Follow Along

Set VirtualBox Network settings for these three VMs:

- **pfSense_Reference:**
  - Adapter 1: NAT
  - Adapted 2: Internal Network: CS312LAN
- **CentOS_CLI_Reference:**
  - Adapter 1: Internal Network: CS312LAN
- **Alpine_Reference:**
  - Adapter 1: Internal Network: CS312LAN

1. Start the pfSense_Reference VM
2. Wait until router VM is up
3. Start the CentOS_CLI_Reference VM
4. Start the Alpine_Reference VM

Ansible

Benjamin Brewster & Elijah Voigt

Why You Need to Care

- Gaining experience with a Configuration Management tool will open your eyes to what automation can do for you
- You'll never again want to configure a program or computer by hand
    - Sorry about that
- Ansible is amazing, and you gotta see this
- Your employers want to know that you can deploy all the things to all the things at once - if they don't know they want it, they will after you get done telling them about it

Ansible Reminder

- Ansible operates on a "push" model: whenever you want, you can execute commands on the hosts
- This is the mechanism, straight from the documentation:
  "Ansible works by connecting to your 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 then executes these modules (over SSH by default), and removes them when finished."
- I.e., it establishes an SSH connection, pushes over the modules it needs to arrive at the state you've told it to assume, then runs those commands on the pushed module; finally, removes the modules when done (which is good: they can't be used after the fact)

Basic Ansible Elements

- A **node** is a PC/computer/host
- A **hosts** file specifies authentication information for relevant nodes
  - Similar to, but not to be confused with, the "/etc/hosts" or "C:\Windows\System32\drivers\etc\hosts" file which provides a local lists of DNS mappings
- A **playbook** is a YAML file that specifies which nodes should be manipulated, and what should be done to them
- A **module** is a unit of code that does a particular thing, like "ping", which pings nodes

Prepare Virtual Machines

Say

- Reset our VMs to their base configuration

Demonstrate

- Restore the pfSense_Reference VM to base
- Restore the CentOS_CLI_Reference VM to base
- Restore the Alpine_Reference VM to base

- Check to make sure the networking is correct:
  - pfSense_Reference:
    - Adapter 1: NAT
    - Adapted 2: Internal Network: CS312LAN
  - CentOS_CLI_Reference:
    - Adapter 1: Internal Network: CS312LAN
  - Alpine_Reference:
    - Adapter 1: Internal Network: CS312LAN

- Start the pfSense_Reference VM
- Start the CentOS_CLI_Reference VM
- Start the Alpine_Reference VM

Prepare Controller and Node

Say	**Demonstrate**
• Reset our CentOS CLI to its base configuration	• On the CentOS VM, log in: u: centosuser, p: password • On the Alpine VM, log in: u: root, p: password
• On our Alpine node VM: Get the IP address of the Alpine VM	• On the Alpine VM: `$ ip addr`
• On our CentOS controller VM: Lets test the network state (use the node IP)	• On the CentOS VM: `$ ping 192.168.1.XXX`
• Now on the controller VM, let's install ansible! That's a lot of Python goo that gets installed!	• On the CentOS VM: `$ sudo yum install -y ansible`

Set up RSA SSH Keys

SAY

- Ansible works with RSA SSH keys, and we now need to set some up
  - It can use passwords, too, but they have to be saved in plain text on your system. Ick!
- Generate two things: our private RSA key, and a public shareable one
- Copy our public RSA key from our controller to our node
- FYI: Ideally, you'd:
  - `# chmod 600 ~/.ssh`

DEMONSTRATE

- On our controller:
- `$ ssh-keygen -t rsa`
  - Hit enter three times to accept default storage location, and no passphrase associated with this key
- `$ ssh-copy-id -i ~/ssh/id_rsa root@192.168.1.XXX`

Test SSH Connection

Say	**Demonstrate**
• Let's test our new SSH key setup	• `$ ssh root@192.168.1.XXX`
• This should just log in with no password needed	
• Remember that the Alpine VM has had a toggle thrown that allows the root account to SSH in (this is usually disabled)	
• Log back off the node VM from our controller VM	• `# exit`

Ping All the Things/Nodes!

