDE_04_support_files.zip` from the course website. The archive contains an **examples** folder and you need to uncompress its content to the **webfront/examples** folder of your infrastructure. Verify that you will end up in the **webfront** folder with the following file structure:

```
.
|-- Vagrantfile
|-- bootstrap-mgmt.sh
|-- examples
    |-- ansible.cfg
    |-- files
    |   |-- ntp.conf
    |   |-- ntp.conf.j2
    |
    |-- frontend_templates
    |   |-- default-site.j2
    |   |-- haproxy.cfg.j2
    |   |-- index.html.j2
    |   |-- nginx.conf.j2
    |
    |-- inventory.ini
    |-- ntp-install.yml
    |-- ntp-remove.yml
    |-- ntp-template.yml
    |-- parallel.yml
    |-- serial.yml
    |-- site.yml
    |-- ssh-addkey.yml
```

3.1 Benchmarking Tool

For the experiments is this LAB we need to use a Benchmark command-line tool, which is a load generator for testing websites and HTTP APIs, commonly known as **Apache Benchmark** (`https://httpd.apache.org/docs/2.4/programs/ab.html`).

You will use the tool, but running from your own personal computers (not inside the Vagrant environment).

For NIX* based systems (Linux, macOS), proceed as follows:

- **ApacheBench** is included with the Apple macOS distribution and no configuration or installation is necessary.
- For Linux Debian distributions you should proceed as follows to install the tool:

```
$ sudo apt-get update
$ sudo apt-get install apache2-utils
```

- The command line call for **ApacheBench** in both macOS or Linux is simply `ab` with parameters, for example:

```
$ ab -n 1000 -c 2 http://127.0.0.1:8080/
```

For Microsoft Windows based systems, use the Chocolatey package manager by opening a ***Git-Bash*** Terminal and issue the following commands:

```
$   choco upgrade all
$   choco install superbenchmarker
```

The command line call of Superbenchmarker in Microsoft Windows is also simply `sb` with parameters, for example:

```
$ sb -n 1000 -c 2 -u http://127.0.0.1:8080/
```

3.2 Launching the Vagrant environment

Before launching the Vagrant environment run `vagrant status` to see that there are **at least 5 virtual machines defined**, i.e., a Management node, a Load Balancer, and a minimum of 3 Web servers.

```
:~/Project/webfront $ vagrant status
Current machine states:

mgmt                      not created (virtualbox)
balancer                  not created (virtualbox)
```

```
web1                      not created (virtualbox)
web2                      not created (virtualbox)
....
web6                      not created (virtualbox)

This environment represents multiple VMs. The VMs are all listed
   above with their current state.
```

In case there are machines in the "not created" state, you need to verify the coherence of all the "infrastructure" files to be used by Ansible, and then do `vagrant up` to lauch them. This will launch the machines into the Vagrant environment.

As soon as the machines are running you can login to the Management Node with `vagrant ssh mgmt`.

The Vagrantfile assigned predetermined IP addresses to each server, and then the bootstrap post-install script added those entries to the `/etc/hosts` file, allowing you to have name resolution for use in the hosts inventory for Ansible.

Do not forget to verify connectivity, by pinging the launched machines. For example, ping the Load Balancer node and then the other nodes:

```
vagrant@mgmt:~$ ping balancer
PING balancer (192.168.56.11) 56(84) bytes of data.
64 bytes from balancer (192.168.56.11): icmp_seq=1 ttl=64 time=1.11 ms
```

3.3 Confirming the SSH Trust

NOTE: If you did not destroy the VMs from previous Lab (and if you created at least the 5 VMs needed for this Lab), you can jump this section and proceed to Section 4.

If you either destroyed the VMs from previous LAB, or you need to add more VMs, you will need to recreate/redo the process, and so, proceed as indicated here.

