NYU COURANT INSTITUTE OF MATHEMATICAL SCIENCES

Infrastructure as Code

Fall 2020, CSCI-GA 2820, Graduate Division, Computer Science

Instructor:

John J Rofrano

Senior Technical Staff Member | DevOps Champion
IBM T.J. Watson Research Center
rofrano@cs.nyu.edu (@JohnRofrano)

What Will You Learn?

What are Configuration Management Systems

Why would you use them for DevOps

Difference between Agent and Agent-less Solutions

How to use Ansible to manage servers (hands-on)

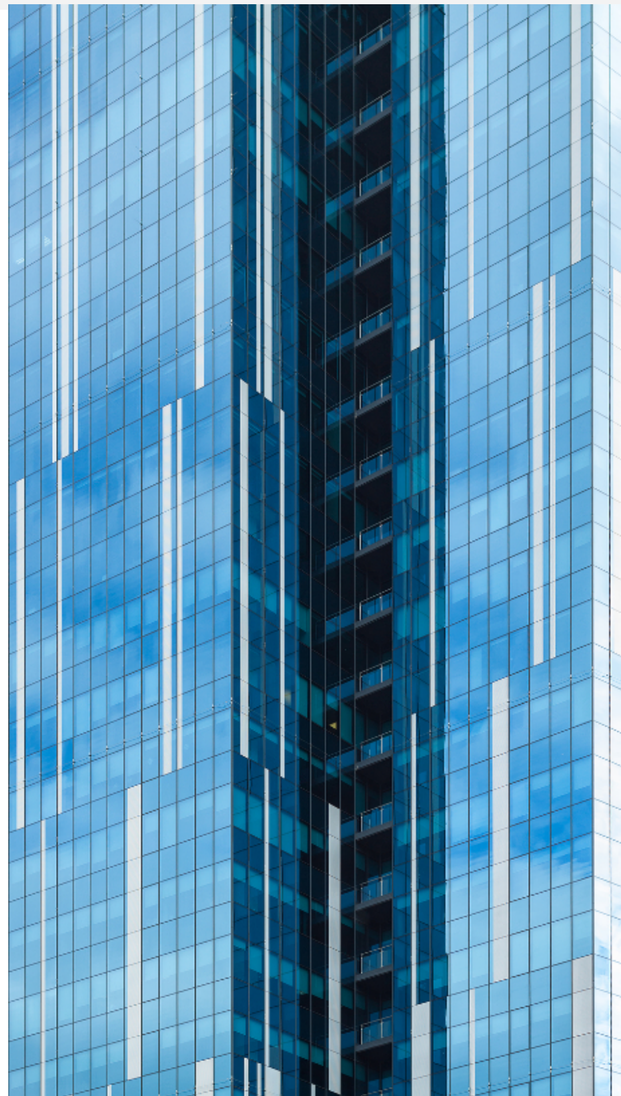
Source for This Lab

- The source for this lab can be found on GitHub at:

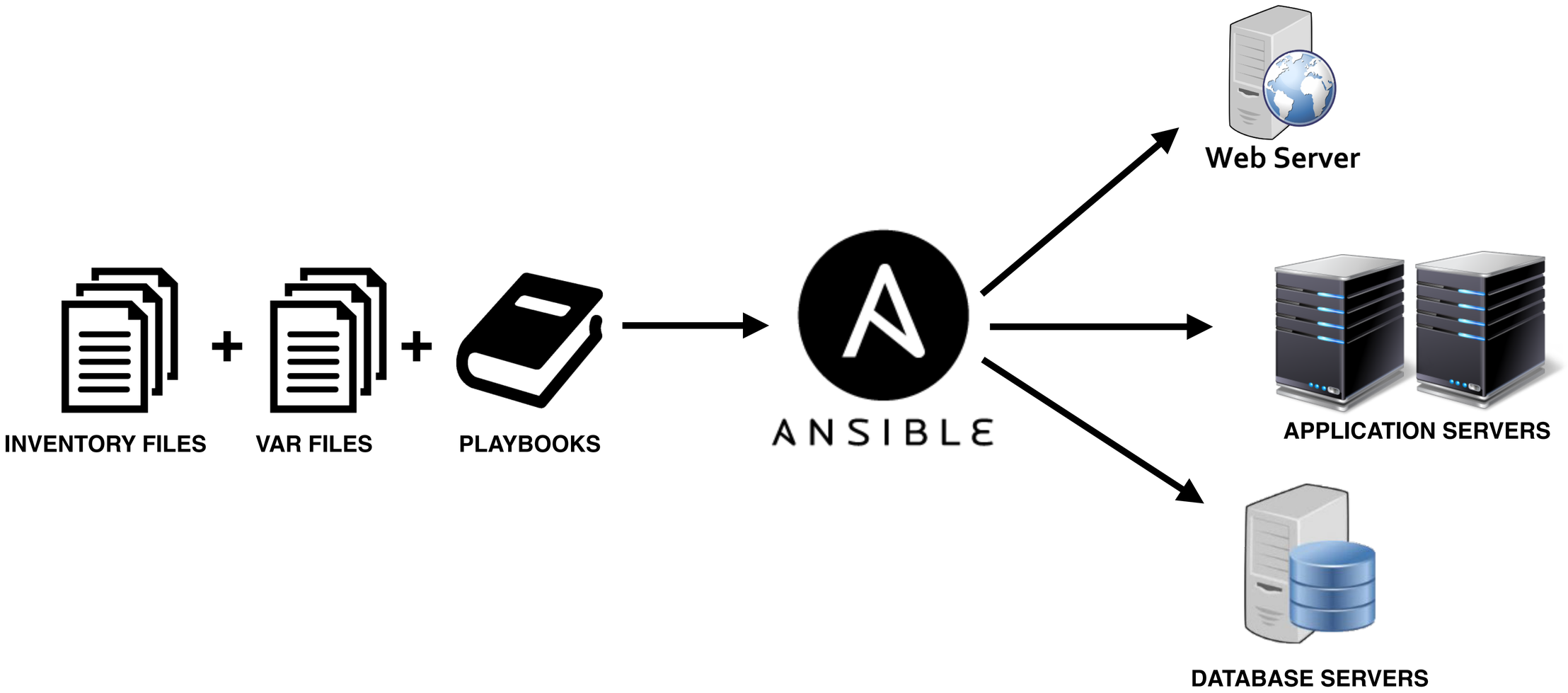
```
$ git clone https://github.com/nyu-devops/lab-ansible.git
$ cd lab-ansible
$ vagrant up
```

Infrastructure as Code

- Describe your infrastructure in textual format and configure them using that description
- Never perform system and software configurations manually
  - Use templates / scripts describing how to install / configure systems /devices /software / users
- Configuration Management Systems make this possible



