Copyright Information

Published in the United States by Alta3 Research, Inc.

4 Bonnywick Drive
Harrisburg, PA 17111
Phone: *717-566-4428*
Web: *https://alta3.com*
Email: *info@alta3.com*

Welcome!

Thank you for selecting Alta3 Research as your training provider! Since 1997, Alta3 Research has been empowering organizations and individuals using a no-nonsense approach to IT and DevOps training. We specialize in technologies such as Python, Ansible, 5G, Software Defined Networking, Microservices, Kubernetes, GoLang, Jenkins, GitHub, GitLab and more.

Alta3 Research excels at taking the advanced technologies required for IT and DevOps professionals, breaking them down and building proven training courses with over 95% student satisfaction. Training is available at the customer's site or virtually through instructor-led webinars. For more information on additional courses to help you achieve your career goals, please visit us at https://alta3.com

PDF Content Expiration

- **PDF created: 2022-02-22**

- **Refresh your content at: https://alta3.com**

Subscribe to us on these social media platforms for updates and free training!

- **YouTube:** https://youtube.com/alta3research
- **Twitter:** @alta3research
- **Facebook:** https://facebook.com/alta3research
- **LinkedIn:** https://www.linkedin.com/company/alta3-research-inc

Table of Contents

Gennady Frid
gfrid@bloomberg.net

1. Welcome to Alta3 Research Labs

Lab Objective

Welcome to the Alta3 Research Lab environment! This page lists some common shortcuts you might want to use throughout the course. At the bottom, you'll also find a list of common questions and solutions.

Helpful Resources

Alta3 Research Instruction & Training: https://alta3.com/

Alta3 Research Posters & Cheat Sheets: https://alta3.com/posters

Alta3 Research YouTube: https://www.youtube.com/user/Alta3Research/videos

tmux (CLI) Shortcuts (tmux)

The CLI uses an application called **tmux** that allows for some really cool screen tricks.

- **Paste into tmux (CLI):**
  - **Ctrl** + **Shift** + **v**
- **Paste into tmux (CLI):** *2nd trick*
  - **Shift** + **Insert**
- **Split tmux (CLI) horizontally:**
  - **Ctrl** + **b**, take your hands off the keyboard, and then press, **Shift** + **'**
- **Split tmux (CLI) vertically:**
  - **Ctrl** + **b**, take your hands off the keyboard, and then press, **Shift** + **5**
- **Close an 'extra' tmux (CLI) pane:**
  - Type `exit`
- **Create a new tmux (CLI) session:**
  - **Ctrl** + **b**, take your hands off the keyboard, and then press **Shift** + **;** next type `new-session` and press **Enter**
- **Scroll UP / DOWN:**
  - **Ctrl** + **b**, take your hands off the keyboard, and then press **[** (left square bracket). Next press the up and down arrow to scroll up or down. To exit, press `q`

Using the ~/static/ folder

Within the home directory you'll find a directory, `~/static`. Anything you move into this folder is easily reached by changing the "source" of your right most pane, when using the split screen feature. Changing a source is as simple as clicking the icon that looks like three sheets of paper stacked on top of each other.

git Shortcuts

- **Display information about current branch:**
  - `git status`
- **Add file to current branch for tracking:**
  - `git add *`
- **Create a commit:**
  - `git commit -m "reason for commit"`
- **Push to origin:**
  - `git push origin main`
- **Redefine the `origin` var:**
  - `git remote set-url origin git@GitHub.com:<username>/newAnsRepo.git`

