TÉCNICO LISBOA

AGISIT 20/21	LAB ASSIGNMENT	Guide:	03
Multi-Node Scenarios		Issue Date:	12 Oct 2020
Configuration Management of a Web System - part 2		Report Due:	16 Oct 2020
Author: Prof. Rui Cruz		Due Date:	20 Oct 2020

1 Introduction

This lab experiment is the second 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.

The objective of the experiments in this Lab is therefore to start using **Ansible** to model an IT infrastructure by describing how all systems inter-relate, and to automatically configure those systems.

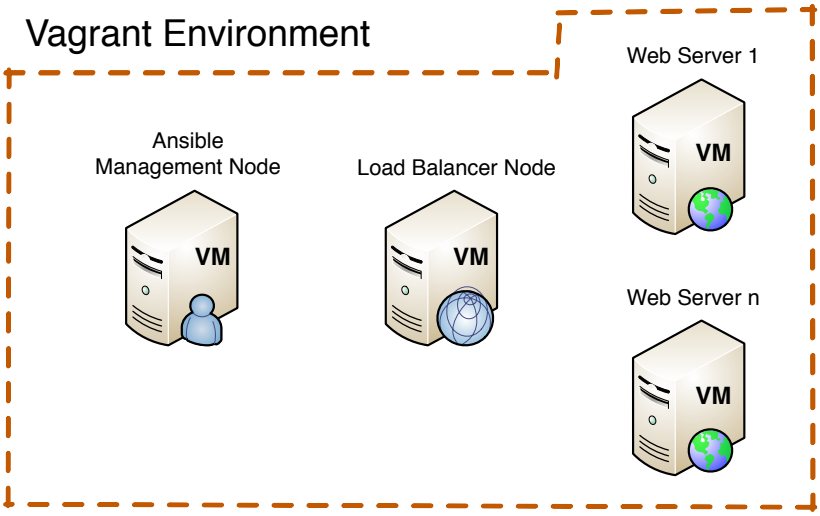

Figure 1: Vagrant Environment for Ansible Configuration Management

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:	03
Multi-Node Scenarios		Issue Date:	12 Oct 2020
Configuration Management of a Web System - part 2		Report Due:	16 Oct 2020
Author: Prof. Rui Cruz		Due Date:	20 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. Submit your modified "**Vagrantfile**" in the TSUGI Form where asked;
2. Submit your modified `inventory.ini` file in the TSUGI Form where asked;
3. Post the result of the command `ansible-playbook ssh-addkey.yml` in the TSUGI Form where asked;
4. Post the result of the command `ansible all -m ping` in the TSUGI Form where asked;
5. Post the result of the command `ansible all -m shell -a "uptime"` in the TSUGI Form where asked;
6. Post the result of the command `ansible all -m shell -a "uname -a"` in the TSUGI Form where asked;
7. Report your interpretation of the tasks of the **ntp-install.yml** Playbook in the TSUGI Form where asked;
8. Post the result of the execution of the **ntp-install.yml** Playbook in the TSUGI Form where asked;
9. Post the result of the command `ansible all -m shell -a "ntpq -p"` taken from the "**mgmt**" machine in the TSUGI Form where asked;
10. Post the result of execution of the **ntp-remove.yml** Playbook, together with your interpretation of what happened. Submit 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 Environment

To start the experiments, go to the directory created in **Part 1** 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.

In the **webfront** folder confirm that you have the following files and the `examples` directory, using the command `ls -lah` in Linux, macOS, or Windows (using Git-Bash).

```
# contents of the webfront project folder
~/Project/webfront $  ls -lah

drwxr-xr-x   9 user   staff    288B Oct  1  10:11 .
drwxr-xr-x  31 user   staff    992B Oct  1  10:11 ..
-rwxr-xr-x   1 user   staff    1.7K Oct  1  10:11 Vagrantfile
-rw-------   1 user   staff    711B Sep 20  2017  bootstrap-mgmt.sh
drwxr-xr-x  10 user   staff    320B Aug 12  16:11 examples
```

NOTE: As there were some slight changes due to recent updates in some of the Tools to use, and also security fixes in Hosts Operating Systems, it is advisable to Destroy the previously created VMs from Part 1 of this lab experiment.

Please review also your Vagrantfile, comparing it with the one you will obtain from the `AGISIT_20_21_LAB_GUIDE_03_support_files.zip` you can download from the course website, adapting it to follow some slight changes in the shared folders instructions.