Infrastructure as Code



Infra-As-Code Examples

```
class httpd {
  package { 'httpd-devel':
    ensure => installed,
  }
  service { 'httpd':
    ensure => running,
    enable => true,
    subscribe => Package['httpd-devel'],
  }
}
```


```
# /srv/salt/mysql.sls
mysql:
 pkg.installed:
  - name: mysql-server
 service.running:
  - enable: True
  - require:
   - pkg: mysql-server
```


```
package "ntp" do
    action [:install]
end

template "/etc/ntp.conf" do
    source "ntp.conf.erb"
    variables( :ntp_server => "time.nist.gov" )
    notifies :restart, "service[ntpd]"
end

service "ntpd" do
    action [:enable,:start]
end
```


```
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

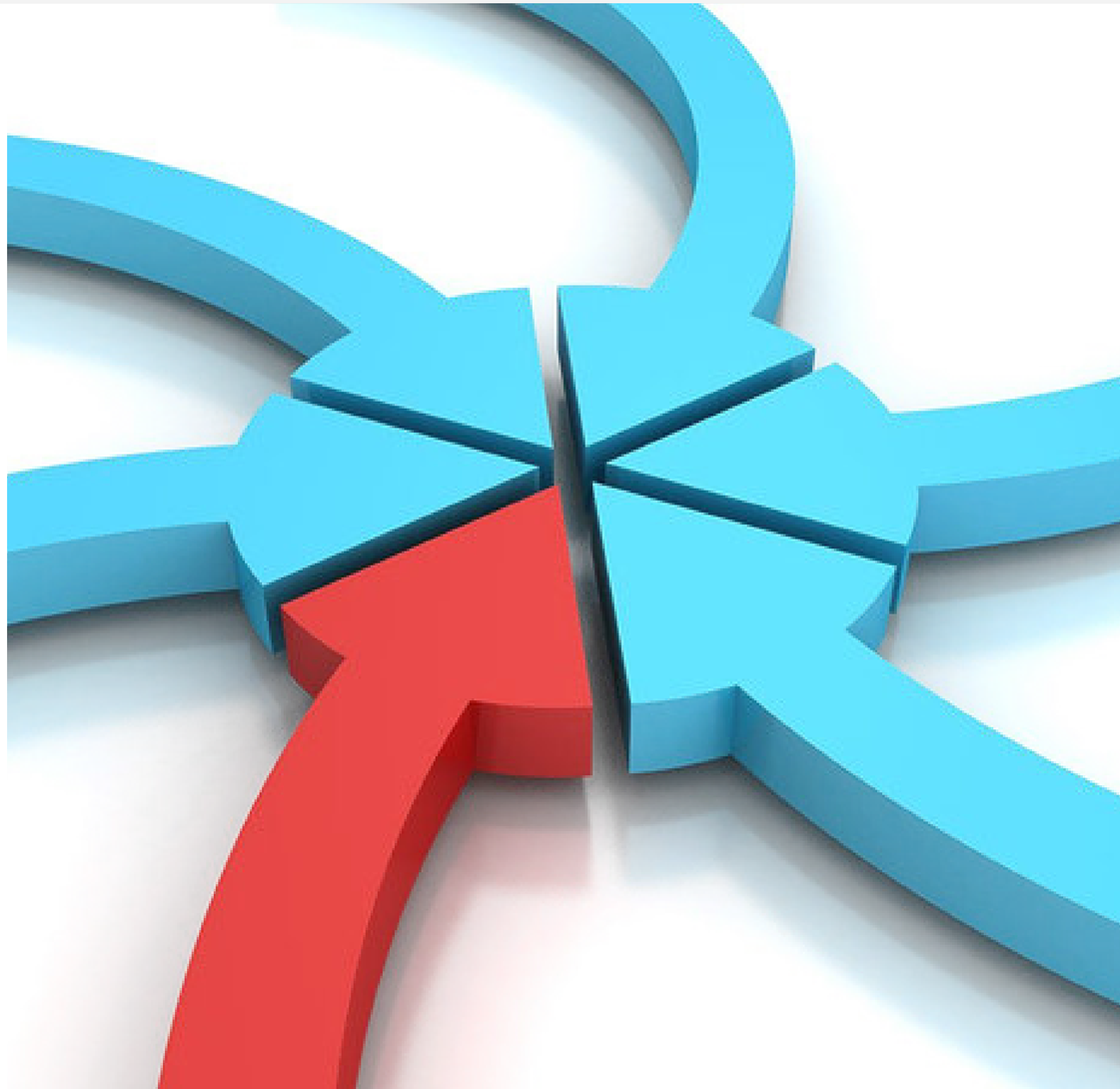
In the Beginning...

CFEngine

- CFEngine is an open source configuration management system, written by Mark Burgess in 1993
- Its primary function is to provide automated configuration and maintenance of large-scale computer systems, including the unified management of servers, desktops, consumer and industrial devices, embedded networked devices, mobile smartphones, and tablet computers
- One of the main ideas in CFEngine is that changes in computer configuration should be carried out in a **convergent** manner

Convergence

- Rather than describing the steps needed to make a change, the language describes the final state in which one wants to end up
- The agent then ensures that the necessary steps are taken to end up in this "policy compliant state"
- Thus, configuration can be run again and again, whatever the initial state of a system, and it will end up with a predictable result



Modern History

Tool	puppet labs	CHEF	SALTSTACK	ANSIBLE
First Release	2005	2009	2011	2012
Out-of-order execution	Yes	No	Yes	No
Push / Pull	Pull (Agents)	Pull (Agents)	Both	Push (Agentless)
Notes	More OSs supported, Large user base	Easier to learn	Infrastructure as data	Very simple, best for memory constrained environments (IoT)

Agent vs Agent-less

- Advantages of **Agent** based
  - Rich client can do more and run on a schedule
  - Low volume of network traffic
- Advantages of **Agent-less** based
  - Small memory footprint and support of rare systems
  - No need to run an agent all the time on each server


ANSIBLE

Ansible Design Principles

- Have a dead simple setup process and a minimal learning curve
- Manage machines very quickly and in parallel
- Avoid custom-agents and additional open ports, be agentless by leveraging the existing SSH daemon
- Describe infrastructure in a language that is both machine and human friendly
- Focus on security and easy auditability/review/rewriting of content
- Manage new remote machines instantly, without bootstrapping any software
- Allow module development in any dynamic language, not just Python
- Be usable as non-root
- Be the easiest IT automation system to use, ever

Ansible Terminology

The following list contains a quick overview of the most relevant terms used by Ansible:

- **Control Node:** the machine where Ansible is installed, responsible for running the provisioning on the servers you are managing.
- **Inventory:** an INI file that contains information about the servers you are managing.
- **Playbook:** a YAML file containing a series of procedures that should be automated.
- **Task:** a block that defines a single procedure to be executed, e.g.: install a package.
- **Module:** a module typically abstracts a system task, like dealing with packages or creating and changing files.
- **Role:** a set of related playbooks, templates and other files, organized in a pre-defined way to facilitate reuse and share.
- **Play:** a provisioning executed from start to finish is called a play.
- **Facts:** global variables containing information about the system, like network interfaces or operating system.
- **Handlers:** used to trigger service status changes, like restarting or reloading a service.

Ansible Components

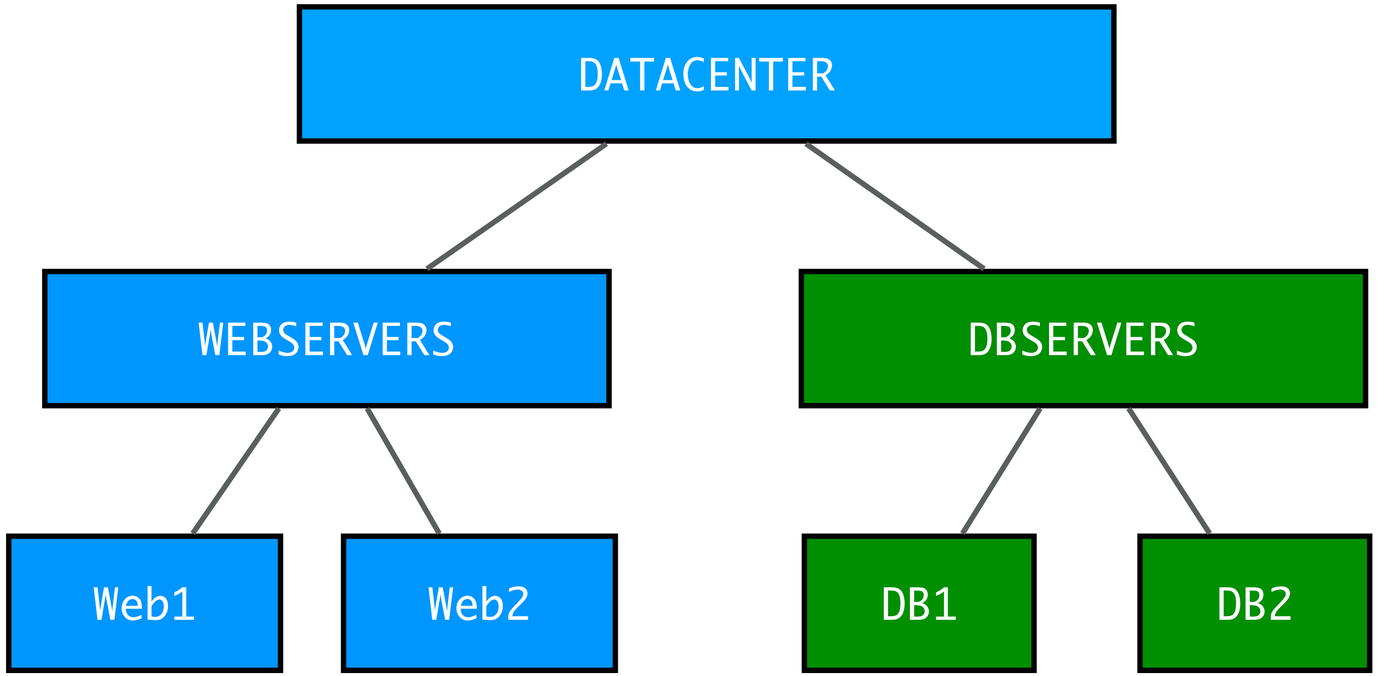
Inventory	The list of servers and optionally variables to operate on
Playbook	The collection of "plays" in YAML format to operate on the servers
Plays	A list of tasks and handlers to execute on each targeted server
Tasks	Instructions to cary out the configuration

Inventory File

- The inventory files tells Ansible what servers to act on
- Ansible allow you to place servers into groups (e.g., webservers, dbservers) and then make changes on entire groups of servers at the same time instead of changing them one-by-one
- Groups allow you to specify servers of the same type