As observed in Part 2 of this Lab, when working with Ansible and other Configuration Management tools, it is a MUST to establish a "**password less**" but secure access (i.e., a **SSH trust**) to the infrastructure nodes – for a "continuous integration system" – in order to deploy automated application updates on a frequent basis.

As you learned in previous LAB, we did that using the **ssh-addkey.yml** Ansible **Playbook**.

You can now deploy it to the remote client machines. We do this by running the command `ansible-playbook` with the `--ask-pass` option (because it is the first time and we do not have "**password less**" login configured yet in those remote machines).

The password to use is *vagrant*.

```
vagrant@mgmt:~/examples$ ansible-playbook ssh-addkey.yml --ask-
   pass
SSH password:
```

When finished, run an ad-hoc Ansible command, targeting all nodes, saying that we want to use the **ping module** to verify that everything works as expected and without a password.

```
vagrant@mgmt:~/examples$ ansible all -m ping
```

You should be able to see that a successful **ping pong** message from each of the remote machines is received, and that there was no error related with authenticity and Host key verification. This verifies that a whole chain of things is working correctly, i.e., the remote machines are up, Ansible is working, the hosts inventory is configured correctly, `ssh` is working with the vagrant user account, and that Ansible can remotely execute commands.

4 Now, on your own: Setting Up a Load Balanced Web Service

The end result of of this lab experiment will be a Vagrant environment with a fully functional Load Balancer (implemented with ***haproxy***, `http://www.haproxy.org`), with 3 (or up to six) web servers sitting behind it (implemented with ***nginx***, `https://www.nginx.com`). We will use Ansible to install all of the required packages, deploy configuration files, and start the correct services to each of these servers, all without logging manually into any of those nodes.

In the previous part of this Lab experiment you verified that the environment was successfully launched and that the VMs (**balancer** and **web1**...**webX**) were reachable from the host and among themselves using the `ping` command or the Ansible "ping" module.

It is now time to proceed with the Configuration of the servers.

4.1 Deploying Time Synchronization

It is a good practice to have all the servers in the infrastructure synchronized to the same Time Reference.

The synchronization of Time on servers and networks is often vitally important. Without synchronization, the time on individual servers will **slowly drift** away from each other at varying degrees until each has a significantly difference in time. In systems that rely on ordered events occurring at specific times or logging of events, this can be a real problem. For example, transactions generated by a server with a system time slower than another server may log a transaction as being received before the other server even though it was generated after it. In order to prevent that type of problem, a mechanism is required to disseminate accurate time to servers and network devices.

The Network Time Protocol (NTP) provides that mechanism through complex algorithms that maintain a high degree of time synchronization of servers on a network. NTP clients in the servers obtain highly precise timing information from Time Servers in the Internet (or from a Time Server in a Datacenter) in order to calibrate their own internal system time.

As you learned in previous Lab, you can deploy the Time service in all servers via an Ansible Playbook. Execute the following procedures, recording the results, as you will need them to answer the questions for this Lab Assignment

Experiment 1: Setting up the customized NTP Service. For this experiment you have the **ntp-template.yml** Playbook.

1. Study the Playbook carefully, and referencing the Ansible documentation site, `http://docs.ansible.com/ansible/playbooks.html`, describe what it does.
2. Run the Playbook, and after having successfully deployed the NTP service, run the command `ansible all -m shell -a "ntpq -p"` in the Management Node to query the NTP service. From the query output, you may confirm if the NTP service in the infrastructure nodes report with the correct Time.
3. Capture the Output of the Playbook (text). **Did the servers report the correct time, or is it there something apparently wrong?**
4. **NOTE:** As a matter of fact, **there is something wrong!** The Developer of the code for the configuration of the NTP service, left something incorrect there. **Your first mission is now to find the BUG and correct it!**

   Having found the BUG, use again the (corrected) **ntp-template.yml** Playbook to install the NTP service. After deployment, run the command `ansible all -m shell -a "ntpq -p"` in the Management Node to query the NTP service, and confirm that all servers are now synchronizing the Time correctly.
5. Interpret the Outputs of the `"ntpq -p"` before and command after correction of the error, describing what was the BUG in the code.