You also need to replace the `ssh-addkey.yml` Playbook in the `examples` folder due to the update of some "deprecated" instructions, as well as replace the ansible.cfg file (as an additional "default" configuraton parameter is also needed to be present).

3.1 Launching the Vagrant environment

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

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

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

```
web1                      not created (virtualbox)
....
webX                      not created (virtualbox)

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

The listed machines are all in the "not created" state, so lets launch them up by running, `vagrant up`. This will start up the machines into the Vagrant environment, and the post-boot script (`bootstrap-mgmt.sh`), installs Ansible on the Management Node, and configures the "hosts" file so that the Management Node will have **name resolution** to be used by Ansible for the hosts defined in the inventory `inventory.ini` for this infrastructure.

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

You can verify the connectivity, by pinging the launched machines. For example, **ping** (from your host and from the Management Node) the Load Balancer node and then the other nodes, confirming that you have connectivity:

```
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
```

3.2 Ensuring SSH Connectivity Between Nodes

Now that you verified that the environment was successfully launched and that the VMs were reachable, you are in conditions to proceed.

When using an Automation Engine such as **Ansible**, the connectivity to remote machines from the Management Node is done via `ssh` (Secure Shell). One issue that we need to deal with, is that, if the Management Node has not yet connected to a machine via SSH, you will be prompted to verify the `ssh` **authenticity** of the remote node.

So, in order to continue, try to connect from the **mgmt** node to the **web1** node using `SSH`. You should **Answer: No** when receiving the response, that should be similar to the following:

```
:~/Project/webfront $ vagrant ssh mgmt

vagrant@mgmt:~$ ssh web1
The authenticity of host 'web1 (192.168.56.21)' cannot be
   established.
ECDSA key fingerprint is 96:df:64:6f:a7:9f:86:50:8f:aa:56:dd
   :71:07:d5:cb.
Are you sure you want to continue connecting (yes/no)? no
Host key verification failed.
```

Attention: the following steps **must be executed carefully**. Read through before typing commands and make sure you do not step ahead in the answers as **some steps are not easily reversible!**.

As you could see, you were prompted to verify the authenticity of **web1**. You would find the exact same behavior when trying to open SSH to the other systems in your infrastructure. If nothing is done, this error condition will cause Ansible to prompt you whenever it tries to reach a host. For example, let's run the Ansible **ping module**, which is not exactly like a traditional ICMP ping, but rather the equivalent of a *Hello World* program for Ansible to verify that it can login to the remote machines. The command for **web1** host is: `ansible web1 -m ping`. For now, **answer No** to the response:

```
vagrant@mgmt:~$ ansible web1 -m ping
The authenticity of host 'web1 (192.168.56.21)' cannot be
   established.
ECDSA key fingerprint is 96:df:64:6f:a7:9f:86:50:8f:aa:56:dd
   :71:07:d5:cb.
Are you sure you want to continue connecting (yes/no)? no
web1 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Host key
       verification failed.\r\n",
    "unreachable": true
}
```

What you have seen from the response to the command was that Ansible could not connect due to authentication to the **web1** host.

Given the error message saying that Ansible cannot connect to that host, what would happen if your infrastructure would be composed by 25, 50, 100 or a 1,000 machines? Would you be willing to constantly be prompted? **Perhaps not!**

There are several ways of dealing with this issue:

1. Populate **known_hosts** file: Use `ssh-keyscan` command to populate the file at the Management Node, with keys of the machines from the environment; $\longrightarrow$ this option still needs password authentication.
2. **Establishing a SSH Trust**. $\longrightarrow$ secure and appropriate for automated processes.
3. **Turn off the authentication**: via a configuration in the `ssh` *config* file; $\longrightarrow$ insecure and dangerous!

It is obvious that **option 3** is **not an option!**

3.3 First option: Knowing the Hosts

This option may be satisfactory for an infrastructure with just a few systems, or for a development environment. This is typically what happens when you launch an environment with Vagrant for local testing or development.

Let's run `ssh-keyscan` against **web1**:

```
vagrant@mgmt:~$ ssh-keyscan web1
# web1 SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
web1 ecdsa-sha2-nistp256
   AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBN2....
# web1 SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
web1 ssh-rsa
   AAAAB3NzaC1yc2EAAAADAQABAAABAQDNt1YP04vLKWx5PmkRbXX7rCC....
```

As you can see, **web1**, returns two `ssh` public keys, that can be used to populate the **known_hosts** file of the Management Node.

So lets run again `ssh-keyscan` for the Load Balancer machine, **balancer**, as well as for **web1**, **web2**. . . **webX**, and then pipe the output into **.ssh/known_hosts** file:

