Ansible Handson

Convenient Vagrant commands:

vagrant up: brings up all machines defined in the Vagrantfile of the current directory
vagrant ssh <name>: connects to the VM
vagrant reload <name>: forces a reload of the VM, useful to force new configuration to be loaded
vagrant halt <name>: stops the VM
vagrant provision <name>: provisions the VM using the provision configuration in the vagrantfile

A warning beforehand. Read the instructions **carefully**, every step is important.

Setting up:

First off, make sure your Ansible host VM is installed and configured. The VM supplied by Vagrant does not have Ansible installed so you must install it yourself. The provided Vagrantfile will provide you with a basic VM, run it using *vagrant up*. Once it is up and running connect to it (vagrant ssh ansiblehost). The set of commands below will install Ansible and check the version to see if the installation is successful.

sudo rpm -ivh http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-9.noarch.rpm
sudo yum -y install ansible
ansible --version

Congratulations, Ansible is now configured and ready to use. Time to install another VM that we will use to host our Apache webserver. Let's call it apachewebserver. Paste below code into the Vagrantfile, at the bottom, before the last **end**.

```
config.vm.define "apachewebserver" do |apachewebserver|
  apachewebserver.vm.hostname = "apachewebserver"
end
```

Again, type *vagrant up* in your command line, Vagrant will now create a new VM to host your Apache webserver. Notice how Vagrant leaves your ansiblehost alone because it's already up and running. Now we have two VMs but no network connecting them together. This way it will be impossible to have our Ansible machine do anything to our Apache server. Time to set up a network! Paste the following lines into the Vagrantfile, into their respective VM blocks. Make sure to match the indentation! It matters!

ansiblehost.vm.network "private_network", ip: "192.168.33.10"

apachewebserver.vm.network "private_network", ip: "192.168.33.20"

Now, type *vagrant reload* to make sure Vagrant sets up your network. Once both VMs have restarted ssh to one of the VMs and check the ip address with ip addr. The private network should show up! Ping the other server (using the IP address assigned in the Vagrantfile) to check if the

connection works fine. Now it is time to configure the Ansible hosts file. Create a new hosts file /etc/ansible/hosts and paste the following into it (*sudo touch /etc/ansible/hosts && sudo vi /etc/ansible/hosts*):

[apache]
192.168.33.20

N.b., remember that you can write your file inside vi using :w and quit using :q.

Let us inspect what this means. There is a group of hosts called servers, inside this group there is a server 192.168.33.20 which has the name apache. OK! Ansible will now know that if you want to run Ansible commands on the group servers it needs to connect to 192.168.33.20 (just one server at this moment). Time to run our very first Ansible commands to check if it all works fine. Issue the following command on ansiblehost:

ansible servers -a "cat /etc/hosts"

Ouch! That did not go as planned. It appears that the apachewebserver does not allow us to connect using SSH. Since Ansible uses SSH that means we need to set up a key, so that it will accept the connection. On the ansiblehost server run the following command (leave the password empty and save it in the default directory: /home/vagrant/.ssh/id_rsa)

ssh-keygen -t RSA

Now copy the contents of /home/vagrant/.ssh/id_rsa.pub to the clipboard. We will copy the key to our apachewebserver. Log out of the ansiblehost and log in to the apachewebserver. **Make sure you copy the public key, not the private key.** Paste the contents of your key inside ~/.ssh/authorized_keys using vi. Alright, log out and back in to our ansiblehost server; try to ssh 192.168.33.20. It should work fine. If it doesn't check if you pasted the public key into authorized_keys of the apachewebserver, if you pasted the private key it will not work.

Now that we have verified that the SSH connection works, it is time to run your very first Ansible command. This will be an Ansible ad hoc command.

ansible servers -a "cat /etc/hosts"

If all went well you should see

```
[vagrant@ansiblehost ~]$ ansible servers -a "cat /etc/hosts"
192.168.33.20 | SUCCESS | rc=0 >>
127.0.0.1       apachewebserver apachewebserver
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

Congratulations! You managed to run your very first Ansible command.

Your first Playbook: provisioning the apachewebserver

It is time to write your first Playbook. Ansible ad hoc commands are very useful when you need to quickly run some command(s) but when you want to automatically provision or deploy you will need to use a Playbook. A Playbook basically automates a set of ad hoc commands. A Playbook is run against a predefined set of hosts and as one specific user, usually a "deployment" user. In this tutorial we will have Ansible become root during deployments. This tutorial is quite long. Remember that you can check the validity of your Playbook at any time not only through the Ansible Visual Studio Code extension, but also by running *vagrant provision ansiblehost*. Doing so regularly will

alert you to mistakes early on, instead of having to fix error after error after completing the Playbook.

The Playbook must contain a certain set of tasks to be executed. An Ansible task consists of a command that is to be run using a certain Ansible module. For example the following task will ensure that the httpd service is enabled and started. Notice the syntax. First is the (optional) name of the task. Second is the name of the Ansible module, in this case the service module. Third is a dictionary (an array) specifying what the module should ensure. In this case it will ensure that the httpd service (our Apache webservice) is enabled and started. See the Ansible documentation on modules for more information: https://docs.ansible.com/ansible/latest/user_guide/modules.html.