```
web1 ansible_ssh_host=192.168.33.20
db1 ansible_ssh_host=192.168.33.30

[webservers]
web1

[dbservers]
db1

[datacenter:children]
webservers
dbservers
```

Hierarchy of Groups




```
# Inventory file

[webservers]
web1
web2

[dbservers]
db1
db2

[datacenter:children]
webservers
dbservers
```

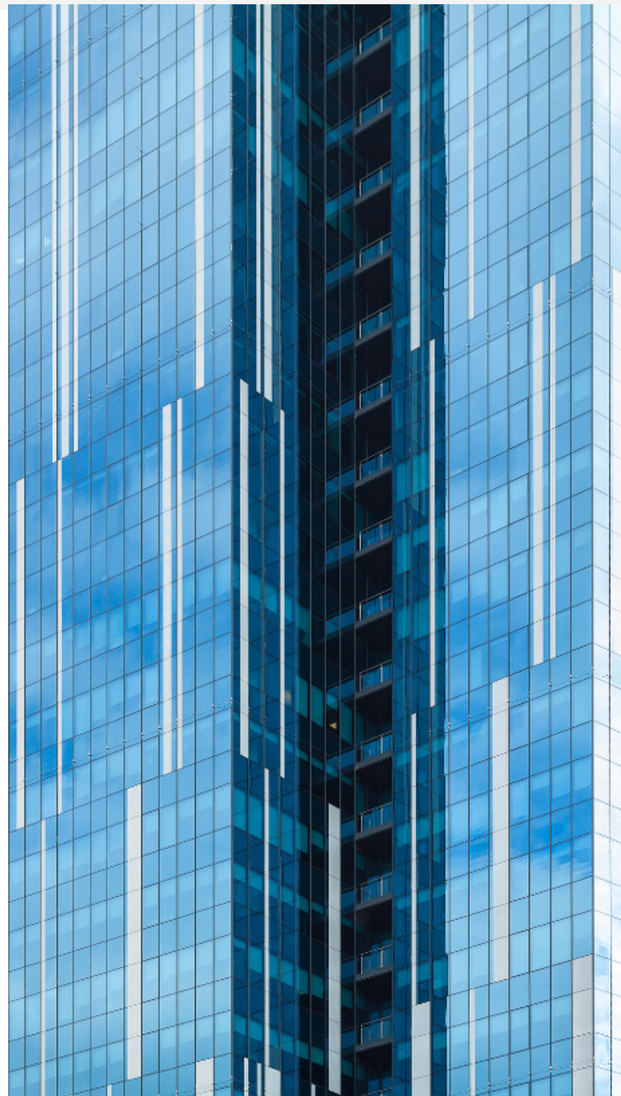
As you might imagine there could be 100's or 1,000's of servers in a datacenter and groups makes them easy to manage

Playbooks

- Ansible stores it’s instructions in files called "Playbooks"
- Expressed in YAML language
- Composed of one or more "plays" in a list
- Allows for multi-machine deployment orchestration

Playbook — Tasks

- Are executed in the order they are specified against all machines that match the host pattern
- May be included from other files
- If a task fails, the remaining playbook are not executed for that host
- Each task executes a module with specific options
- Modules are idempotent in order to bring the system to a desired state



Playbook — Handlers

- Notifications may be triggered at the end of each block of tasks that modify the remote system
- Handlers are referred to by name and can perform operations like restarting services that have had their configuration changed

```
tasks:
- name: Create template configuration file
  template: src=template.j2 dest=/etc/httpd/httpd.conf
  notify:
    - restart apache

handlers:
  - name: restart apache
    service: name=apache state=restarted
```

Example Playbook

python.yaml

```
---
- name: Performing Package Maintenance on All nodes
  hosts: all
  become: yes
  tasks:
  - name: Update and upgrade local packages
    apt: upgrade=full update_cache=yes

- name: Installing Python development environment
  hosts: all
  become: yes
  tasks:
  - name: install required packages for Python using the apt module
    apt: package={{ item }}  update_cache=yes
    with_items:
      - git
      - python-pip
      - python-dev
      - build-essential
```

Example Playbook

python.yaml

```
---
- name: Performing Package Maintenance on All nodes
  hosts: all
  become: yes
  tasks:
  - name: Update and upgrade local packages
    apt: upgrade=full update_cache=yes

- name: Installing Python development environment
  hosts: all
  become: yes
  tasks:
  - name: install required packages for Python using the apt module
    apt: package={{ item }}  update_cache=yes
    with_items:
      - git
      - python-pip
      - python-dev
      - build-essential
```

sudo apt-get update

Example Playbook

python.yaml

```
---
- name: Performing Package Maintenance on All nodes
  hosts: all
  become: yes
  tasks:
  - name: Update and upgrade local packages
    apt: upgrade=full update_cache=yes

- name: Installing Python development environment
  hosts: all
  become: yes
  tasks:
  - name: install required packages for Python using the apt module
    apt: package={{ item }}  update_cache=yes
    with_items:
      - git
      - python-pip
      - python-dev
      - build-essential
```

sudo apt-get update

sudo apt-get install -y git python-pip python-dev build-essentials

Roles

- Based on a know file structure

```
site.yaml
webservers.yaml
roles/
    webservers/
        files/
        templates/
        tasks/
        handlers/
        vars/
        defaults/
```

webservers.yaml

```
---
- hosts: webservers
  roles:
    - webservers
```

Vagrant Integration

- Ansible is supported by Vagrant so that you can use the same Ansible Playbooks to configure your local VM and remote servers

```
#
# Run Ansible using the guest plugin
#
config.vm.provision :guest_ansible do |guest_ansible|
  guest_ansible.playbook = "python.yaml"
  guest_ansible.extra_vars = { user: "vagrant" }
  guest_ansible.sudo = true
end
```

LAMP STACK Example #1

- This will install Apache, MySQL, & PHP on Linux (LAMP)

```
################################################################################
# LAMP Stack Example #1
################################################################################

---
- hosts: webserver
  become: yes
  tasks:
  - name: Install MySQL server
    apt: name=mysql-server state=latest

  - name: Install Apache module for MySQL authentication
    apt: name=libapache2-mod-auth-mysql state=latest

  - name: Install MySQL module for PHP
    apt: name=php5-mysql state=latest
```

LAMP Stack Example #2

- A better way would be to use the iteration of `with_items`

```
############################################################
# LAMP Stack Example #2
############################################################

---
- hosts: webservers
  become: yes
  tasks:
  - name: install packages
    apt: name={{item}} state=latest update_cache=yes
    with_items:
    - mysql-server
    - libapache2-mod-auth-mysql
    - php5-mysql
```

Hands-On

Some Assembly Required

- Tools you will need to complete this lab:
  - Computer running macOS, Linux, or Windows*
  - Vagrant and VirtualBox installed
  - Text Editor (i will use VSCode)
  - GitHub Account

* Windows users may need an ssh client
* PC users must have "VT-x/AMD-V hardware acceleration" turned on in your BIOS for VirtualBox to work

Source for This Lab

- The source for this lab can be found on GitHub at:

```
$ git clone https://github.com/nyu-devops/lab-ansible.git
$ cd lab-ansible
$ vagrant up
```

Vagrant status shows 3 VMs

```
$ vagrant status
Current machine states:

web                       running (virtualbox)
db                        running (virtualbox)
client                    running (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`.
$
```

Vagrant ssh client

```
vagrant ssh client
```

Vagrant ssh client

```
vagrant ssh client
```

Because we have multiple VM's we have to tell Vagrant which VM to ssh into

Run Ansible

- Let’s see if we can ping the servers

```
$ cd /vagrant
$ ansible -m ping all
```

Nope! 😞

```
vagrant@client:/vagrant$ ansible -m ping all
web1 | FAILED! => {
    "changed": false,
    "failed": true,
    "module_stderr": "",
    "module_stdout": "bash: /usr/bin/python: No such file or directory\r\n",
    "msg": "MODULE FAILURE",
    "parsed": false
}
db1 | FAILED! => {
    "changed": false,
    "failed": true,
    "module_stderr": "",
    "module_stdout": "bash: /usr/bin/python: No such file or directory\r\n",
    "msg": "MODULE FAILURE",
    "parsed": false
}
```

Nope! 😞

```
vagrant@client:/vagrant$ ansible -m ping all
web1 | FAILED! => {
    "changed": false,
    "failed": true,
    "module_stderr": "",
    "module_stdout": "bash: /usr/bin/python: No such file or directory\r\n",
    "msg": "MODULE FAILURE",
    "parsed": false
}
db1 | FAILED! => {
    "changed": false,
    "failed": true,
    "module_stderr": "",
    "module_stdout": "bash: /usr/bin/python: No such file or directory\r\n",
    "msg": "MODULE FAILURE",
    "parsed": false
}
```

Looks like the end-points don't have Python 2 and that is required for Ansible to work

Let's Check the Web VM

- Let's ssh to the `web1` VM and see if Python is installed

```
$ ssh web1
```

```
vagrant@web:~$ python --version
The program 'python' can be found in the following packages:
 * python-minimal
 * python3
Ask your administrator to install one of them

vagrant@web:~$ exit
```

Let’s Prepare the Servers

- Ansible’s only pre-requisite is that Python 2 is installed on the end-point
- Luckily, we can use Ansible to bootstrap Ansible!

```
$ ansible-playbook prechecks.yaml
```

Prechecks.yaml

```yaml
---
#####################################################################
# Make sure that all nodes have Python installed
#####################################################################
- name: Ensure connectivity to all nodes
  hosts: all
  gather_facts: false
  become: yes
  pre_tasks:
    - name: Check if python is installed
      raw: test -e /usr/bin/python
      register: python_installed
      changed_when: false
      failed_when: false
    - name: Install python
      raw: (apt-get -y update && apt-get install -y python)
      when: python_installed.rc != 0
  tasks:
    - name: Ansible setup
      action: setup
      tags: ['ping']
    - name: Ensure that aptitude is installed
      apt:
        name: aptitude
        state: present
  any_errors_fatal: true
  max_fail_percentage: 0
```

Prechecks.yaml

```
---
#####################################################################
# Make sure that all nodes have Python installed
#####################################################################
- name: Ensure connectivity t
  hosts: all
  gather_facts: false
  become: yes
  pre_tasks:
    - name: Check if python is installed
      raw: test -e /usr/bin/python
      register: python_installed
      changed_when: false
      failed_when: false
    - name: Install python
      raw: (apt-get -y update && apt-get install -y python)
      when: python_installed.rc != 0
  tasks:
    - name: Ansible setup
      action: setup
      tags: ['ping']
    - name: Ensure that aptitude is installed
      apt:
        name: aptitude
        state: present
  any_errors_fatal: true
  max_fail_percentage: 0
```

Test if Python is installed and save the result in a variable called 'python_installed'

Prechecks.yaml

```
---
#####################################################################
# Make sure that all nodes have Python installed
#####################################################################
- name: Ensure connectivity t
  hosts: all
  gather_facts: false
  become: yes
  pre_tasks:
    - name: Check if python is installed
      raw: test -e /usr/bin/python
      register: python_installed
      changed_when: false
      failed_when: false
    - name: Install python
      raw: (apt-get -y update && apt-get install -y python)
      when: python_installed.rc != 0
  tasks:
    - name: Ansible setup
      action: setup
      tags: ['ping']
    - name: Ensure that aptitude is installed
      apt:
        name: aptitude
        state: present
  any_errors_fatal: true
  max_fail_percentage: 0
```

Test if Python is installed and save the result in a variable called 'python_installed'

If python_installed.rc !=0 install it

Success!!!

😀

```
vagrant@client:/vagrant$ ansible-playbook prechecks.yaml

PLAY [Ensure connectivity to all nodes] ****************************************

TASK [Check if python is installed] ********************************************
ok: [db1]
ok: [web1]

TASK [Install python] **********************************************************
ok: [web1]
ok: [db1]

TASK [Ansible setup] ***********************************************************
ok: [db1]
ok: [web1]

TASK [Ensure that aptitude is installed] ***************************************
changed: [db1]
changed: [web1]

PLAY RECAP *********************************************************************
db1                        : ok=4    changed=1    unreachable=0    failed=0
web1                       : ok=4    changed=1    unreachable=0    failed=0
```

Let’s try ping Again

- Let’s see if we can ping the servers now

```
vagrant@client:/vagrant$ ansible -m ping all
web1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Using Playbooks

- We can list the tasks that will run in a playbook

```
$ ansible-playbook playbook.yaml --list-tasks
```

List of Tasks

```
playbook: playbook.yaml

  play #1 (all): Performing Package Maintenance on All nodesTAGS: []
    tasks:
      Update and upgrade local packagesTAGS: []

  play #2 (webservers): TAGS: []
    tasks:
      Ensure that Apache is at the latest versionTAGS: []
      Start Apache ServicesTAGS: []

  play #3 (dbservers): TAGS: []
    tasks:
      Ensure Redis is installedTAGS: []
      Start Redis TAGS: []

  play #4 (webservers:dbservers): TAGS: []
    tasks:
      Stop UFW NOW!!!TAGS: []
```

Using Dry Run Mode

- Dry run mode will tell us what would have happened

```
$ ansible-playbook playbook.yaml --check
```