Say

- Build a hosts file called hosts.ini
- Examine the hosts.ini file
  - All nodes are specified under "mynodes" (just one right now)
  - Our connection is ssh, using port 22, with user "root", and we want to use the python interpreter located at the given path
- Fire up Ansible using the "mynodes" group, in the hosts.ini file (-i), and ask them to use the module "ping" (-m)
  - More on modules coming up
- There should be a response back in red, showing that ping failed because Python wasn't installed!

Demonstrate

- `$ curl web.engr.oregonstate.edu/~brewsteb/CS312/ansible/hosts.ini > ~/hosts.ini`
- `$ cat hosts.ini`
- `$ ansible mynodes -i ./hosts.ini -m ping`

Installing Python and THEN Pinging

SAY

- First we'll install Python
  - `mynodes -i ./hosts.ini` :: run this on all nodes found in this file
  - `-b` :: "become", as in "become root"
  - `-m raw` :: use the raw module, which really means don't use a module, run this raw CLI command
  - `-a` :: Specify the arguments for the module, which in this case is the raw CLI command that follows

DEMONSTRATE

```
$ ansible mynodes -i ./hosts.ini -b -m raw -a "apk -U add python3"
```

SAY

- Now we can ping!

DEMONSTRATE

```
$ ansible mynodes -i ./hosts.ini -m ping
```

Ansible Playbooks

- Ansible **playbooks** are YAML files which contain instructions (tasks) to be performed on a set of machines
- These can include installing a program, copying a template file, starting a service, and much more

- Instead of manually typing in a command:

  -
    ```
    $ ansible mynodes -i ./hosts.ini -m ping
    ```
- Tell Ansible to run a particular (lengthy) playbook on a set of nodes:

  -
    ```
    $ ansible-playbook webserver.yaml -i hosts.ini
    ```

Modules & Playbooks

- The **modules** listed in the playbooks perform a complete task: they are complicated, able to handle all kinds of contingencies and conditions, and are configurable

- Sample modules might install WordPress, set up Nagios monitoring, configure and run tools on Amazon's Web Services (AWS) Lambda functions or Elastic Cloud compute service, or even send Slack notifications

- Playbooks can specify different combinations of nodes and modules to make your management be modular

Interlude: Playbooks use "YAML" Files

```yaml
---                                              # Begins the file
key: value                                       # A simple mapping
multi-word key: "value has lots too"             # multi-word is OK, quotes optional
a nested mapping:                                # Begin a nested collection
  Bill: Bob                                      # Bill is a part of the parent collection
  Georgio: Moroder                               # So is Moroder; two-space indents
my array:                                        # Begin an array
- item1                                          # Arrays use the hyphen to define items
- item2: can be a map                            # Items can be maps
- An array item can also hold an array           # Another array item
-                                                # Begins a nested array, nothing else here
  - of more key: value pairs                     # Two more spaces in
  - This array item: value pair                  # Two key-pairs in this one array item
    Has more than one: key and value pair
```

Our YAML Playbook

```
---                                                        # Starts the document
- hosts: mynodes                                           # Which nodes to run this playbook on
  gather_facts: no                                         # Do not gather data, which could fail without Python
  become: true                                             # Means "become root when running this"
  tasks:                                                   # Interpreted as tasks Ansible will do
  - name: Ensure Python is at the latest version           # A text label for a task
    raw: apk -U add python3                                # Run a raw command: ensure Python3 installed
  - name: Ensure apache is at the latest version           # A text label for a task
    apk:                                                   # The apk task installs packages...
      name: apache2                                        # Specifically the apache package
      state: latest                                        # Use the latest version
  - name: Copy our index.html into place                   # A text label for another task
    template:                                              # Copies a file over, replaces vars IN that file
      src: ansible-index.html                              # The page we're copying and replacing vars in
      dest: /var/www/localhost/htdocs/index.html           # Where we're copying the page to
    notify:                                                # Run the following "handlers" after this task...
    - restart apache                                       # The handler to run is called "restart apache"
  - name: ensure apache is running                         # A text label for another task
    service:                                               # Ensure a specific service is in a state...
      name: apache2                                        # The service in question is Apache
      state: started                                       # Ensure that it is running ("started")
  handlers:                                                # Here is where we define handlers
    - name: restart apache                                 # The "restart apache" handler...
      service:                                             # Manages a systemd daemon...
        name: apache2                                      # The apache daemon, specifically
        state: restarted                                   # Set it to the "restarted" state
```

