ANSIBLE FUNDAMENTALS

Instructor: Rick Copeland rick@arborian.com

Class Repository: https://github.com/Arborian/ansible-class

Class Times: 09:00 - 05:00 / Lunch 12:00-1:00

- Interactive demo / lecture
- Hands-on, with frequent labs
- In addition to lab times, breaks every 60 ~ 90 minutes

GETTING TO KNOW YOU

- Name
- Programming / operations experience? (e.g. Chef, Puppet, Salt, Fabric?)
- Job role
- Specific thing you'd like to learn

OUTLINE

- Introduction to Ansible and DevOps
- Setting up learning environment
- Inventory your infrastructure
- Ad Hoc Server Management
- Survey of useful ansible modules

OUTLINE (CONT'D)

- Ansible Playbooks
- Templating with Jinja2
- Roles for modularizing code
- Orchestration with cloud modules
- Building your own custom modules

WHAT IS DEVOPS?

DevOps (a clipped compound of "development" and "operations") is a software engineering culture and practice that aims at unifying software development (Dev) and software operation (Ops).

The main characteristic of the DevOps movement is to strongly advocate automation and monitoring at all steps of software construction, from integration, testing, releasing to deployment and infrastructure management.

> *DevOps aims at shorter development cycles, increased deployment frequency, more dependable releases, in close alignment with business objectives.*

— Wikipedia

BUT REALLY?

- Development and operations are no longer separate organizations
  - developers deploy code
  - operators develop code

- Operations uses software development principles and tools to manage infrastructure
    - revision control (i.e, git)
    - configuration management (i.e. Ansible/Puppet/Chef)

Executed well, your infrastructure has a *repeatable* and *revertable* configuration

(Some even disable ssh for production machines...)

WHAT IS IT NOT?

- An excuse to give the workload of two people to just one
- An excuse to deploy sloppy code to production

We want to bring the best of both worlds together, not the worst.

WHY AND HOW OF ANSIBLE

- Configuration management tool (a-la Puppet, Chef)
- Ad-hoc management tool (some overlap with Fabric, Capistrano)
- Python library / tool
- "Push" model (generally via ssh)

ANSIBLE: GETTING STARTED

- Managed hosts need minimal "bootstrapping"
  - Puppet, Chef, Salt, etc. all need an agent on the managed host.
  - Ansible only needs sshd (Python is helpful, as well, but we can bootstrap that)

LEARNING ENVIRONMENT

First, you'll also need to clone the repository at https://github.com/arborian/ansible-class:

```
$ git clone https://github.com/Arborian/ansible-class.git
```

In this repository, you'll find all the examples, as well as these slides as HTML, Markdown, and PDF.

You should already have installed Ansible. If you have not, you can follow the instructions here.

LEARNING ENVIRONMENT

Once you've cloned the repo, you'll need to edit the file aws_ec2.yaml to include the `aws_access_key` and `aws_secret_key` given in class (or use a new `aws_profile`).

Once you've done that, verify your ansible installation by navingating to the repository and executing the following command:

```
$ ansible -m ping all
ansible | SUCCESS => {
...
```

ANSIBLE CONFIGURATION FILE

Ansible looks for a configuration file when it runs, located by default at /etc/ansible.cfg or in the local directory. The repository contains a configuration file that matches our environment.

ansible.cfg

```
[defaults]
inventory = ./aws_ec2.yaml,./inventory.yaml
remote_user = centos
private_key_file = ./keys/class-keypair.pem
host_key_checking = False
library = ./modules
roles_path = ./roles
vault_password_file = vault-password.txt
```

ANSIBLE INVENTORY

Ansible needs to know which hosts it manages. For this, it uses an *inventory*.

In our initial command, the inventory was specified in the ansible.cfg file, but we can also specify additional sources with the `-i` command-line argument.

STATIC INVENTORY

If you're just getting started with Ansible, it can be helpful to use a **static** inventory, which is just a text file in one of several file formats:

- ini
- toml
- yaml

STATIC INVENTORY (INI / TOML)

Ansible actually uses **plugins** to load the inventory sources. If you have a static text file that looks like this, Ansible will use the ini plugin:

```
[web]
arborian-01.class.arborian.com
arborian-02.class.arborian.com
...
```

STATIC INVENTORY (YAML)

Static inventory can also be specified in a YAML file, which *can* make some aspects of configuration easier:

```
all:
    hosts:
        arborian-01.class.arborian.com:
        arborian-02.class.arborian.com:
        arborian-03.class.arborian.com:
        arborian-04.class.arborian.com:
        arborian-05.class.arborian.com:
        arborian-06.class.arborian.com:
    children:
        web:
            hosts:
                arborian-01.class.arborian.com:
                arborian-02.class.arborian.com:
        ...
```

EC2 DYNAMIC INVENTORY

We can also dynamically discover inventory using plugins. For this class, we'll use the Amazon dynamic inventory plugin. It is configured using the aws_ec2.yaml file in the current directory:

```
plugin: aws_ec2
aws_access_key: AKIA...
aws_secret_key: Soi1...
regions:
  - us-east-1
filters:
  tag:Cluster: AnsibleFundamentals
hostnames:
- tag:Name
compose:
  ansible_host: public_ip_address
```

TESTING INVENTORY

To verify that your inventory is working properly, use the `ansible-inventory --list` command:

```
$ ansible-inventory --list

... lots of JSON output ...
```

HOSTS AND GROUPS

Each machine we manage is a *host*. One or more machines can be included in a *group*.