What would have run

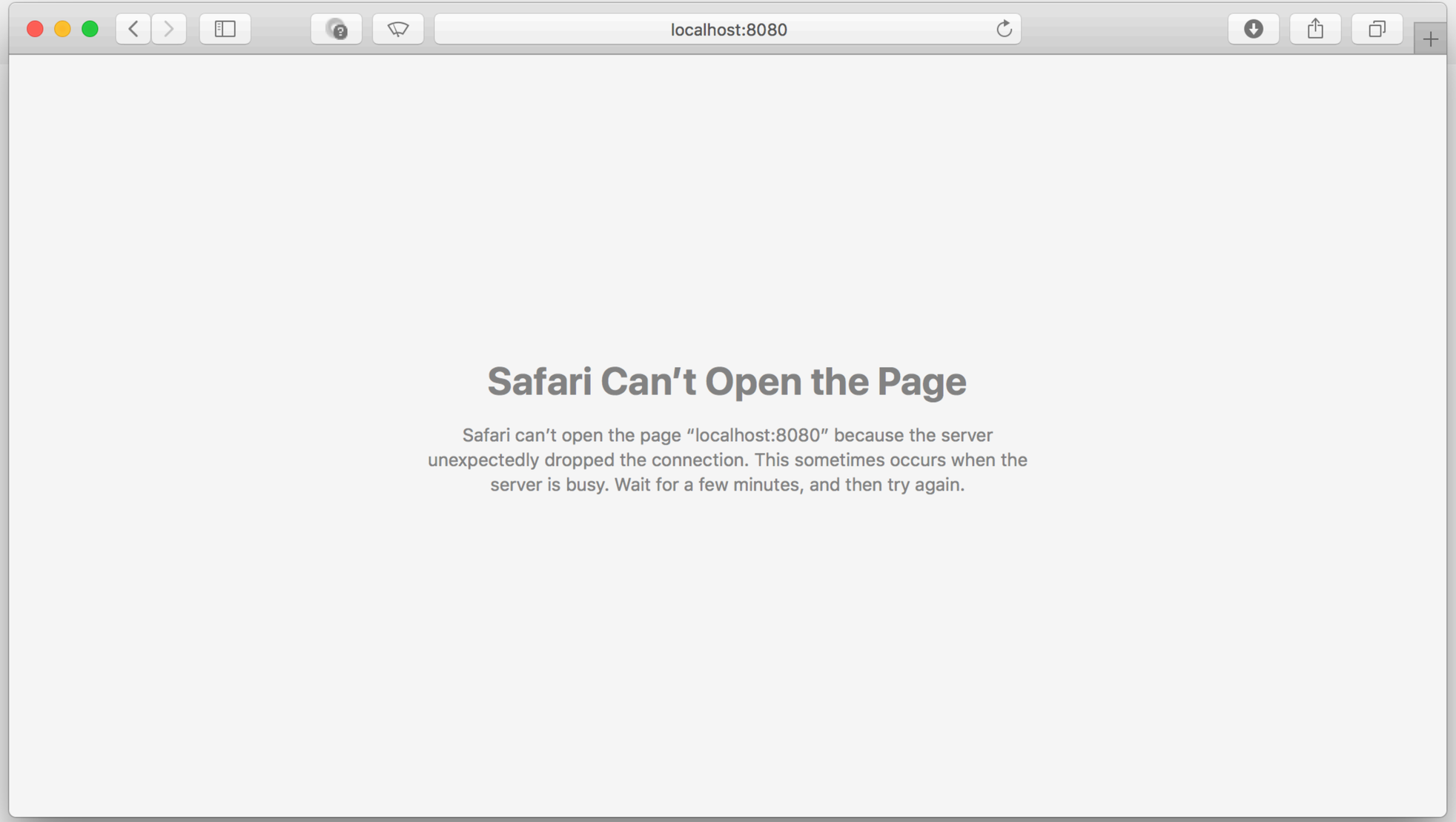
```
vagrant@client:/vagrant$ ansible-playbook playbook.yaml --check

PLAY [Performing Package Maintenance on All nodes]
**********************************************************************************************************

TASK [Gathering Facts]
**********************************************************************************************************************************************
ok: [db1]
ok: [web1]

TASK [Update and upgrade local packages]
**********************************************************************************************************************************
changed: [web1]
changed: [db1]

PLAY [webservers]
**********************************************************************************************************************************************************

TASK [Gathering Facts]
**********************************************************************************************************************************************
ok: [web1]

TASK [Ensure that Apache is at the latest version]
**********************************************************************************************************
changed: [web1]

TASK [Start Apache Services]
**********************************************************************************************************************************************
changed: [web1]

RUNNING HANDLER [restart apache2]
**********************************************************************************************************************************
changed: [web1]

PLAY [dbservers]
**********************************************************************************************************************************************************

TASK [Gathering Facts]
**********************************************************************************************************************************************
ok: [db1]

TASK [Ensure Redis is installed]
**********************************************************************************************************************************
changed: [db1]

TASK [Ensure Redis is listening on all ports]
**********************************************************************************************************************************
fatal: [db1]: FAILED! => {"changed": false, "msg": "Destination /etc/redis/redis.conf does not exist !", "rc": 257}
   to retry, use: --limit @/vagrant/playbook.retry

PLAY RECAP
**********************************************************************************************************************************************************
db1                        : ok=4    changed=2    unreachable=0    failed=1
web1                       : ok=6    changed=4    unreachable=0    failed=0
```

What would have run

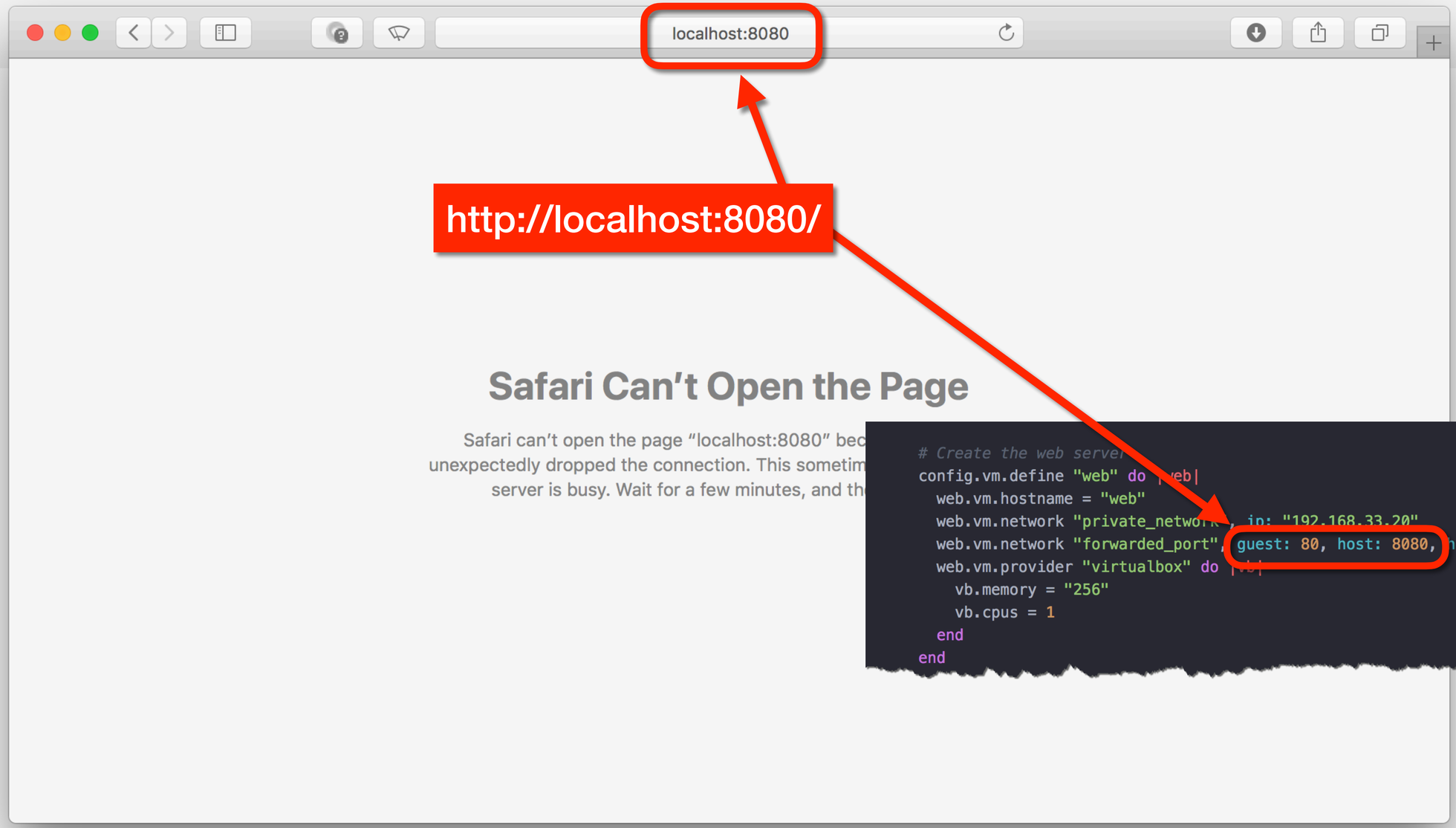
```
vagrant@client:/vagrant$ ansible-playbook playbook.yaml --check

PLAY [Performing Package Maintenance on All nodes]
*********************************************************************************************************

TASK [Gathering Facts]
*********************************************************************************************************
ok: [db1]
ok: [web1]

TASK [Update and upgrade local packages]
*********************************************************************************************************
changed: [web1]
changed: [db1]

PLAY [webservers]
*********************************************************************************************************

TASK [Gathering Facts]
*********************************************************************************************************
ok: [web1]

TASK [Ensure that Apache is at the latest version]
*********************************************************************************************************
changed: [web1]

TASK [Start Apache Services]
*********************************************************************************************************
changed: [web1]

RUNNING HANDLER [restart apache2]
*********************************************************************************************************
changed: [web1]

PLAY [dbservers]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [db1]

TASK [Ensure Redis is installed]
*********************************************************************************************************
changed: [db1]

TASK [Ensure Redis is listening on all ports]
*********************************************************************************************************
fatal: [db1]: FAILED! => {"changed": false, "msg": "Destination /etc/redis/redis.conf does not exist !", "rc": 257}
   to retry, use: --limit @/vagrant/playbook.retry

PLAY RECAP
*********************************************************************************************************
db1                        : ok=4    changed=2    unreachable=0    failed=1
web1                       : ok=6    changed=4    unreachable=0    failed=0
```