```
- name: Ensure httpd is started
  service:
    name: httpd
    enabled: yes
    state: started
```

Ansible has a lot of modules, after a while it becomes second nature to simply Google for an Ansible module because it's difficult to know them all by heart. Why use Ansible modules though? Ansible can run commands simply by executing shell commands like so:

```
- command: systemctl daemon-reload
```

Ansible modules ensure idempotency. A shell command would simply execute that command, a module will ensure that the server reaches a certain state. Whenever possible you should be using modules in your Playbook, not bare commands.

Now, in your main directory create a file called playbook.yml. This will be your Playbook. When in doubt, check the lecture slides for the exact syntax or ask the instructor! Let the playbook run against all hosts and become root. Now comes the fun part, time to install the Apache webservice. Use the Ansible yum module to install Apache and use the Ansible service module to ensure that it is started after installation. If you get stuck, remember that Google is a useful tool. The Ansible documentation is very good and should help you out. If you are still stuck, ask the instructor for help!

Once you have your Playbook it is time to run it. Remember, the goal is to have the ansiblehost machine execute the Playbook against the Apache webserver. To do so we must edit the Vagrantfile. It is also possible to ssh to the ansiblehost machine and make that run the Playbook but that would require us to place the Playbook inside of the VM. Using the Vagrantfile is much more simple. Add the following to the ansiblehost configuration. Again mind the indentation!

```
ansiblehost.vm.provision :ansible_local do |ansible|
  ansible.playbook      = "playbook.yml"
  ansible.verbose       = false
  ansible.install       = false
  ansible.limit         = "all" # or only "servers" group, etc.
  ansible.inventory_path = "inventory"
end
```

Notice that this needs an inventory file (*ansible.inventory_path = "inventory"*). For now we can simply copy the contents of our /etc/ansible/hosts file into a file called inventory, in the main directory of Vagrant. Remove the file /etc/ansible/hosts, it will no longer be necessary.

Now in your terminal, run the command vagrant provision. This will use Vagrant to provision the ansiblehost. Now because our definition of provisioning the ansiblehost has it run a Playbook, it will actually provision the apachewebserver. When successful, it should show something like below:

```
==> ansiblehost: Running provisioner: ansible_local...
Vagrant has automatically selected the compatibility mode '2.0'
according to the Ansible version installed (2.4.2.0).

Alternatively, the compatibility mode can be specified in your Vagrantfile:
https://www.vagrantup.com/docs/provisioning/ansible_common.html#compatibility_mode
    ansiblehost: Running ansible-playbook...

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

TASK [Gathering Facts] *********************************************************
ok: [192.168.33.20]

TASK [ensure yum installs httpd] ***********************************************
changed: [192.168.33.20]

TASK [Ensure httpd is started] *************************************************
changed: [192.168.33.20]

PLAY RECAP *********************************************************************
192.168.33.20              : ok=3    changed=2    unreachable=0    failed=0
```

Inspect the output. First Ansible gathers facts (it connects to the host and gathers the hostname). Next it shows the tasks that are run; including the custom name we have defined in the Playbook. It shows whether the tasks were successful or not. Notice that Ansible also reports that the task has changed the server. If you were to rerun the Playbook, with Apache already installed, up and running, you will find that Ansible simply checks the server and reports that the server is in "OK" state.

```
TASK [ensure yum installs httpd] ***********************************************
ok: [192.168.33.20]