Common Questions and Solutions

- **My screen has "dots" around it :(**
  - *If you have more than one screen open, the "smallest" resolution wins. Therefore, the solution is to close the second tab you have open.*
- **HELP! My screen STILL has "dots" around it!**
  - *Create a new tmux "session" with the following* **Ctrl** + **b**, *take your hands off the keyboard, and then press* **Shift** + **;** *next type* `new-session` *and press* **Enter**
- **I typed exit too many times and my CLI session closed!**
  - *Press the Refresh icon on your browser to refresh your session and create a new tmux session*
- **How do I get my content out of the tmux (CLI) environment?**
  - *The "best" way is probably by using git and GitHub. Alternatively, if you move content into the `~/static/` folder, you may access it by changing the "source" of your right-most pane in the split-screen session. To change the source, click on the icon that looks like three sheets of paper in the upper-right hand corner.*
- **Will the lab environments "shut off" this week?**
  - *No. The your lab environment is on 24x7, until the termination date.*
- **Can I close the tab to live.alta3.com**
  - *Yes! Unless you clear your internet cache, you should just be able to revisit your custom course link to regain access. If you are asked to log-in again, simply use the same email address and handle to regain access.*
- **Is there a "best" browser to use?**
  - *At this time Chrome and Microsoft Edge seem to support all of the tmux tricks best.*

2. Ansible Essentials for Dell Storage Products

- **5 days**
- **Lecture & Labs**

Upon the completion of the course, students are able to earn a certificate by writing a playbook that automates a Dell Storage product with Ansible.

Course Overview:

This course offers a flexible (customizable) overview of using Ansible to automate the suite of Dell storage products including PowerMax, Isilon/ PowerScale and PowerStore. Examples highlight Ansible's latest current release capabilities, building complex playbooks, and developing workflow strategies all while observing best-practice techniques.

Students will be encouraged to share with the class the manual ways they are currently doing work in an effort to create highly applicable Ansible solutions. Through these demonstrations, students will learn how to use code to extend structure and consistency to their specific job operations. Although this course focuses on Dell Storage solutions, Ansible is a Swiss-Army knife of abstracted automation. Every lesson is highly applicable beyond "just" the Dell storage suite.

What You'll Learn:

- Ansible for Storage Strategies
- Ansible and Dell PowerMax
- Ansible and Dell Isilon / PowerScale
- Ansible and Dell PowerStore
- Ansible and Dell PowerFlex (VxFlex OS)
- Ansible and Pure Storage
- Ansible for NetApp Storage
- Writing and Executing Ansible Playbooks
- Developing Roles with Molecule
- Installing and creating Collections for content delivery
- Using Ansible to target Linux, Windows, Clouds, Network Devices and REST APIs
- Ansible and AWX / Tower for Orchestration
- Ansible and Jenkins for Orchestration

Course Outline:

1. Ansible Introduction
    - Ansible definition
    - Exploring modules
    - Building a task
    - Collections (content delivery)
    - Places to define Ansible vars
    - hosts aka "Inventory"
    - Creating a “play”
    - Looking at the “handler”
    - Running a playbook
    - Requirements for connecting to remote hosts
2. Installation and Configuration
    - Configuration requirements on the control machine
    - Understanding pre-requisites for connecting to remote infrastructure
    - Ansible configuration with ansible.cfg
    - Ansible connection types
    - Touring the Ansible Python Plugin system
3. Ansible Static Inventory
    - Defining Hosts and Groups
    - Host and Group variables
    - Understanding variable precedence
4. YAML
    - YAML Spec
    - YAML Dictionary
    - YAML list
    - YAML list of dictionaries
    - YAML Alternate format
    - Relationship to JSON
5. Writing a Simple Playbook for Dell Storage (PowerMax, Isilon/PowerScale, PowerStore and PowerFlex)
    - Elements of a playbook
    - Dell Storage Collections
    - Using include files for tasks
    - Available Dell Storage modules
    - Creating dynamic playbooks with external variable files
    - How a Dell inventory may differ from static inventories
6. Ansible base modules to know
    - Understanding modules documentation
    - setup / gather_facts
    - gathering facts on Dell storage arrays
    - apt / yum / pip
    - command / shell
    - uri module for API lookups
    - git
    - debug for variable display
    - lineinfile for building configuration files
    - Writing Ansible Playbooks with Dell Storage Modules and Ansible Base modules
7. Variables, Conditionals and Looping Tasks
    - Variables and Loops
    - Building conditionals from Dell Storage module results
    - Blocks

- Getting variables from the system
- Setting variables in playbooks
- Getting variables from the command line
- Where is the best source to derive variable values?