This is expected because we really didn't install Redis so its conf file doesn't exist yet

Try and Access the Web Server



Try and Access the Web Server



Running the Playbook

- Finally we will run the playbook to install apache2 and redis

```
$ ansible-playbook playbook.yaml
```

Use verbose mode for more info

```
$ ansible-playbook -v playbook.yaml
```

Performing Package Maint

```
---
#########################################################################
# Perform maintenance on all servers
#########################################################################
- name: Performing Package Maintenance on All nodes
  hosts: all
  become: yes

  tasks:
  - name: Update and upgrade local packages
    apt: upgrade=full update_cache=yes

  # - name: Upgrade distribution packages
  #   apt: upgrade=dist update_cache=yes
```

Web servers

```
##################################################################################
# Web Servers
##################################################################################
- hosts: webservers
  become: yes

  tasks:
  - name: Ensure that Apache is at the latest version
    apt: pkg=apache2 state=latest
    notify:
      - restart apache2

  - name: Start Apache Services
    service: name=apache2 enabled=yes state=started

  handlers:
  - name: restart apache2
    service: name=apache2 state=restarted
```

Database Servers

```
##################################################################################
# Database Servers
##################################################################################
- hosts: dbservers
  become: yes

  tasks:
  - name: Ensure Redis is installed
    apt: pkg=redis-server state=present

  - name: Start Redis
    service: name=redis-server state=started
```

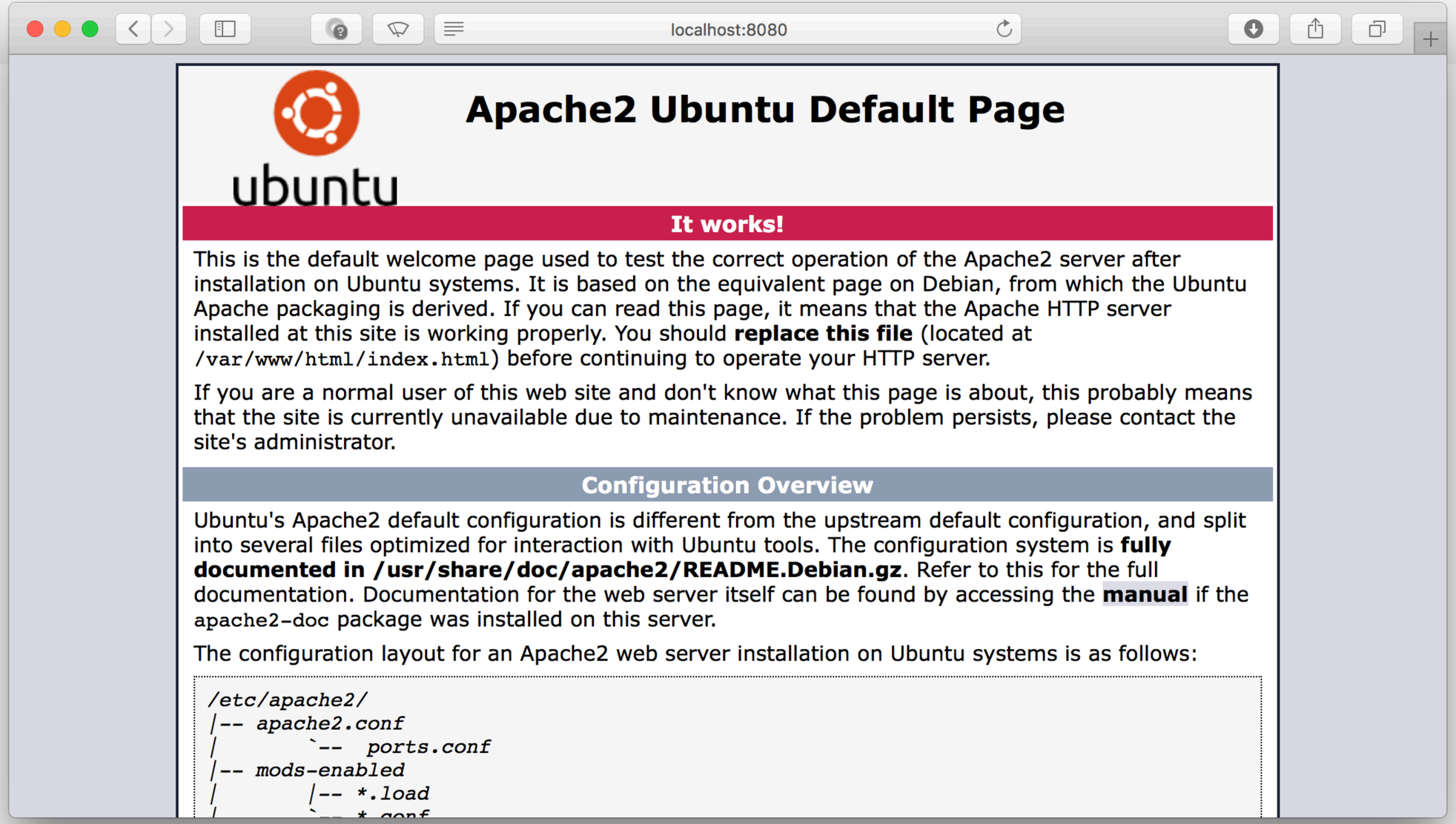
All Servers

```
##################################################################################
# Both web and db Servers
##################################################################################
- hosts: webservers:dbservers
  become: yes

  tasks:
  - name: Stop UFW NOW!!!
    service: name=ufw state=stopped
```

Playbook Output

```
vagrant@client:/vagrant$ ansible-playbook playbook.yaml

PLAY [Performing Package Maintenance on All nodes] ****************************

TASK [setup] *******************************************************************
ok: [web1]
ok: [db1]

TASK [Update and upgrade local packages] ***************************************
changed: [db1]
changed: [web1]

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [web1]

TASK [Ensure that Apache is at the latest version] *****************************
changed: [web1]

TASK [Start Apache Services] ***************************************************
ok: [web1]

RUNNING HANDLER [restart apache2] **********************************************
changed: [web1]

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [db1]

TASK [Ensure Redis is installed] ***********************************************
changed: [db1]

TASK [Start Redis] *************************************************************
ok: [db1]

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [web1]
ok: [db1]

TASK [Stop UFW NOW!!!] *********************************************************
changed: [web1]
changed: [db1]

PLAY RECAP *********************************************************************
db1                        : ok=7    changed=3    unreachable=0    failed=0
web1                       : ok=8    changed=4    unreachable=0    failed=0
```

Now Access the Web Server



Exit

- Let's ssh to `web1` and see if apache is running

```
$ ssh web1
```

```
vagrant@web:~$ ps -aef | grep apache
root      7492     1  0 16:24 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  7495  7492  0 16:24 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  7496  7492  0 16:24 ?        00:00:00 /usr/sbin/apache2 -k start
vagrant   7969  7948  0 16:28 pts/0    00:00:00 grep --color=auto apache

vagrant@web:~$ exit
```

What about db1?