Ansible Variables

- The "template" module copies a file into place AND replaces any variables in that file with values you define

- In our case, we'll simply allow this module to replace any instance of:

```
{{ template_run_date }}
```

... found in the copied file with a timestamp that shows the last time we ran a playbook that updated the variable

Get the Playbook and Webpage

Say

- Grab our playbook and a basic index.html web page for deployment
- Display the file contents

Demonstrate

- `$ curl web.engr.oregonstate.edu/~brewsteb/CS312/ansible/webserver.yaml > ~/webserver.yaml`
- `$ curl web.engr.oregonstate.edu/~brewsteb/CS312/ansible/ansible-index.html > ~/ansible-index.html`
- `$ cat ~/webserver.yaml`
- `$ cat ~/ansible-index.html`

Use Our Playbook

Say

- The current state of the new Alpine VM is that it has our controllers SSH key, enabling us to automatically log in
- The Alpine node VM does not have:
  - Python
  - Any web server
  - Any web files
- Let's now completely provision our node VM!
- Test it from the controller!
- Check out that web page! With the timestamp!

Demonstrate

- `$ ansible-playbook webserver.yaml -i ./hosts.ini`
- `$ curl 192.168.1.XXX`

Replicate Twice!

Say	**Demonstrate**
• Time to make this work across 2 VMs	
	• Shut down the Alpine node VM
	• Restore it back to its default state
	• Change it's networking to be Internal Network: CS312LAN
	• Start the Alpine VM up
• Copy our public RSA key from our controller to our node	• `$ ssh-copy-id -i ~/ssh/id_rsa root@192.168.1.XXX`
• Test the pushed key by trying to login	• `$ ssh root@192.168.1.XXX`
• Halt the VM while we're in there	• `# halt`
• Now, duplicate this Alpine VM that has our SSH key twice, all with the correct network settings	• Do this twice to create VMs YYY and ZZZ: Right-click on the Alpine node VM and click Clone, use settings: Rename it, Full Clone, Current machine state, Re-init MAC address

Provision All!

Say

- Start them all up!
- Gather data
- Prep our hosts file to target everyone
- Capture the *other* 2 public SSH keys and add them to our `known_hosts` file (*not* `authorized_hosts`) so that we don't have to type "yes" a bunch of times
- Fire up Ansible to provision all three VMs at once as web servers!
- Get the Homepages of each one, note the differences in timestamps

Demonstrate

- Start up all three Alpine node VMs (ensure they start!)
- Record the IP addresses of all three
- Add the IP addresses to hosts.ini on the Controller:
  - `$ vi ~/hosts.ini`
- `$ ssh-keyscan 192.168.1.YYY 192.168.1.ZZZ >> ~/.ssh/known_hosts`
- `$ ansible-playbook webserver.yaml -i hosts.ini`
- `$ curl 192.168.1.XXX`
- `$ curl 192.168.1.XXX`
- `$ curl 192.168.1.XXX`

Integrating Configuration Management

- The goal is something called "Continuous Deployment" - you could do this a dozen times an hour:

1. Write code on local development VMs (which themselves could be built by Ansible to contain everything you need)
2. Have a Continuous Integration system like Jenkins automatically deploy to a staging environment on every code change committed. This deploy job calls testing scripts to pass/fail the just-deployed build wherever it's deployed to
3. If the deploy job succeeds, it runs the same deploy playbook against production inventory

Conclusion

- Just think of the possibilities: Ansible could configure all of your servers and computers and routers at once, to any config you need
- If a Docker controller spun these up as a hundred containers, Ansible could set them ALL up as web servers!