```
vagrant@mgmt:~$ ssh-keyscan balancer web1 web2 >> .ssh/known_hosts
# balancer SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
# balancer SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
# web1 SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
# web1 SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
# web2 SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
# web2 SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
```

You can now verify that the **known_hosts** file in the Management Node contains a bunch of keys from the remote machines:

```
vagrant@mgmt:~$ cat .ssh/known_hosts
```

Now that the hosts are "known" at the Management Node, we will use the Ansible **ping module** to all hosts. Since we are using `ssh` to connect remotely as the *vagrant user*, we want Ansible to prompt us, for now, to enter the password. The password is just *vagrant*:

```
vagrant@mgmt:~$ ansible all -m ping --ask-pass
SSH password:
web1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
balancer | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
web2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

You can 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.

3.4 Second Option: Establishing a SSH Trust

To establish a "**password less**" but secure access to the infrastructure nodes, independently of their number – for a "continuous integration system" – in order to deploy automated application updates on a frequent basis, we need to opt for establishing a **SSH trust**, between the Management Node and the other nodes of the infrastructure.

We will do that using Ansible itself via a **Playbook**. It is now time to learn a bit about how Ansible works.

In order to proceed, please **delete** the `.ssh/known_hosts` file that was created in previous section.

Let's look at the files in the **examples** folder of the Management Node:

```
vagrant@mgmt:~/examples$ ls
ansible.cfg  files  inventory.ini  ntp-install.yml  ntp-remove.yml
    ntp-template.yml  ssh-addkey.yml
```

You may notice a file named **ssh-addkey.yml**, which is an Ansible **Playbook**, and so, let's see its content:

```
vagrant@mgmt:~/examples$ cat ssh-addkey.yml
---
- hosts: all
  become: yes
  become_method: sudo
  gather_facts: no
  remote_user: vagrant

  tasks:

  - name: install ssh key
    authorized_key:
      user: vagrant
      key: "{{ lookup('file', '/home/vagrant/.ssh/id_rsa.pub') }}"
      state: present
```

Playbooks are Ansible's configuration, deployment, and orchestration language. The file is formatted in YAML (Yet Another Markup Language), human-readable and very easy to understand. You can learn about Playbooks in more detail at the Ansible documentation site: `http://docs.ansible.com/ansible/playbooks.html`.

A **Playbook** is composed of one or more "plays" in a list. Looking at the **ssh-addkey.yml** you can see that we have only the **hosts** play, and the task to deploy an authorized **ssh key** onto a remote machine, which will allow Ansible to connect without using a password.

NOTE: look carefully at the indentation of the Playbook lines, as it is crucial for the interpretation of the content of the YAML file:

- hosts: This "play" defines which hosts in the hosts inventory we want to target . For this "play" the hosts will bee "**all**"

remote_user: The name of the user.

become: We want to run these commands as super user.

become_method: The method is **sudo**.

gather_facts: By default, "facts" are gathered when a Playbook is run against remote hosts. These facts include things such as *hostname*, network addresses, etc. These facts can be used to alter the Playbook execution behavior, or to be included in configuration files. We are not going to use them just yet, so "facts" are turned **off**.

tasks: This section of the **hosts** play, is where we define the tasks we want to run against the remote machines.

- name: Each task has a *name* associated with it, and it is just an arbitrary meaningful title.

authorized_key: This is the Ansible **module** to be used in the task.

user The remote authorized *vagrant* user.

key The module is instructed to add an authorized key, and where on the Management Node the key file is to be looked up.

state the module is instructed to make sure that the key exists on the remote machine.

It is a good practice to verify if the syntax of the playbooks is correct. For that purpose, run the following command against each of the playbooks, for example, in the case of the **ssh-addkey.yml** playbook if synstax would not be correct, some warning or error would result, similar to te following:

```
vagrant@mgmt:~/examples$ ansible-playbook ssh-addkey.yml --syntax-
   check
[DEPRECATION WARNING]: Instead of sudo/sudo_user, use become/
   become_user and make sure become_method is 'sudo' (default).
   This feature will be removed in version 2.9. Deprecation
   warnings can be disabled by setting deprecation_warnings=False
   in ansible.cfg.

playbook: ssh-addkey.yml
```

In this case you may observe some warnings related with “deprecated” syntax (although still valid, but just up to a future specific release of Ansible). What you would need to do, is to correct the syntax.