- Let's ssh to `db1` and see if Redis is running

```
$ ssh db1
```

```
vagrant@db:~$ ps -aef | grep redis
redis     6639     1  0 16:24 ?        00:00:00 /usr/bin/redis-server 0.0.0.0:6379
vagrant   6861  6844  0 16:33 pts/0    00:00:00 grep --color=auto redis

vagrant@db:~$ redis-cli ping
PONG

vagrant@db:~$ exit
```

Use Ansible to Provision Python 3

- Let's add Python 3 to the client VM using Ansible
- Uncomment these lines from the `Vagrantfile` (at bottom)

```
    client.vm.provision "ansible" do |ansible|
      ansible.playbook = "python.yaml"
    end
```

- Reload the client and retprovision

```
$ vagrant reload client --provision
```

Playbook Output

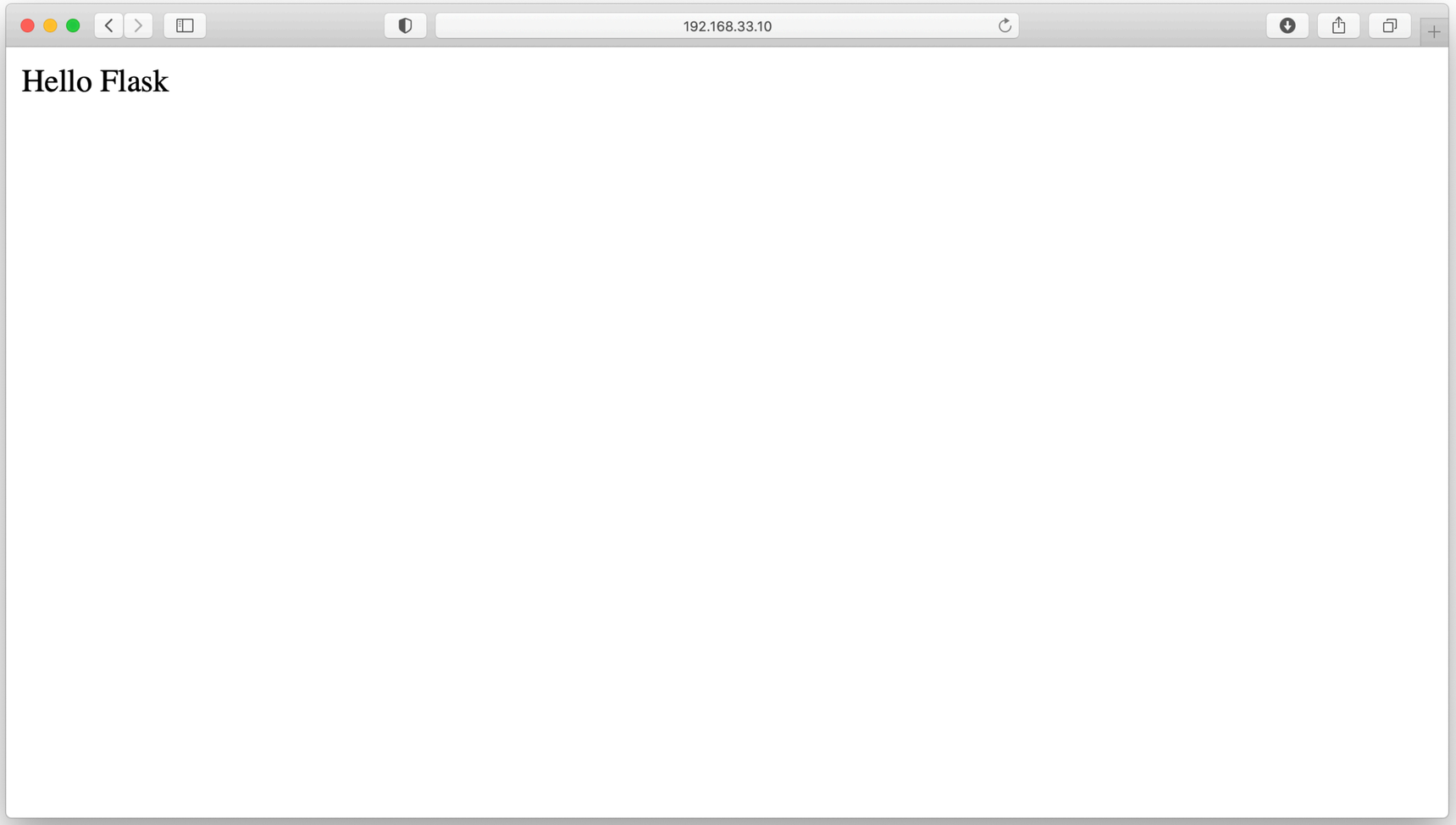
```
PLAY [Performing Package Maintenance on All nodes] ****************************

TASK [Gathering Facts] *********************************************************
[DEPRECATION WARNING]: Distribution Ubuntu 18.04 on host client should use
/usr/bin/python3, but is using /usr/bin/python for backward compatibility with
prior Ansible releases. A future Ansible release will default to using the
discovered platform python for this host. See https://docs.ansible.com/ansible/
2.10/reference_appendices/interpreter_discovery.html for more information. This
 feature will be removed in version 2.12. Deprecation warnings can be disabled
by setting deprecation_warnings=False in ansible.cfg.
ok: [client]

TASK [Update and upgrade local packages] ***************************************
[WARNING]: Updating cache and auto-installing missing dependency: python-apt
changed: [client]

PLAY [Installing Python development environment] *******************************

TASK [Gathering Facts] *********************************************************
ok: [client]

TASK [Install Python packages using the apt module] ****************************
changed: [client]

TASK [Install Python requirements into the specified (virtualenv)] *************
changed: [client]

PLAY RECAP *********************************************************************
client                     : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

Let's check the results

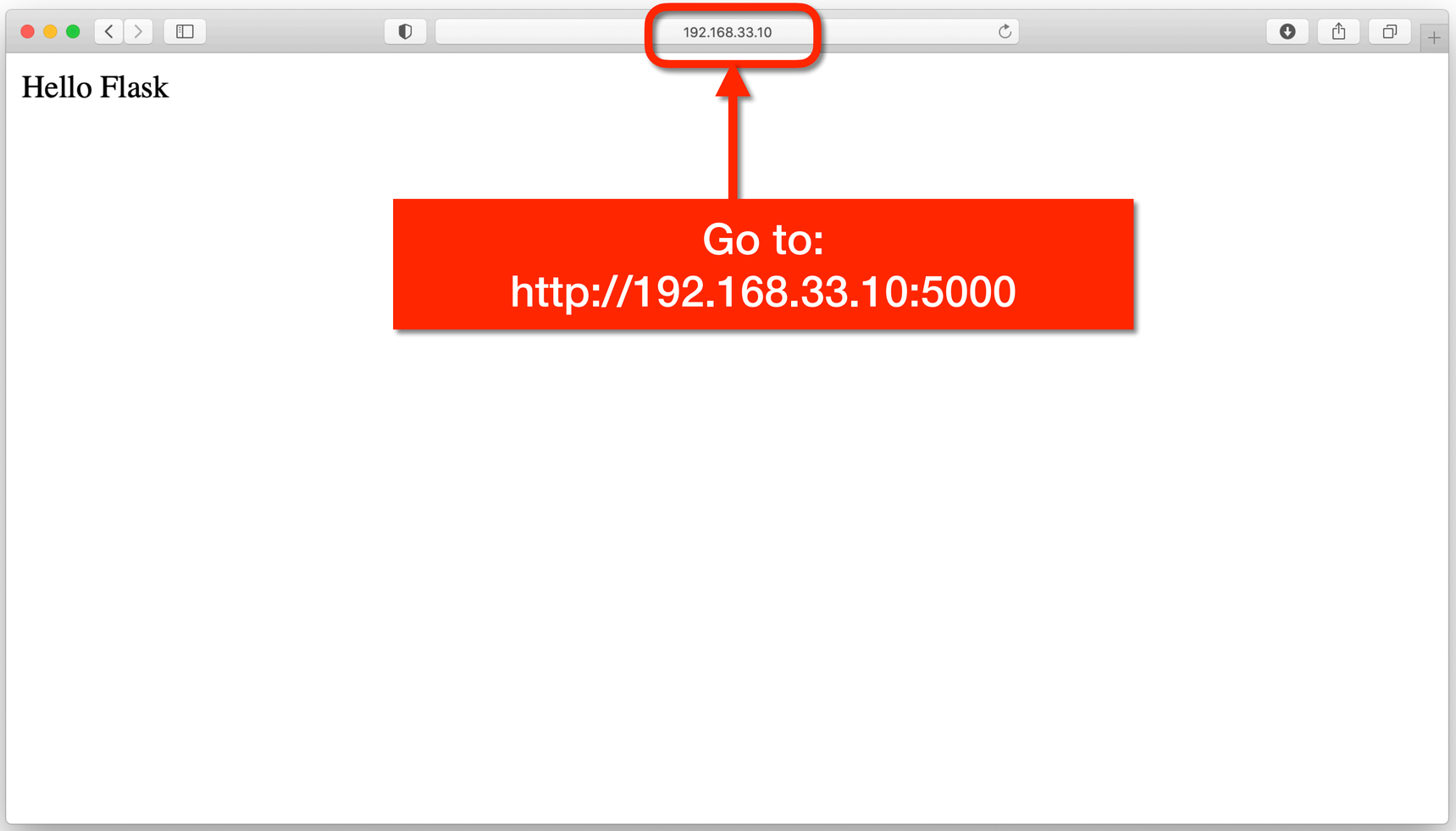
- SSH into the Vagrant VM and run the app