8. Ansible and Jinja

   - What is Jinja2?
   - jinja variables
   - jinja filters
   - How to use ansible template
   - Building Jinja2 templates for Dell Storage

9. Ansible and RESTful API Requests

   - Passing parameters within RESTful API requests
   - General tips for connecting Ansible to any API
   - AWS S3 Bucket Storage Example

10. Dynamic Inventory Management

    - What is Dynamic Inventory?
    - A review of static Inventory Practices
    - Using JSON as an Inventory Source
    - Using YAML as an Inventory Source

11. Securing Credentials for Dell Storage

    - Best practices for credential management
    - Prompting for sensitive input
    - YAML shortcuts for passing credentials to Dell Storage
    - Securing credentials with Ansible Vault
    - Encrypting Playbooks with Ansible Vault

12. Collections, Roles and Ansible Galaxy

    - Why we need Ansible Roles
    - The problems solved by Ansible Collections
    - Creating Roles
    - Creating Collections
    - Role Directory Structure
    - Role default variables
    - Converting a Playbook to a Role
    - Exploring Ansible Galaxy
    - Finding code on GitHub
    - Augmenting Ansible with Ansible Galaxy content (Roles and Collections)

13. Ansible Tower / Jenkins

    - Running Dell Storage playbooks in Ansible Tower
    - Running Dell Storage playbooks in Jenkins
    - Why you should consider using Jenkins as a replacement for Ansible Tower
    - Pushing and pulling playbooks from GitHub

Hands On Labs:

1. Installing Dell EMC Storage Collections
2. Git and Software Control Management Platforms
3. Methods for Installing Ansible
4. Ansible Host Inventory
5. Running your first Playbook
6. Debug, Loops, and YAML Lists
7. When Conditionals, YAML Dictionaries, and Jinja

8. ansible.cfg setup
9. Building Playbooks using ‘mount’, ‘shell’, ‘copy’, ‘uri’, ‘file’, ‘template’, and ‘git’
10. Ansible Galaxy and Roles
11. Ansible Collections
12. Ansible for Dell PowerMax
13. Ansible for Dell PowerScale Isilon
14. Ansible for Dell Unity
15. Ansible for Dell ECS S3
16. Ansible for Dell PowerStore
17. Ansible for Dell PowerFlex (VxFlexOS)
18. Ansible for Pure Flash Array
19. Ansible for NetApp Storage
20. Playbook Tags
21. Handlers & Listeners
22. Ansible Error Handling - Rollback Strategy for Storage
23. Ansible Keywords - register and when
24. Reading Variables into Playbooks
25. ansible-doc
26. Ansible Plugin System
27. YAML, JSON, Dynamic, and Cloud Inventories
28. Ansible and AWS
29. Playbook Vars Prompt
30. Ansible Vault (Securing Credentials for Dell Storage)
31. Molecule Testing
32. Ansible Playbook Output Logging
33. Ansible ‘script’ module
34. Automating Brocade Switches
35. Introduction to Jenkins
36. Introduction to Ansible Tower
37. Ansible Best Practice

Prerequisites:

- Strong typing skills
- Coding experience in another language is helpful, but not required

Who Should Attend?:

- Storage Engineers (specifically those using Dell Storage products), System Administrators, Network Engineers, and Developers will find this course a compelling overview for using Ansible to automate their workflows.

Follow-on Courses:

- Git and GitLab (2 days), Git and Bitbucket (2 days), or Git and GitHub (2 days)
- Python Basics (5 days)
- API’s and RESTful API Design with Python (5 days)
- Jenkins Automation Server Essentials (2 days)

3. Using vim

Lab Objective

Throughout the course you'll find our documentation suggests using the vim text editor. Vim is an improved version of vi, so if you know vi you'll just be refreshing some basic skills in this lab.