You can see your groups using the `ansible-inventory --graph` command:

```
$ ansible-inventory --graph
@all:
  |--@Role_app:
  |  |--server-4
  |  |--server-5
  |--@Role_db:
  |  |--server-2
...
```

LAB: CREATE A GROUP OF ONE FOR YOUR OWN SERVER

- Edit the `inventory.yaml` file to add a new group `"me"` that contains just your server. In my case, I added the following lines:

```
me:
  hosts:
    rick:
```

- Verify that the group creation worked by executing `ansible me -m ping`. Only your host should respond.

AD-HOC COMMANDS

Ansible command line

```
ansible <host pattern> [options]
```

Frequently used options

- `-m MODULE_NAME` (default is command)
- `-a MODULE_ARGS`
- `--become` ("become" root, i.e. `sudo`)

Frequently used host patterns

- Specific hostname
- `localhost`
- `all`
- `GROUP-NAME` (one of the groupings from your inventory)

EXAMPLE

Run `whoami` on all the servers, with and without `--become`

```
$ ansible -a whoami all
...
$ ansible -a whoami --become all
...
```

PATTERNS

In simple situations, our pattern will be either a host or a group name. Ansible, however, gives us quite a bit of flexibility in the patterns we use.

For instance, you can:

- combine hosts and groups `group:group2`
- use "globbing" expressions `web-*`
- exclude hosts `web-*:!web-01`
- use regular expressions `~(web|db).*\.example\.com`
- ...and much more

YAML INVENTORY

We can use YAML as a syntax to define our inventory, even combining it with our dynamic inventory to define **variables** for each group:

HOST AND GROUP VARIABLES

Variables can be used in `playbooks` and `roles` (which we'll get to shortly).

We can add the variables in a separate yaml file:

inventory.yaml

```
Role_app:
  vars:
    name: app-server
Role_db:
  vars:
    name: db-server
Role_web:
  vars:
    name: web-server
    port: 80
```

CHECKING THE VALUE OF A VARIABLE

To check the value of a variable, we can use Ansible's debug module:

CHECKING THE VALUE OF A VARIABLE

To check the value of a variable, we can use Ansible's debug module:

```
rick@ansible:~$ ansible localhost -m debug -a var=ansible_host
localhost | SUCCESS => {
    "ansible_host": "127.0.0.1"
}
```

LAB: VARIABLES AND PATTERNS

- Add a custom variable for the 'Role_app' group and view it using the `debug` module (`ansible Role_app -m debug -a var=VARNAME`)
- Target all the servers using `ansible all` and view the value of your new variable (`ansible all -m debug -a var=VARNAME`)

LARGER INFRASTRUCTURE VARIABLE ORGANIZATIONS

In larger infrastructures, we may want to split our variables out of the inventory and into separate files. In this case, we can create a `host_vars` directory and a `group_vars` directory.

OPTION 1: CREATE A SINGLE FILE PER HOST/GROUP

If you place a file (using YAML syntax) into the host_vars or group_vars directory, then that file defines the variables for that host/group.

host_vars/somehost.yaml

```
var1: value1
var2: [1,2,3]
```

group_vars/somegroup.yaml

```
var3: groupvar3
```

OPTION 2: USE SUBDIRECTORIES FOR HOSTS/GROUPS

If you create a subdirectory under host_vars or group_vars, then *all files* in that subdirectory are used to define the host/group variables:

host_vars/somehost/vars1.yaml

```
var1: value1
```

host_vars/somehost/vars2.yaml

```
var2: value2
```

LAB: MORE VARIABLES

- Create a `host_vars` directory with a file web-0.yaml in it.
  - Set some variables in the web-0.yaml file and verify that they appear with `ansible web-0 -m debug...`
- Create a `group_vars` directory with a subdirectory `all`
  - Create a few files inside `group_vars/all` and verify that the variables are applied to all hosts using `ansible all -m debug ...`

AD-HOC SERVER MANAGEMENT

Sometimes we might want to just run a simple one-off command on a server (or a group). For that, we can use the *command* module:

```
$ ansible me -m command -a 'whoami'
rick | CHANGED | rc=0 >>
centos
```

AD-HOC SERVER MANAGEMENT

Sometimes we might want to just run a simple one-off command on a server (or a group). For that, we can use the *command* module:

```
$ ansible me -m command -a 'whoami'
rick | CHANGED | rc=0 >>
centos
```

For the special case of the command, module, we can leave off the `-m command` as well:

AD-HOC SERVER MANAGEMENT

Sometimes we might want to just run a simple one-off command on a server (or a group). For that, we can use the *command* module:

```
$ ansible me -m command -a 'whoami'
rick | CHANGED | rc=0 >>
centos
```

For the special case of the command, module, we can leave off the `-m command` as well:

```
$ ansible me -a 'whoami'
rick | CHANGED | rc=0 >>
centos
```

TOUR OF COMMON MODULES

ANSIBLE MODULE: `ping`

Ping is handy to ensure that our host is reachable and that it will work with other Ansible commands:

```
$ ansible all -m ping
app-0 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

ANSIBLE MODULE: command

We can execute commands on the remote system with the command module:

```
$ ansible all -m command -a hostname
web-0 | CHANGED | rc=0 >>
ip-172-30-1-73.ec2.internal
db-1 | CHANGED | rc=0 >>
ip-172-30-1-6.ec2.internal
db-0 | CHANGED | rc=0 >>
ip-172-30-1-191.ec2.internal
...
```

ANSIBLE MODULE: `raw`

Host of the modules will require that we have a version of Python already installed on the target system(s). If you need to "bootstrap" a system, you can use the `raw` module:

```
$ ansible me --become -m raw -a "yum install -y python3"
... (lots of output) ...
Dependency Installed:
  python3-libs.x86_64 0:3.6.8-13.el7         python3-pip.noarch 0:9.0.3-7.el7_7
  python3-setuptools.noarch 0:39.2.0-10.el7

Complete!
Shared connection to 3.92.73.102 closed.
```

ANSIBLE MODULE: `shell`

Generally, it's safer to use the `command` module, but if you want to use the shell (or use a particular shell), you can use the `shell` module:

```
$ ansible me -m shell -a 'echo $(hostname) $USER'
rick | CHANGED | rc=0 >>
ip-172-30-1-102.ec2.internal centos
```

ANSIBLE MODULE: `file`

The `file` module is used to create/modify file permissions, directories, and symlinks. It does *not* change the contents of files.

```
$ ansible me -m file -a "name=test state=touch"
rick | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "dest": "test",
    "gid": 1000,
    "group": "centos",
    "mode": "0664",
    "owner": "centos",
    "secontext": "unconfined_u:object_r:user_home_t:s0",
    "size": 0,
    "state": "file",
    "uid": 1000
}
```

```
$ ansible me -m file -a "name=test1 state=directory"
rick | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "gid": 1000,
    "group": "centos",
    "mode": "0775",
    "owner": "centos",
    "path": "test1",
    "secontext": "unconfined_u:object_r:user_home_t:s0",
    "size": 6,
    "state": "directory",
    "uid": 1000
}
```

ANSIBLE MODULE: copy

The copy module can be used to copy the contents of a (local) file to the remote server:

```
$ ansible me -m copy -a 'src=ansible.cfg dest=/tmp/ansible.cfg'
rick | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "checksum": "3b76e742785708cab79b896fed79e258791d05ed",
    "dest": "/tmp/ansible.cfg",
    "gid": 1000,
    "group": "centos",
    "md5sum": "85d93a12fd4faf71d9ae3198164d727d",
    "mode": "0664",
    "owner": "centos",
    "secontext": "unconfined_u:object_r:user_home_t:s0",
    "size": 162,
    "src":
"/home/centos/.ansible/tmp/ansible-tmp-1600382304.8640385-23060-128780978348980/source"
    "state": "file",
    "uid": 1000
}
```

ANSIBLE MODULE: `template`

Most of the time, we may want to customize the contents of a file. For this, we will use [Jinja2][jinja2] templating:

`templates/host_info.txt.j2`:

```
The host is {{ansible_host}}

The port is {{port}}
```

```
$ ansible web-0 -m template -a 'src=templates/host_info.txt.j2 dest=host_info.txt'
web-0 | CHANGED => {
    "ansible_facts": {
...
$ ansible web-0 -a 'cat host_info.txt'
web-0 | CHANGED | rc=0 >>
The host is 54.144.11.23

The port is 80
```

ANSIBLE PACKAGING MODULES: `apt`, `yum`, ETC.

```
$ ansible me --become -m yum -a name=git
rick | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
...lots and lots of output...
```

ANSIBLE MODULES: `user` AND `authorized_key`

The `user` module can be used to create a user on the remote host.

```
$ ansible --become me -m user -a 'name=rick shell=/bin/bash'
rick | CHANGED => {
    ...
    "home": "/home/rick",
    "name": "rick",
    "shell": "/bin/bash",
    ...
}
```

ANSIBLE MODULES: `user` AND `authorized_key`

The `authorized_key` module can be used to add a (public) ssh key to the user's .ssh directory.

```
$ ansible --become me -m authorized_key \
    -a 'user=rick key={{lookup("file", "keys/class-keypair.pem.pub")}}'
rick | CHANGED => {
    ...
    "state": "present",
    "user": "rick",
    ...
}
```

ANSIBLE MODULE: `git`

We can use the `git` module to check out a repository (or a particular version) on the host:

```
$ ansible me -m git -a 'repo=https://github.com/ansible/ansible.git dest=ansible'
rick | CHANGED => {
    "after": "e6e98407178556c1eb60101abef1df08c753d31d",
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "before": null,
    "changed": true
}
```

ANSIBLE MODULE: `service`

You can use the `service` module to start/stop system services. Let's start by installing nginx:

```
$ ansible me --become -m yum -a 'name=epel-release'
rick | CHANGED => {
...
```

```
$ ansible me --become -m yum -a 'name=nginx'
rick | CHANGED => {
...
```

Nginx won't start on its own, (try it) so we start it using the service module

```
$ ansible rick --become -m service -a 'name=nginx state=started'
rick | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
...
```

And now you can visit the server

LAB: AD-HOC COMMANDS

Using Ansible one-off commands:

- Create a user for yourself on "your" server
- Verify that you can ssh to your new account
- Check out the class repository to your user's account (https://github.com/Arborian/ansible-class.git)
- Install and start nginx in your host
- Verify that it is working by visiting your server with a web browser

PLAYBOOKS

SIMPLE PLAYBOOKS

Playbooks are files that specify **plays** which consist of one or more **tasks** to run on various hosts.

Playbooks are written as YAML files.

A simple playbook with one play:

`first-play.yaml`:

```yaml
- name: First play
  hosts: me
  become: true
  tasks:
  - name: Say my name
    command: whoami
    notify: "demo notify"
  - name: Get the system date and time
    command: date
    notify: "demo notify"
  - name: Ping just for fun
    ping:
  handlers:
  - name: demo notify
    command: 'echo Running the demo notify'
```

RUNNING THE PLAYBOOK

```
$ ansible-playbook playbooks/first-play.yaml

PLAY [First play] ***************************************************************

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

TASK [Say my name] **************************************************************
changed: [rick]

TASK [Get the system date and time] **********************************************
changed: [rick]

TASK [Ping just for fun] ********************************************************
ok: [rick]

RUNNING HANDLER [demo notify] ***************************************************
changed: [rick]

PLAY RECAP **********************************************************************
rick                       : ok=5    changed=3    unreachable=0    failed=0    skipped=
```

RUNNING THE PLAYBOOK (VERBOSE)

```
$ ansible-playbook playbooks/first-play.yaml -v
Using /home/rick446/src/arborian-classes/data/ansible-examples/ansible.cfg as config fi

PLAY [First play] ***************************************************************

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

TASK [Say my name] **************************************************************
changed: [rick] => {"changed": true, "cmd": ["whoami"], "delta": "0:00:00.004822", "enc

TASK [Get the system date and time] *********************************************
changed: [rick] => {"changed": true, "cmd": ["date"], "delta": "0:00:00.002940", "end":

TASK [Ping just for fun] ********************************************************
ok: [rick] => {"changed": false, "ping": "pong"}

RUNNING HANDLER [demo notify] ***************************************************
changed: [rick] => {"changed": true, "cmd": ["echo", "Running", "the", "demo", "notify"

PLAY RECAP **********************************************************************
```

PLAYBOOK RUN ORDER

- For each play
  - For each host specified in the play
    - Each task of the play is executed, in order
      - If a task result is 'changed' and the task has a 'notify', then the notify handler is registered
    - Any notify handlers that are registered will run

PLAYBOOK VARIABLES

We can use **variables** in our playbooks to customize its operation.

After reading a playbook's YAML, Jinja2 expressions using variables are expanded:

```
{{var_name}}
```

This playbook uses a `username` variable:

`setup-user.yaml`

```
- hosts: me
  become: yes
  vars:
    username: ansible-is-so-awesome
    mygroups:
        - wheel
  tasks:
  - name: create user
    user:
        name: '{{username}}'
        groups: '{{mygroups}}'
        generate_ssh_key: yes
        shell: /bin/bash
```

We can also prompt the user running Ansible for values of various variables using `vars_prompt`:

`setup-user2.yaml`

```
- hosts: me
  become: yes
  vars_prompt:
    - name: username
      prompt: What is your username?
  vars:
    mygroups:
        - wheel
  tasks:
  - name: create user
    user:
        name: '{{username}}'
        groups: '{{mygroups}}'
        generate_ssh_key: yes
        shell: /bin/bash
```

This will prompt us for a username **if** no variable has been defined elsewhere called `username`:

```
$ ansible-playbook playbooks/setup-user2.yaml
What is your username?:
```

```
$ ansible-playbook playbooks/setup-user2.yaml -e username=ansible-is-awesome

[no prompt]

PLAY [me] ***********************************************************************

TASK [Gathering Facts] **********************************************************
...
```

LAB: PLAYBOOK INTRODUCTION

- Create a playbook that will
  - Create your user on your host, making sure your user is in group `wheel`
  - Authorize the class public key
  - Allow any user in the `wheel` group to execute `sudo` without a password
    - Hint: Use the `template` or `copy` module to create a file in /etc/sudoers.d/
- Run your playbook and ensure that you can ssh into your host and `sudo` without a password

ANSIBLE MODULE: `setup`

Sometimes we need access to various facts about a host (IP addresses, OS configuration, etc.). For that, we can use the `setup` module:

```
$ ansible me -m setup
... lots of output ...
        "ansible_userspace_architecture": "x86_64",
        "ansible_userspace_bits": "64",
        "ansible_virtualization_role": "guest",
        "ansible_virtualization_type": "kvm",
        "discovered_interpreter_python": "/usr/bin/python",
        "gather_subset": [
            "all"
        ],
        "module_setup": true
    },
    "changed": false
}
```

BOOTSTRAPPING / SKIPPING THE 'GATHERING FACTS' STAGE

By default, playbooks will use the `setup` module to gather "facts" (variables about the hosts) before running any tasks. To skip that, we can specify `gather_facts: off` in the play.

To bootstrap an installation that is missing Python, then, we might do the following:

```
- hosts: all
  gather_facts: off
  become: on
  tasks:
  - name: Boostrap Python
    raw: yum install -y python3
```

PLAYBOOK ORGANIZATION

Ansible encourages reuse and code organization by letting you `import_tasks` into a playbook. You can also set variables when you do so, as in `make-roster.yaml`:

```
- hosts: me
  tasks:
    - name: Ensure file exists
      file: path=~/roster.txt state=touch
    - name: Include rick
      import_tasks: roster-user.yaml
      vars:
        name: "Rick Copeland"
        email: "rick@arborian.com"
        regexp: rick
```

tasks/roster-user.yaml:

```
- name: Look at variables
  debug: var=name
- name: Add user to roster
  lineinfile: line="{{name}} {{email}}" dest=~/roster.txt regexp={{regexp}}
```

LOOPING IN PLAYBOOKS

We can also use variables and loops when doing our includes via the `loop` key, as in make-roster2.yaml, but in this case we must use `include_tasks` (instead of `import_tasks`):

```
- hosts: me
  vars:
    users:
        - name: Rick Copeland
          email: rick@arborian.com
          regexp: rick
        - name: Marc Benioff
          email: marc@salesforce.com
          regexp: marc
        - name: Patrick
          email: pat@vmware.com
          regexp: pat
```

```yaml
tasks:
  - name: Ensure file exists
    file: path=~/roster.txt state=file
  - name: Perform some dynamic includes
    loop: '{{roster}}'
    include_tasks: tasks/roster-user.yaml
    vars:
      name: "{{item.name}}"
      email: "{{item.email}}"
      regexp: "{{item.regexp}}"
```

LAB: PLAYBOOK LOOPS

Add a new variable for your server "roster" which contains several users (you can use the inventory.yaml file for this)

Create a playbook that will add several users to the roster.

PLAY SECTIONS

A play has several different keys which may be present:

- `hosts` - list the hosts on which this play should run
- `tasks` - list of tasks to run
- `handlers` - list of handlers to run *if notified by a task*
- `gather_facts` - allows you to disable running `setup` on each host at the beginning of a play
- `become` - if True, become root during the play

- `become_user` - the user to become (default `root`)
- `remote_user` - override the default remote_user
- `vars` - variables which can be used later via Jinja
- `vars_files` - files to look in for extra variables
- `vars_prompt` - variables whose values can be interactively prompted for

TASK MODIFIERS

- `when` - only execute the task when the condition is true
- `loop` - repeat the task with each of the items listed
- `register` - the result of this task gets put in a variable to be used in a later task

BEST PRACTICES FOR ORGANIZING PLAYBOOKS, VARS, AND ROLES

http://docs.ansible.com/ansible/latest/playbooks_best_practices.html

https://docs.ansible.com/ansible/2.9/user_guide/playbooks_best_practices.

TEMPLATING USING JINJA2

Jinja2 is a third-party package that can be used to generate text (or HTML) from templates specified in a python-like templating language.

Jinja2 is used in the `template` module, as well as being available in our playbooks.

USING FACTS IN OUR TEMPLATES

We can get a lot of information to be used in playbooks from *Facts* collected by Ansible. To get a feel for what's available, you can use the `setup` module:

```
rick@ansible:~/ansible-class$ ansible me -m setup
... lots of output ...
        "ansible_virtualization_type": "kvm",
        "gather_subset": [
            "all"
        ],
        "module_setup": true
    },
    "changed": false
}
```

Sometimes it's more useful to filter the output of `setup`:

```
rick@ansible:~/ansible-class$ ansible me -m setup -a filter=ansible_interfaces
rick-instance | SUCCESS => {
    "ansible_facts": {
        "ansible_interfaces": [
            "ens5",
            "lo"
        ]
    },
    "changed": false
}
```

Once we have these facts, we can use them in subsequent tasks (`setup` is always run at the beginning of the play).

`dump-facts.yaml`:

```
- hosts: me
  tasks:
  - name: Create a file using some facts
    template: src=templates/dump-facts.txt.j2 dest=~/dump-facts.txt
```

templates/dump-facts.txt.j2:

```
ansible_cmdline:
{% for k, v in ansible_cmdline.items() %}
    {{ k }}: {{ v }}
{% endfor %}


ansible_default_ipv4:
{% for k, v in ansible_default_ipv4.items() %}
    {{ k }}: {{ v }}
{% endfor %}
```

Running the above playbook generates the following file:

```
rick@ansible:~/ansible-class$ ansible-playbook playbooks/dump-facts.yaml
...
$ ansible me -m command -a 'cat dump-facts.txt'
rick | CHANGED | rc=0 >>
ansible_cmdline:
{'LANG': 'en_US.UTF-8', 'console': 'ttyS0,115200', 'crashkernel': 'auto', 'BOOT_IMAGE':

ansible_default_ipv4:
    macaddress: 0e:02:db:fb:bc:03
    network: 172.30.1.0
    mtu: 9001
    broadcast: 172.30.1.255
    alias: ens5
    netmask: 255.255.255.0
    address: 172.30.1.102
    interface: ens5
    type: ether
    gateway: 172.30.1.1
```

TEMPLATE SYNTAX

Jinja mostly ignores the text you give it and passes it unmodified when rendering. Certain escape sequences, however, trigger Jinja to do something different:;

- `{{...}}` - Expressions whose value is inserted into the output
- `{%...%}` - Statements for things like looping, inheritance, inclusion, conditional blocks, etc.
- `{#...#}` - Comments which will not be included in the output

JINJA FILTERS AVAILABLE WITH ANSIBLE

Filters are used to format data in Jinja. Ansible provides a long list of filters for use in playbooks and templates.

For instance, to get the 'name' variable from all hosts in the inventory, we can use `json_query`:

```
ansible me -m debug -a 'msg={{hostvars | json_query("*.name")}}'
rick | SUCCESS => {
    "msg": [
        "web-server",
        "app-server",
        "db-server",
        "app-server",
        "web-server",
```

USING `debug` WITH THE `msg` PARAMETER TO CHECK SIMPLE EXPRESSIONS

One useful technique when working with templates and filters is to use the `msg` (instead of the `var`) parameter with the `debug` module:

```
$ ansible me -m debug -a \
    msg="{{hostvars | json_query('*.roster[*].email')}}"
rick | SUCCESS => {
    "msg": [
        [
            "rick@arborian.com",
            "marc@salesforce.com",
            "pat@vmware.com"
        ]
    ]
}
```

DEBUGGING PLAYBOOKS

What do you do when everything doesn't "just work?"

TRY WITH A DRY RUN

- Detect Jinja errors
- Detect missing/invalid arguments for tasks

```
$ ansible-playbook --check playbooks/dump-facts.yaml

PLAY [me]
*********************************************************************************

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

TASK [Create a file using some facts]
**************************************************************
ok: [rick]

PLAY RECAP
*********************************************************************************
rick                       : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

RUN IN VERBOSE MODE / VIEW OUTPUT

- View output of tasks
- View commands (if commands are run)

```
$ ansible-playbook  playbooks/dump-facts.yaml -vv
ansible-playbook 2.9.13
  config file = /home/rick446/src/ansible-class/ansible.cfg
  configured module search path = ['/home/rick446/src/ansible-class/modules']
  ansible python module location = /home/rick446/.virtualenvs/classes/lib/python3.8/sit
  executable location = /home/rick446/.virtualenvs/classes/bin/ansible-playbook
  python version = 3.8.2 (default, Jul 16 2020, 14:00:26) [GCC 9.3.0]
Using /home/rick446/src/ansible-class/ansible.cfg as config file

PLAYBOOK: dump-facts.yaml ******************************************************
1 plays in playbooks/dump-facts.yaml

PLAY [me] **********************************************************************

TASK [Gathering Facts] *********************************************************
task path: /home/rick446/src/ansible-class/playbooks/dump-facts.yaml:1
ok: [rick]
META: ran handlers

TASK [Create a file using some facts] ******************************************
task path: /home/rick446/src/ansible-class/playbooks/dump-facts.yaml:3
```

USE THE PLAYBOOK DEBUGGER

Use `debugger: on_failed` in your play or task to drop into a debugger on errors:

```
$ ansible-playbook  playbooks/debug-demo.yaml

PLAY [me]
*********************************************************************************

TASK [Create a file using some facts]
*********************************************************************
fatal: [rick]: FAILED! => {"changed": false, "msg": "AnsibleUndefinedVariable:
'ansible_cmdline' is undefined"}
[rick] TASK: Create a file using some facts (debug)> help

Documented commands (type help <topic>):
========================================
EOF  c  continue  h  help  p  pprint  q  quit  r  redo  u  update_task

[rick] TASK: Create a file using some facts (debug)> p task.args
{'dest': '~/dump-facts.txt', 'src': 'templates/dump-facts.txt.j2'}
[rick] TASK: Create a file using some facts (debug)></topic>
```

SPAM DEBUG STATEMENTS

vault-demo2.yaml

```
- hosts: localhost
  gather_facts: false
  vars:
    encrypted_value: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          61336566303364356534646362393735323263626436313037653764316136343264623234636
          64626639643931343739623964316265306239373431353 80a313165363865323464303338613
          37386331326564303432386632306333383331656634336536316266623239366263643262613
          66323135346334373 10a376230666665366662383738396238623338303233616237353561653
          31326635653639306163653364363565616236353362383639623636564303330 62
  tasks:
    - name: Show the encrypted_value
      debug: var=encrypted_value
```

```
$ ansible-playbook  --ask-vault-pass playbooks/vault-demo2.yaml
Vault password:

PLAY [localhost]
*********************************************************************************

TASK [Show the encrypted_value]
*********************************************************************
ok: [localhost] => {
    "encrypted_value": "This is an seekrit message!\n"
}

PLAY RECAP
*********************************************************************************
localhost                  : ok=1    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

SINGLE-STEPPING

```
$ ansible-playbook  --step playbooks/make-roster.yaml

PLAY [me]
*******************************************************************************
Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: n

Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue:
*****************************************
Perform task: TASK: Ensure file exists (N)o/(y)es/(c)ontinue: n

Perform task: TASK: Ensure file exists (N)o/(y)es/(c)ontinue:
*************************************
Perform task: TASK: Look at variables (N)o/(y)es/(c)ontinue: n

Perform task: TASK: Look at variables (N)o/(y)es/(c)ontinue:
*****************************************
Perform task: TASK: Add user to roster (N)o/(y)es/(c)ontinue: n

Perform task: TASK: Add user to roster (N)o/(y)es/(c)ontinue:
*************************************

PLAY RECAP
```

PLAYBOOK ERROR HANDLING

What happens when the remote host has an error?

IGNORING FAILED COMMANDS

By default, Ansible will not continue to execute commands on that host.

To change that behavior (as when you may run a command that you *expect* to have a nonzero return code), set `ignore_errors` to `yes`

```
- name: this will not be counted as a failure
  command: /bin/false
  ignore_errors: yes
```

RESETTING UNREACHABLE HOSTS

If Ansible cannot reach a host (it is inaccessible to ssh) for a task, it is removed from the list of active hosts.

To re-enable it (and any other hosts that are being suppressed), you can include