```
$ vagrant ssh client

vagrant@client:~$ source venv/bin/activate
(venv) vagrant@client:~$ cd /vagrant
(venv) vagrant@client:/vagrant$ flask run -h 0.0.0.0
```

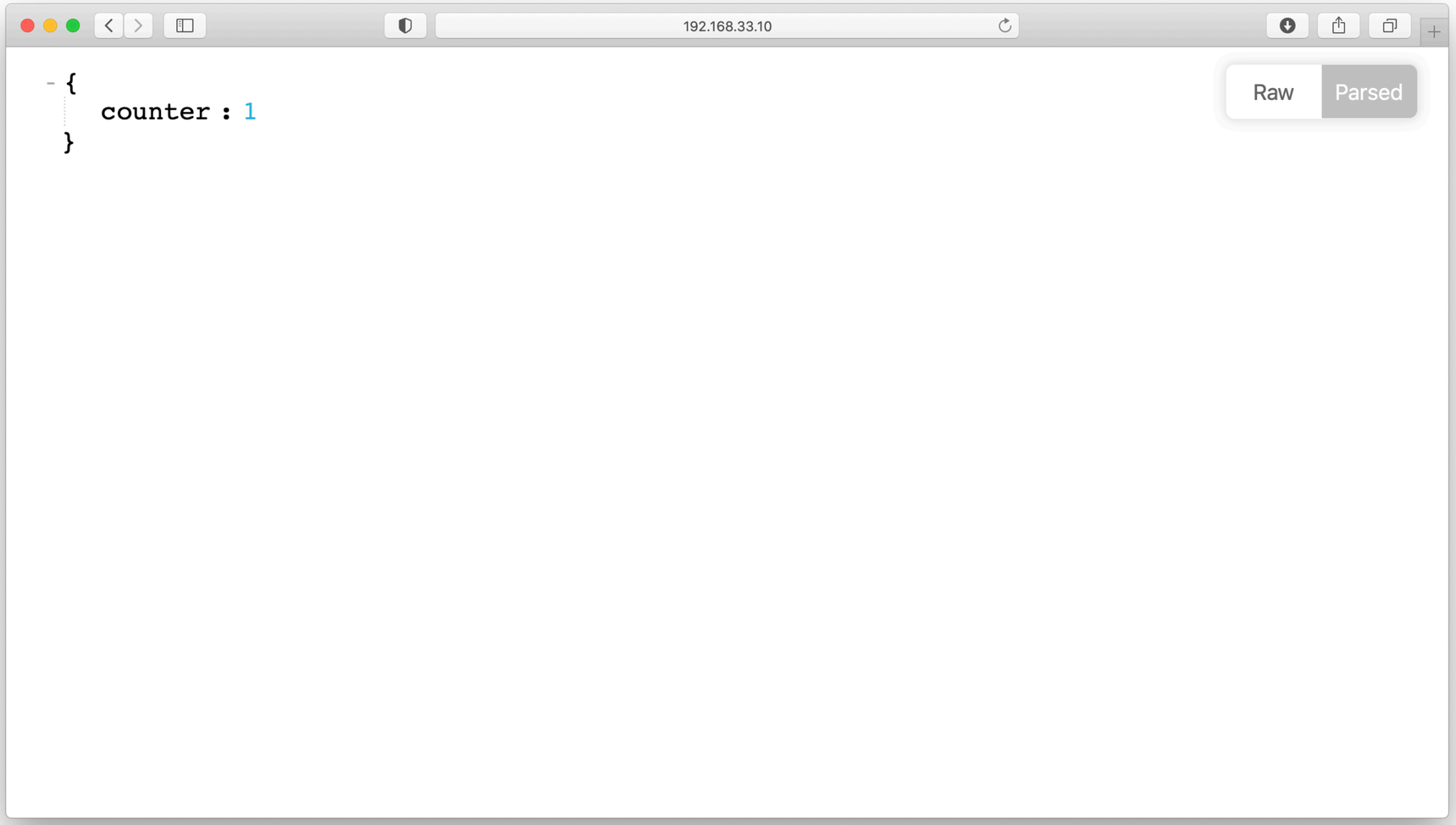
Go to: 192.168.33.10:5000 and see it working



Go to: 192.168.33.10:5000 and see it working



Check the /counter



Check the /counter

Tips for Debugging

- The debugging playbook shows you some debug techniques

```
$ ansible-playbook debugging.yaml
```

Debugging .yaml

```yaml
---
- hosts: webservers
  become: yes
  tasks:
  - name: Show how to register output
    shell: pwd
    register: output_from_pwd
  - name: Show what was saved is JSON
    debug: var=output_from_pwd
  - name: Get return code from JSON output
    debug: msg="Return code was {{ output_from_pwd['rc'] }}"

  - name: Update and upgrade apt packages
    shell: "apt-get update -y && apt-get upgrade -y"
    register: command_output
  - debug: msg="{{ command_output['stdout_lines'] }}"
  - set_fact:
      last_output_line: "{{ command_output['stdout_lines'][-1:][0] }}"
  - name: Echo the last line from command output
    shell: "echo Last line was: {{ last_output_line }}"
```

Debug Output

```
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [web1]

TASK [Show how to register output] *********************************************
changed: [web1]

TASK [Show what was saved is JSON] *********************************************
ok: [web1] => {
    "output_from_pwd": {
        "changed": true,
        "cmd": "pwd",
        "delta": "0:00:00.001683",
        "end": "2018-12-12 16:24:49.923123",
        "rc": 0,
        "start": "2018-12-12 16:24:49.921440",
        "stderr": "",
        "stdout": "/home/vagrant",
        "stdout_lines": [
            "/home/vagrant"
        ],
        "warnings": []
    }
}

TASK [Get return code from JSON output] ****************************************
ok: [web1] => {
    "msg": "Return code was 0"
}

TASK [Update and upgrade apt packages] *****************************************
changed: [web1]
 [WARNING]: Consider using apt-get module rather than running apt-get
```

Vagrant Destroy

- Clean up the lab using `vagrant destroy` and answer `yes` 3 times

```
$ vagrant destroy
    client: Are you sure you want to destroy the 'client' VM? [y/N] y
==> client: Forcing shutdown of VM...
==> client: Destroying VM and associated drives...
    db: Are you sure you want to destroy the 'db' VM? [y/N] y
==> db: Forcing shutdown of VM...
==> db: Destroying VM and associated drives...
    web: Are you sure you want to destroy the 'web' VM? [y/N]
==> web: Forcing shutdown of VM...
==> web: Destroying VM and associated drives...
```

Summary

You should now understand how configuration management systems like Ansible work

You should have a fundamental working of Ansible

You learned how to expose create multiple VM’s from a single Vagrantfile

You can now manage your VM’s with Ansible