Vim is the editor of choice for many developers and power users. It's a "modal" text editor based on the vi editor written by Bill Joy in the 1970s for a version of UNIX. It inherits the key bindings of vi but also adds a great deal of functionality and extensibility that is missing from the original vi.

In most text editors the alphanumeric keys are only used to input those characters unless they're modified by a control key. In vim, the mode that the editor is in determines whether the alphanumeric keys will input those characters or move the cursor through the document. This is what is meant by 'modal.' When you first enter vim, you enter in the command mode.

Procedure

1. Review (read-only) the following **vim** commands:
    - **To start editing changes by entering the INSERT mode:**
        - press `i`
    - **To stop editing and return to command mode:**
        - press `ESC`
    - **To save and quit:**
        - press `SHIFT` + `:` *press SHIFT and the PLUS keys at the same time*
        - type `wq` (write out and quit)
        - press `ENTER` to confirm
    - **To quit without saving:**
        - press `SHIFT` + `:` *press SHIFT and the PLUS keys at the same time*
        - type `q!` (quit and ignore all changes)
        - press `ENTER` to confirm
2. Move into your home directory.

```
student@beachhead:~$ cd
```

3. Now create a text file within the vim environment.

```
student@beachhead:~$ vim secretofmi.test
```

4. Vim is entered in command mode. To write text, you'll need to change to INSERT mode. To begin writing text, press:
    - `i`
5. Notice in the bottom left corner of the screen it now says INSERT
6. Type a few sentences. Be sure to include some carriage returns, like the following:

```
Guybrush: Hi! I'm Guybrush Threepwood, and I'm a mighty pirate!
Pirate: Guybrush Threepwood? That's the most ridiculous name I've ever heard!
Guybrush: Oh yeah? What is yours?
Pirate: (matter-of-factly) My name is Mancomb Seepgood.
```

7. Okay, great! Now leave INSERT mode, and return to command mode, by pressing the escape key.
    - `ESC`
8. Notice that INSERT no longer is at the bottom left of the screen. Generally, pressing the escape key will always return you to the command mode.
9. Use the directional arrow keys on the keyboard to move the cursor around the screen.

10. Perform the following to save changes and return to the command line.
    - press `SHIFT` + `:` *press SHIFT and then the COLON key at the same time*
    - type `wq` (write out and quit)
    - press `ENTER` to confirm
11. Confirm that the file did correctly save by printing its contents. We can use the **cat** command to catenate, or read data from files, and print their contents to the screen.

    ```
    student@beachhead:~$ cat secretofmi.test
    ```
12. Edit the file secretofmi.test again.

    ```
    student@beachhead:~$ vim secretofmi.test
    ```
13. Remember, you enter vim in command mode. Take advantage of that and press the following capital letter to jump to the end of the file:
    - press `SHIFT` + `g` *press SHIFT and the g keys at the same time*
14. Press the following capital letter to begin appending at the end of the line (enter INSERT mode at the end).
    - press `SHIFT` + `a` *press SHIFT and the a keys at the same time*
15. You'll notice it says INSERT at the bottom left of your screen again. Once again you can type normally. Add some more text, such as the following:

    ```
    Guybrush: Hi! I'm Guybrush Threepwood, and I'm a mighty pirate!
    Pirate: Guybrush Threepwood? That's the most ridiculous name I've ever heard!
    Guybrush: Oh yeah? What is yours?
    Pirate: (matter-of-factly) My name is Mancomb Seepgood.
    Taken from "The Secret of Monkey Island" by Lucas Arts (1990)
    ```
16. Okay, great! Now leave INSERT mode and return to command mode by pressing the escape key.
    - `ESC`
17. Perform the following to return to the command line **without** saving any changes.
    - press `SHIFT` + `:` *press SHIFT and the COLON keys at the same time*
    - type `q!` (quit without saving)
    - press `ENTER` to confirm
18. Confirm that none of the changes you just made were saved.