4.2 Deploying the Web Site Frontend

Start by analysing the **site.yml** playbook. You can notice that it consists of **three plays** that will allow to target specific groups of nodes, from within the same Playbook:

1. The first play (**# common**) targets all nodes in the hosts inventory. This is useful for laying down a common configuration across a fleet of machines.
2. The **# web** play, targets all the web nodes, for installing the web server, tweaking configuration files, and starting services.

3. The **# lb** play addresses the load balancer node, and you can notice a peculiar method in the **tasks:** that basically allows to iterate over a list of packages, and have them installed, without duplicating code blocks.

You will likely understand the majority of what each play is doing, as it is just installing the package(s), deploying configuration file(s) and service patterns, some using **Jinja2**, a template engine for Python (`https://palletsprojects.com/p/jinja/`), making sure that things are effectivelly started.

To get acquainted with these templates and the way to use the "variables" known by Ansible, you can now observe the "Ansible Facts". In Ansible, variables related to remote systems are called **facts**, while variables related to Ansible are called **magic variable**s.

Facts that are related to remote systems include operating systems, IP addresses, attached filesystems, etc.

Try the following to discover some of those facts related with any of the servers just configured, for example the balancer:

```
vagrant@mgmt:~/examples$ ansible balancer -m setup -a "filter=
    ansible_eth1"
balancer | SUCCESS => {
    "ansible_facts": {
        "ansible_eth1": {
            "active": true,
            "device": "eth1",
            "features": {
                "fcoe_mtu": "off [fixed]",
                .....
                            }
            ],
            "macaddress": "08:00:27:86:98:ea",
            "module": "e1000",
            "mtu": 1500,
                ....
            "timestamping": [
                "tx_software",
                "rx_software",
                "software"
            ],
            "type": "ether"
        },
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false
}
```

You can then reference, for example the IP address of that interface of the balancer server in a template as:

```
{{ ansible_facts.eth1.ipv4.address }}
```

4.2.1 The haproxy Load Balancer

HAProxy (community Edition `haproxy.org` is a Free, , very fast, Reliable, High Performance TCP/HTTP Load Balancer solution, particularly suited for very high traffic web sites.

The HAProxy configuration file (`haproxy.cfg`) guides the behavior of the load balancer. In this case, you may observe that you have in the **webfront/examples/frontend_template** folder, several files with the extension **j2**. Those files are **Jinja2** template files for configurations of the services to be installed, and one of them corresponds to the `haproxy.cfg`.

There are four essential sections in the `haproxy.cfg` configuration file. They are **global**, **defaults**, **frontend**, and **backend**. These four sections define how the server as a whole performs, what are the default settings, and how client requests are received and routed to the **backend** web servers.

You can also observe in the file, that some parameters are placed inside double curly braces, which correspond to variables that will be "replacede" with specific values by Ansible when configuring each server.

4.3 Experiments Procedure

For the Web System Experiment execute the following procedures, recording the results, as you will need them to answer the questions for this Lab Assignment:

Experiment 2: Setting up the web servers and the load balancer. For this experiment you have the **site.yml** Playbook.

1. Study the Playbook, and referencing the Ansible documentation site, `http://docs.ansible.com/ansible/playbooks.html`, describe in detail what it does. In some tasks of the Playbook, you noticed there are "notify:" keywords. Briefly describe their purpose.
2. Study also the configuration files in folder **templates** and try to figure out the meaning of some of the parameters, or the purpose of some of the *actions* therein. Report your interpretation of those parameters/actions. Describe also the difference between a normal file and a "template" file, as are the cases of the "Jinja2" (.j2) files (`http://jinja.pocoo.org`).
3. Play the **site.yml** Playbook with the `ansible-playbook` command.
4. Report the output of the Playbook and interpret the results.