Continuing with the method for establishing SSH trust, you can verify that on the Management Node, in the **.ssh** directory (of the home user), there is no public RSA key:

```
vagrant@mgmt:~$ ls -la .ssh/
total 16
drwxr-xr-x 2 vagrant vagrant 4096 Nov  3 21:35 .
drwxr-xr-x 6 vagrant vagrant 4096 Nov  8 11:47 ..
-rw-r--r-- 1 vagrant vagrant  486 Nov  3 17:20 authorized_keys
-rw-rw-r-- 1 vagrant vagrant 2322 Nov  8 13:05 known_hosts
```

Let’s then create one key by running `ssh-keygen -t`, and specify the type of key we want to create, which will be **RSA**, and then tell how long a key we want with parameter **-b**. After that, verify that the keys have been created, by listing the contents of **.ssh**.

Please hit **ENTER** to the prompts, and **do not enter a password**.

```
vagrant@mgmt:~$ ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vagrant/.ssh/id_rsa.
```

```
Your public key has been saved in /home/vagrant/.ssh/id_rsa.pub.
The key fingerprint is:
54:34:14:74:f6:38:9d:bd:64:4a:bd:83:20:f4:be:96 vagrant@mgmt
The key's randomart image is:
+--[ RSA 2048]----+
|          =B.o     |
|         ...+ +.o  |
|         .. oo.++.|
|        .  o o.= o|
|         S  . o + |
|             o   .|
|            E      |
+-----------------+
```

You can now verify that in the **.ssh** directory (of the home user), there is a KEYPAIR of a public and private RSA key:

```
vagrant@mgmt:~$ ls -la .ssh/
total 16
drwxr-xr-x 2 vagrant vagrant 4096 Nov  3 21:35 .
drwxr-xr-x 6 vagrant vagrant 4096 Nov  8 11:47 ..
-rw-r--r-- 1 vagrant vagrant  486 Nov  3 17:20 authorized_keys
-rw------- 1 vagrant vagrant 1679 Oct  7 22:40 id_rsa
-rw-r--r-- 1 vagrant vagrant  394 Oct  7 22:40 id_rsa.pub
-rw-rw-r-- 1 vagrant vagrant 2322 Nov  8 13:05 known_hosts
```

Now that we have fulfilled the requirements of generating a local RSA keypair for the Vagrant user, lets deploy it to our remote client machines. We do this by running the command `ansible-playbook`, and then the Playbook name, in this case **ssh-addkey.yml**.

We also need to add the `--ask-pass` option, since 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:

PLAY [all] ********************************************************

TASK [install ssh key] ********************************************
changed: [balancer]
changed: [web2]
changed: [web1]

PLAY RECAP ********************************************************
balancer                   : ok=1    changed=1    unreachable=0    failed=0
```

```
web1                       : ok=1    changed=1    unreachable=0    failed=0
web2                       : ok=1    changed=1    unreachable=0    failed=0
```

The Playbook response says that it changed three nodes, **web1**, **web2** ... **webX**, and **balancer**, to deploy the authorized key from the Management Node. In the end we are given a Playbook "recap" section that provides with the **okay**, **changed**, **unreachable**, and **failed** tallies per node.

Lets just rerun the `ansible-playbook` command for **ssh-addkey.yml** but removing the `--ask-pass` option, as we are not going to need that anymore. You will notice that in our previous playbook run, things changed, but in the most recent run, the tasks are all green, and the playbook recap is green too, meaning that nothing was changed.

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

TASK [install ssh key] *********************************************
ok: [web2]
ok: [web1]
ok: [balancer]

PLAY RECAP *********************************************************
balancer                   : ok=1    changed=0    unreachable=0    failed=0
web1                       : ok=1    changed=0    unreachable=0    failed=0
web2                       : ok=1    changed=0    unreachable=0    failed=0
```

What you are looking at here is a good example of **idempotence**, meaning that Ansible scripts can be run many times in a row, and nothing will change, unless a change needs to be made. So, in this example, Ansible was smart enough to check the clients authorized key file, and see that it already exists, so it does not do anything.