    ```
    student@beachhead:~$ cat secretofmi.test
    ```
19. Remove the file.

    ```
    student@beachhead:~$ rm secretofmi.test
    ```
20. Review a vim cheat sheet. These make very useful wall art, you may have seen one hanging in a colleague's workspace. http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html
21. If you want to get REALLY good at vim, run the following command. You don't need to do the entire tutorial now, but try reading the first few lessons.

    ```
    student@beachhead:~$ vimtutor
    ```
22. Quit with `:wq`
23. If you got stuck a few times, go back and try it again! Working within the vim environment is a basic Linux admin skill that is useful to everyone that expects to work at the Linux CLI.

4. Revision Control with Git and GitHub

In this lab, we're going to explore SCM, or Software Control Management platforms. Git is the de-facto tool for tracking and version controlling your work. Git is a tool that we run on our local machine. GitHub is an HTTPS browser-friendly platform that syncs with git, and makes your code available to the world. This is a good thing.

In this lab, you'll make a GitHub account. This is free and will allow you to save the code you develop this week.

Procedure - GitHub Account

1. In your local browser, open a new tab to https://github.com
2. Click on `Sign-up` on the landing page.
3. Enter the following information.
    - **Username** - This handle will be shared with career professionals.
    - **Email Address** - Use an email address you check often.
    - **Password** - Always make passwords unique and ultimately change them often.
4. Now click on the **Create account** button at the bottom of the screen.
5. At the bottom of the screen, click the green `Continue` button.
6. Answer the questions as best you can to help the GitHub metrics, and then click `Submit` at the bottom of the screen.
7. You'll need to verify your email address. Check the email address you used to sign up, and click on the link or button they sent to you.
8. Now that we have a verified GitHub account, let's try tracking some work. In the upper right hand corner, click on your profile icon, then on `Your Profile`. This is where you can choose to create a bio and a profile. This account is public and the world will look at your code, so be professional.
9. In the center of the screen, click on the word `Repositories`, which is followed by a `0`.
10. Click on the green `New` button, with the little book icon on it. This creates a repository. Repositories track work.
11. Set the **Repository name** as `mycode`.
12. Set the **Description** as something like, `Tracking my code`
13. The repository should be set to **Public**.
14. The README option is always best practice, so check the box. The objective is to describe what your repo contains.
15. Change `Add a license: None` to `GNU General Public License v3.0`. Here's the TLDR, "We're comfortable with the world borrowing our code, provided they give us credit, should it get used."
16. Now click the green `Create repository` button at the bottom of the screen.
17. Your new repo will appear. Click on the file, `README.md`.
18. You'll find a small pencil on the right side of the screen. Click on this.
19. Enter the following template into your **README.md** and then customize it. The goal is to advertise to the world the purpose of your repository. Don't hold back. If you're a 1st time student interested in Microservices and/or Python, let the GitHub community know! FYI, files on GitHub are written in **markdown** which is a web-friendly way to format text. Read about it here: https://guides.github.com/features/mastering-markdown/

```
# mycode (Project Title)

One Paragraph of your project description goes here. Describe what you're trying to do.
What is the purpose of putting up this repo?

## Getting Started

These instructions will get you a copy of the project up and running on your local machine
for development and testing purposes. See deployment for notes on how to deploy the project
on a live system.

### Prerequisites

What things are needed to install the software and how to install them. For now, maybe copy in
"how to install python and python3 using apt."

## Built With

* [Python](https://www.python.org/) - The coding language used

## Authors

* **Your Name** - *Initial work* - [YourWebsite](https://example.com/)
```

20. When you finish, click the green `Commit changes` button at the bottom of the screen.
21. Generally, we don't want to use GitHub as a development tool. Just for viewing. But to get our README.md started, this is fine. From now on, we'll only add to our repo from the command line.
22. At the top of the screen, click on your username to return to your homepage.
23. You should see your repository `mycode`.
24. GitHub lets you have friends just like other social media platforms. Checkout all of the Alta3 Python Instructors:
    - Zach
    - Sam
    - Chad
25. While you are there, click follow, to follow all of your Alta3 Python Instructors.
26. You might try asking some fellow students for their usernames and repeating the above steps.