```
meta: clear_host_errors
```

to re-activate all failed hosts.

HANDLER BEHAVIOR ON ERROR

If a task fails in a play, any handlers which were notified will, by default, *not* run. To change this behavior, you can:

- Use the `--force-handlers` command-line option
- Include `force_handlers: true` in a play
- Include `force_handlers = True` in `ansible.cfg`

DEFINING WHAT 'FAILED' MEANS

Normally, Ansible uses the result code of a command to define failure. Sometimes you want something else (like checking for some string in a command output).

You can configure this using the `failed_when` flag on a task.

Some examples from http://docs.ansible.com/ansible/latest/playbooks_error_handling.html:

```
- name: Fail task when the command error output prints FAILED
  command: /usr/bin/example-command -x -y -z
  register: command_result
  failed_when: "'FAILED' in command_result.stderr"
- name: Fail task when both files are identical
  raw: diff foo/file1 bar/file2
  register: diff_cmd
  failed_when: diff_cmd.rc == 0 or diff_cmd.rc >= 2
```

DEFINING WHAT 'CHANGED' MEANS

Ansible uses the `changed` flag to detect whether to run handlers. To customize whether a task 'changes' anything, you can use `changed_when:`

```
- shell: wall 'beep'
  changed_when: False
```

ABORTING A WHOLE PLAY

Normally, Ansible tries to complete a play for as many hosts as possible. You may instead have Ansible mark *all* hosts failed by including the `any_errors_fatal: true` flag in a play.

LAB: ERROR HANDLING

- Update your playbook from the previous lab to cause an error whenever your own name is added to the roster
- Run the playbook in `--check` mode to ensure it works
- Run the playbook with `debugger: on_failed` to ensure you get the debugger

ROLES

A *Role* is a directory structure organizing different parts of plays in a modular, reusable way. Roles include the following parts:

- tasks: any tasks to be included in the play
- handlers: any handlers to be included in the play
- files: files to be used with modules such as `copy`
- templates: template files to be used with the `template` module

- vars and defaults: variables to be added to the play
- meta: metadata about the role

If you have a role defined, it can be used in a play using the `roles:` key, e.g.:

```
- hosts: all
  roles:
    - my_custom_role
```

means to look in the `my_custom_role` directory and include various `main.yaml` files in the play.

ROLE RULES

The particular rules are as follows:

- If `my_custom_role/tasks/main.yml` exists, it is added to the `tasks:` part of the play
- If `my_custom_role/handlers/main.yml` exists, it is added to the `handlers:` part of the play

- If `my_custom_role/vars/main.yml` exists, it is added to the `vars:` part of the play
- If `my_custom_role/defaults/main.yml` exists, it is added to the `vars:` part of the play (as super-low priority)
- If `my_custom_role/meta/main.yml` exists, it is added to the `roles:` part of the play (think "sub-roles")

USING ROLES

In order to use roles, you need to set up a `roles_path` in `ansible.cfg` (yours should already be set up):

```
[defaults]
...
roles_path = ./roles
```

INCLUDING A ROLE

To activate a role for a play, just add it to the `roles` section of the play:

`role-test.yaml`

```
- hosts: me
  gather_facts: false
  roles:
    - rick446.testrole
```

```
$ ansible-playbook playbooks/role-test.yaml

PLAY [me] ***********************************************************************

TASK [rick446.testrole : debug] *************************************************
ok: [rick] => {
    "msg": "You included the test role!!!"
}

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

ANSIBLE GALAXY

Ansible Galaxy has a number of roles available for public use.

We can download/install a role into our roles path using the `ansible-galaxy` command-line tool:

```
$ ansible-galaxy install nginxinc.nginx
- downloading role 'nginx', owned by nginxinc
- downloading role from
  https://github.com/nginxinc/ansible-role-nginx/archive/0.16.0.tar.gz
- extracting nginxinc.nginx to
  /home/rick446/src/arborian-classes/data/ansible-examples/ansible-class-repo/roles/ngi
- nginxinc.nginx (0.16.0) was installed successfully
```

EXAMINING THE NGINX ROLE

The most important files to look at are generally `tasks/main.yml`, `defaults/main.yml`, and `vars/main.yml`:

`tasks/main.yml`

```
- name: "(Setup: All OSs) Setup Prerequisites"
  include_tasks: "{{ role_path }}/tasks/prerequisites/setup-{{ ansible_os_family | lowe
  tags: nginx_prerequisites

- name: "(Setup: All OSs) Setup Keys"
  import_tasks: keys/setup-keys.yml
  when:
    - ansible_os_family == "Alpine"
      or ansible_os_family == "Debian"
      or ansible_os_family == "RedHat"
      or ansible_os_family == "Suse"
    - nginx_install_from == "nginx_repository"
      or nginx_amplify_enable | bool
      or nginx_unit_enable | bool
  tags: nginx_key
...
```

EXAMINING THE NGINX ROLE

vars/main.yml

[empty]

defaults/main.yml

```
# Enable NGINX options -- `nginx_install` and `nginx_configure`.
# Default is true.
nginx_enable: true

# Install NGINX and NGINX modules.
# Variables for these options can be found below.
# Default is true.
nginx_install: true
...
```

ANSIBLE GALAXY: CREATING YOUR OWN ROLE

You can use the `ansible-galaxy` tool to create your *own* roles, as well, generating all the boilerplate:

```
$ ansible-galaxy role init roles/classrole
- Role roles/classrole was created successfully
```

LAB: USING A ANSIBLE GALAXY ROLE

- Use `ansible-galaxy` to install the `geerlingguy.docker` role
- Use the role to create a (simple) playbook which will (on your server)
    - install docker using the role
    - add your user to the list of docker_users (use a var)

Hint: configuration information on the role is available on the readme page

SECRET MANAGEMENT USING THE VAULT

We often need to store sensitive data but would prefer not to store it in clear-text (database credentials, third-party API keys, etc.). Ansible's solution to this is the **Vault**.