Let's 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
web2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
web1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
balancer | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

3.5 Ansible Ah-hoc Shell Commands

Besides providing an automated method for Configuration Management based on Playbooks, Ansible also provides a very powerful method for interactively performing actions, namely the *ad-hoc* command to run remote shell commands at the hosts.

For example, if you want to know the uptime of your hosts you can just do the following (with example of results):

```
vagrant@mgmt:~/examples$ ansible all -m shell -a "uptime"
balancer | SUCCESS | rc=0 >>
 23:05:57 up 14:21,  1 user,  load average: 0.00, 0.01, 0.05

web1 | SUCCESS | rc=0 >>
 23:05:57 up 14:20,  1 user,  load average: 0.08, 0.03, 0.05

web2 | SUCCESS | rc=0 >>
 23:05:57 up 14:20,  1 user,  load average: 0.08, 0.03, 0.05
```

You are returned the command output from all machines in the hosts inventory, but you could limit this to various groups, or individual machines.

Another example is for knowing what kernel versions are running on the hosts:

```
vagrant@mgmt:~/examples$ ansible all -m shell -a "uname -a"
web2 | SUCCESS | rc=0 >>
Linux web2 3.13.0-100-generic #147-Ubuntu SMP Tue Oct 18 16:48:51
   UTC 2016 x86_64 x86_64 x86_64 GNU/Linux

web1 | SUCCESS | rc=0 >>
Linux web1 3.13.0-100-generic #147-Ubuntu SMP Tue Oct 18 16:48:51
   UTC 2016 x86_64 x86_64 x86_64 GNU/Linux

balancer | SUCCESS | rc=0 >>
Linux lbal 3.13.0-100-generic #147-Ubuntu SMP Tue Oct 18 16:48:51
   UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
```

Or, maybe you want to reboot all the **web** nodes, for example:

```
vagrant@mgmt:~/examples$ ansible web -m shell -a "/sbin/reboot" -b
```

Note that the ”-b” command must be used whenever the command require administrative privileges and therefore the remote agent must ”become” root with ”sudo”.

4 On your own: Install and Uninstall a Service

The pattern in this experiment is really useful for laying down a common build, or standard package set, across a large number of machines. When you are running a Data Center with a few hundred machines, you will likely want to have common **DNS (Domain Name Service)** settings, **NTP (Network Time Protocol)** time settings, keys and accounts, security tweaks, etc. Ansible is perfect for that type of job, using just a couple of Playbooks to perform all the necessary work.

In the **examples** folder there are several Playbooks and also some configuration files (in folder **files**) for a service (in this case the **Network Time Service**.

```
vagrant@mgmt:~/examples$ ls -l

-rwxr-xr-x 1 vagrant vagrant   50 Apr 21  2015 ansible.cfg
drwxr-xr-x 2 vagrant vagrant 4096 Apr 21  2015 files
-rwxr-xr-x 1 vagrant vagrant   71 Nov  3 18:47 inventory.ini
-rwxr-xr-x 1 vagrant vagrant  457 Nov  9 22:27 ntp-install.yml
-rwxr-xr-x 1 vagrant vagrant  158 Nov  9 22:42 ntp-remove.yml
-rwxr-xr-x 1 vagrant vagrant  478 Nov  9 21:59 ntp-template.yml
-rwxr-xr-x 1 vagrant vagrant  260 Nov  9 16:36 ssh-addkey.yml
```

The objective of those Playbooks is to install/uninstall the required packages for the **NTP** service, as well as starting the service with a certain configuration, **in several web servers** and **in the Load Balancer**.

For that purpose execute the following experiments:

Experiment 1: Setting up the **NTP** service with the **ntp-install.yml** Playbook.

- Study the Playbook, and referencing the Ansible documentation site, `http://docs.ansible.com/ansible/playbooks.html`, try to describe what it does. (your description will be needed to answer a question in this Lab Assignment).
- Play the **ntp-install.yml** Playbook with the `ansible-playbook` command.
- Collect the output of the Playbook and interpret the results in a text file. (your description will be needed to answer a question in this Lab Assignment)
- Run the command `ansible all -m shell -a "ntpq -p"` in the Management Node to query the NTP service and capture output to a text file.

Experiment 2: Uninstall the **NTP** Service. For this experiment you have the **ntp-remove.yml** Playbook.

- Play the **ntp-remove.yml** Playbook with the `ansible-playbook` command.
- Collect the output of the Playbook and interpret the results in a text file. (your description will be needed to answer a question in this Lab Assignment).