TASK [Ensure httpd is started] *************************************************
ok: [192.168.33.20]
```

This is the idempotence that makes Ansible excellent. It will not waste time running commands that do not need to be run. Ansible gets a server in a wanted state, and if it already is in that wanted state, it will not touch the server.

We should test whether Apache actually works. It should show a basic page. At the moment port 80 of our Apache webserver cannot be reached; we need to forward a port to open it up. Add the following line to the apachewebserver configuration:

apachewebserver.vm.network :forwarded_port, guest: 80, host: 8080

This will forward port 8080 on the host machine (your PC) to port 80 of the VM. *vagrant reload apachewebserver* to enable the new configuration and visit 127.0.0.1:8080 to see your very own Apache webserver, provisioned by Ansible.

Getting more advanced: adding a simple React application

Now that we have an Apache webserver it's time to put it to good use. We will create and provision a new VM that will host a simple React application. No more hand-holding, this time you will have to figure out the code for yourself. Create a VM that is named reactserver, connected to the private

network with IP 192.168.33.21. It should not be reachable from the outside so no ports need to be forwarded. Don't forget to copy your public key!

Once you have the VM it's time to adapt your Playbook so that will provision the reactserver. This is not as straightforward as it sounds! If you simply add the commands below the Apache webserver part it will install and start Apache on your reactserver as well as on your apachewebserver. This is not the goal! We want to install Apache only on apachewebserver, not on the reactserver. Hint: use Ansible roles! To run our React application we need to install the latest version (!) of NodeJS on our reactserver. You can use the Ansible yum module to install NodeJS, but you first need to add the EPEL repository because it is a prerequisite to install NodeJS. Use the yum_repository module to add the EPEL repository, import the EPEL public key (since this is a fresh machine it won't be there) using the rpm_key module. Download the NodeJS repository install information using this command:

- name: execute the nodejs repo script
shell: "curl --silent --location https://rpm.nodesource.com/setup_10.x | sudo bash -"

Then use the yum module to install NodeJS.

In short:

1) Change your Playbook and directory structure to suit Ansible roles
2) Apache should be one role, NodeJS should be the other role
3) Make sure Apache role is run only on the Apache server
4) Make sure NodeJS role is run only on the React server
5) Make the NodeJS role installs NodeJS using yum

N.b.: the URL for the EPEL repository is https://download.fedoraproject.org/pub/epel/$releasever/$basearch/. The URL for the EPEL public key is http://download.fedoraproject.org/pub/epel/RPM-GPG-KEY-EPEL-7.

Once your React machine has NodeJS it's time to install npx, the NodeJS package we will use to run our React application. Ansible conveniently has an npm module, use that to install npx to /usr/local/lib/node_modules. You can set the install path using the path: option. Now it's time to deploy our react application to the reactserver and start it!

Create a folder files in the nodejs role folder. Copy the react_application folder to the files folder. Now we can have the nodejs role copy the react_application folder to the reactserver. Use the copy module. Mind that Ansible will automatically look inside the files/ folder of the nodejs/ folder to find files when copying, so no need to explicitly mention the path. The copy module will also copy the entire directory recursively so there is no need to worry about the individual files. Mind that a trailing / does matter for the copy module. Copy folder/ will copy the **contents** of the folder to the destination. Copy folder (no trailing slash) will copy **the folder and the contents** to the destination.

Next up we want to, of course, run the React app as a service to make sure it will be started whenever we start our VM. Create a file files/etc/systemd/system/react_application.service. Fill it with:

[Unit]
Description=React application

[Service]
ExecStart=/usr/bin/npx serve -s /apps/react_application

Restart=on-failure
User=root
Group=root
Environment=PATH=/usr/bin:/usr/local/bin
WorkingDirectory=/apps/react_application

[Install]
WantedBy=multi-user.target

Now make sure this file is copied to /etc/systemd/system/react_application.service. The service needs to be recognised and started as well. Use the systemd module to daemon-reload, and then use the systemd module to start the react_application service and make sure it is enabled as well. When a service is enabled it will start automatically when the VM starts.

We are almost there but not quite yet… The React application is listening on port 5000, but how will we connect to it? Port 5000 is closed and worse yet, we can never reach port 5000 from outside the network. Time to set up our Apache webserver to be a proxy!

First things first, we want to open up port 5000 in the firewall to traffic coming from other VMs inside the network. Ansible conveniently has a firewalld module. Use it to permanently open port 5000 to tcp traffic. Use the service module to make sure the firewall is restarted or else port 5000 won't be opened until you restart the VM!

Next up: set up Apache as a proxy. Inside the apache role directory, create a files/ folder, and inside of that etc/httpd/conf.d/reverseproxy.conf. Paste the following into the reverseproxy.conf:

<Location "/react_application">
ProxyPass http://192.168.33.21:5000/
ProxyPassReverse http://192.168.33.21:5000/
</Location>

You know the drill; make sure that Ansible copies that file to /etc/httpd/conf.d/reverseproxy.conf using the copy module. Also make sure that the httpd service is restarted afterwards, or it won't pick up on the change!

Run the Playbook again using vagrant provision ansiblehost. If all is well you should now have a fully working webproxy and React application running. You can connect to it by going to http://127.0.0.1:8080/react_application.

Congratulations and I hope you learned a lot about Ansible! These are the very, very basics. There is a lot more and you can definitely optimize the structure you have now by using variables for example.