5. When finished, and if all went okay, run the **site.yml** playbook again and interpret and report the output. This is a common practice when testing complex configurations in order to confirm that nothing was forgotten.

Experiment 3: Testing the web cluster. Recall that you have a desktop or laptop in which you are running Vagrant on – the **host machine** – and that the **Vagrantfile** created a virtual Vagrant environment that mapped port 8080 from the host machine, into the Load Balancer on port 80. So, when you connect to **127.0.0.1 port 8080** on the host machine, it will be redirected into the Web Front environment via the Load Balancer on port 80.

1. Open a web browser in your **host**, write the URL `http://127.0.0.1:8080` and hit return. You should be connected (into the Web Front environment), through the Load Balancer, and one of the Web servers should have served your request. Report the observed result.
2. Hit the refresh button on the web browser (forcing with the Shift key). Did something change? If so, repeat the refresh several times. Briefly report the results you have observed each time you refreshed the page.
3. Open a second tab in your browser and navigate to `http://127.0.0.1:8080/haproxy?stats`. You should land on the Load Balancer statistics page. Interpret some of the key aspects (metrics and or statuses shown). For reference, read `https://www.haproxy.com/blog/exploring-the-haproxy-stats-page/`.
4. Going back to the first tab, Hit the refresh button on the web browser several times, and then go to the second tab (the statistics page of the Load Balancer) and interpret the changes observed.
5. **If you are in a Microsoft Windows host** open a **Git-Bash** Terminal and run the command `sb -n 1000 -c 2 -u http://127.0.0.1:8080/`. This command tells the Benchamarker tool to send 1,000 requests, with a concurrency of 2, for the specified URL. Try to interpret the results from the benchmark tool. **If you are in a macOS or Linux host** open a Terminal and run the command `ab -n 1000 -c 2 http://127.0.0.1:8080/`. This command tells the Benchmarker tool to send 1,000 requests, with a concurrency of 2, for the specified URL. Try to interpret the results from the benchmark tool.
6. Go to the statistics page of the Load Balancer in the web browser, refresh the page and interpret the changes observed.
7. Modify the Benchmarker command for the concurrency parameter (-c) to a value still sustainable (try with some values, noting that in some host systems that value can be as high as 20 or more). Try to interpret the results from the Benchmarker tool and also from the statistics page of the Load Balancer.
8. Now modify the Benchmarker command for the requests parameter (-n) to a value of 10,000 but keeping the highest concurrency value reached previously. If there is no concurrency constraint (in case of error reduce the

concurrency value), try to interpret the new results from the benchmark tool and also from the statistics page of the Load Balancer.

Experiment 4: Parallel or **Serial** task execution in Ansible. One of the key things that makes a rolling deployment possible, is running tasks in a serial workflow, rather than a parallel workflow. Up until this point, all the experiments you have performed were focused on running things in parallel. For this experiment, however, you have two playbooks, the **serial.yml** and the **parallel.yml**. Both are targeting all hosts in the infrastructure, with one task defined to connect to each remote host and using the shell module to originate a **sleep for 5 seconds** in the remote host. These Playbooks have a very simple concept, but the implications are pretty big, especially when you want to do roll out updates for a very large number of systems.

1. Study each Playbook and describe what they do.
2. Play each Playbook but **prefixed with a timing command**, such as `time ansible-playbook parallel.yml` and `time ansible-playbook serial.yml`.
3. Interpret and Report the output of each Playbook comparing the values obtained.

Finishing the experiments: When finished the experiment, Stop the Virtual Machines and verify the global state of all active Vagrant environments on the system, issuing the following commands:

```
:~$ vagrant halt
:~$ vagrant global-status
```

Confirm that the statuses of the VMs is “powered off”. You can destroy these machines as they will not be used for the next Labs.