Any Ansible yaml file can be encrypted and then used in a playbook as variables. We can create encrypted yaml files using ansible-vault, e.g.:

```
ansible-vault create vault.yml
```

We can also edit existing vault files (if we have the `EDITOR` environment variable configured:

```
export EDITOR='nano`
ansible-vault edit stack-key.yml
```

USING THE VAULT PASSWORD FILE

We can use the `--vault-password-file` option. The filename you provide can be either a static file containing the password or a program that returns the password. To use Lastpass to store your secrets, for instance, you can use the following script:

```
#!/bin/bash
/usr/local/bin/lpass show --password PASSWORDNAME
```

VIEWING THE VAULT

```
$ ansible-vault view --vault-password-file vault-password.txt
ansible-class-repo/vault.yaml
foo: bar
baz: bat
anything: |
  Can go here, but it's usually nice to have YAML so we
  can use this in an "include_vars" statement....
```

SINGLE-STRING ENCRYPTION

Sometimes you may just want to embed an encrypted value directly in your playbook. For that, we have

`ansible-vault encrypt_string`:

```
$ ansible-vault encrypt_string --vault-password-file vault-password.txt "this is super
!vault |
          $ANSIBLE_VAULT;1.1;AES256
          66633137303563303865663461366566336431373764653565356166666534316161666532373
          38303166633764623733633961613865663337313933393370a336361643162313730613031353
          62376430306366386365393531376165653961393036393938306561333037383963353566323
          36633734373937643 40a396665626436363538336236373732383935636336633932653438373
          61623637303331393361376439343834316432663264313132386166366431666636
Encryption successful
```

SINGLE-STRING ENCRYPTION

Once we have the output of `encrypt_string`, we can use it in a playbook:

```
$ ansible-playbook --vault-password-file vault-password.txt ansible-class-repo/vault-de

PLAY [localhost] ***************************************************************

TASK [Show the encrypted_value] ************************************************
ok: [localhost] => {
    "encrypted_value": "this is super seekrit"
}

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

ORGANIZING VAULTS & VARS

One common practice is, when using the `host_vars` and `group_vars` directories, include both encrypted vault files and unencrypted variable files in the directories.

- Include secrets (with a naming convention) in vault.yaml
- Reference the secret variables in vars.yaml

vars.yaml

```
username: larry
password: "{{secret_password}}"
```

vault.yaml (unencrypted)

```
secret_password: foo
```

LAB: USING THE VAULT

- Create a playbook that will create a few users on your server (give them whichever names you want)
- Use the `include_vars` task to include a vault file containing their usernames and passwords

ORCHESTRATION

Ansible has a number of modules that support various cloud providers, including

- Amazon
- Azure
- VMware
- Google
- DigitalOcean
- ... and several more

We'll look at EC2 in particular.

EXPLORING EC2 AND ANSIBLE

http://docs.ansible.com/ansible/latest/list_of_cloud_modules.html#amazon

Many of our commands with EC2 will use host delegation (particularly running locally)

By default, it will use the AWS CLI configuration.

GETTING FACTS ABOUT YOUR INFRASTRUCTURE

You can ask ansible to report on your EC2 instances using the `ec2_instance_info` module:

```
$ ansible localhost -m ec2_instance_info
localhost | SUCCESS => {
    "changed": false,
    "instances": [
        {
            "ami_launch_index": 0,
            "architecture": "x86_64",
            "block_device_mappings": [
                {
                    "device_name": "/dev/sda1",
                    "ebs": {
...
```

CREATING INSTANCES

We can, of course, create instances as well. Here is how the student servers for (a previous version of) this class were created:

```
- name: Create student servers
  loop: "{{users}}"
  become: no
  register: student_servers
  local_action:
    module: ec2_instance
    region: "{{region}}"
    name: "{{item.username}}-instance"
    key_name: ansible-class-key
    image_id:   ami-070703f38341d8316
    vpc_subnet_id: subnet-0da0eed5db76664f3
    instance_type: t3.nano
    security_group: ansible-class-sg
    network:
      assign_public_ip: yes
    tags:
      student_email: "{{item.email}}"
      student_name: "{{item.name}}"
```

ORCHESTRATION USING TERRAFORM

You can use Ansible to run your Terraform plans as well:

`terraform.yaml`

```yaml
- hosts: localhost
  gather_facts: false
  tasks:
  - terraform:
    project_path: '{{ project_dir }}'
    state: present
    # workspace: ...
```

```
$ ansible-playbook ansible-class-repo/terraform.yaml -e project_dir=~/src/arborian-clas

PLAY [localhost] ***************************************************************

TASK [terraform] ***************************************************************
ok: [localhost]

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