    RETURN TO YOUR ALTA3 TMUX ENVIRONMENT NOW

27. Now we want to sync our command line (where we run git) with GitHub. The goal is to get our code into the **mycode** repo we created. Move to the **~/.ssh** home directory.

```
student@bchd:~$ cd /home/student/.ssh
```

28. Now we will generate a new RSA keypair. We will use this to communicate securely with GitHub because as of August 2021, GitHub will no longer accept passwords from us. This will generate two files, the private key (**id_rsa_github**) and the public key (**id_rsa_github.pub**).

```
student@bchd:~/.ssh$ ssh-keygen -f id_rsa_github

Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
```

> When you see this, you will need to hit enter twice

```
Enter same passphrase again:
Your identification has been saved in id_rsa_github.
Your public key has been saved in id_rsa_github.pub.
The key fingerprint is:
SHA256:DFjw7WCP+u2luMtPATqKHGBi/+XNUzlRYfqIYz8S2/Q student@bchd
The key's randomart image is:
+---[RSA 2048]----+
|    ...      +.  |
|     + .    +    |
|oo  . * .  o     |
|= .  o O  . =    |
| . .o ..S= * .   |
|.....oo +.O o    |
|... .. ..*.+ E   |
|     o + oo .    |
|      *==        |
+----[SHA256]-----+
```

29. Next we need to copy the text from inside of the **id_rsa_github.pub** public key into our clipboard. We will need to paste this into GitHub soon.

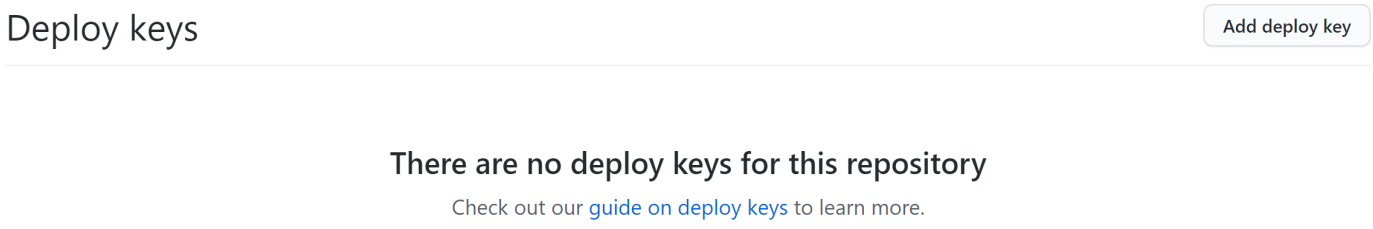
```
student@bchd:~/.ssh$ cat id_rsa_github.pub

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCvTemj7NIlxXEu97VyN4wcmob+F0wQ0nPinZSdUvlWSGAx790EmfXmckZ01/
aFDOKIA4OZWYtW95DaGqQ+Tja3QUYlAdTdIlp4TTgH3ZE+KdaLc16rN9FzHv8hwdLFx8CugNw/u/
sDbYjEM1qlazB5fbAf0LZ+mN5iCDn6IaYbPZ0wQdF9s4RI/z5wS5wE/J++KV/xJA0f2ICZwaKj4Kq/
fron2KoYRkCYcma0oHYvVSnnuxCVGbOJdhWK1LiJRPweNUajp0OItECJxgCR1gB/
DVEWnBmknOcPnY6q3QeaUrm28nPSAjGTkfFSHDbOJptaqKaSkRhwNsuOGiw6KQ1R student@bchd
```

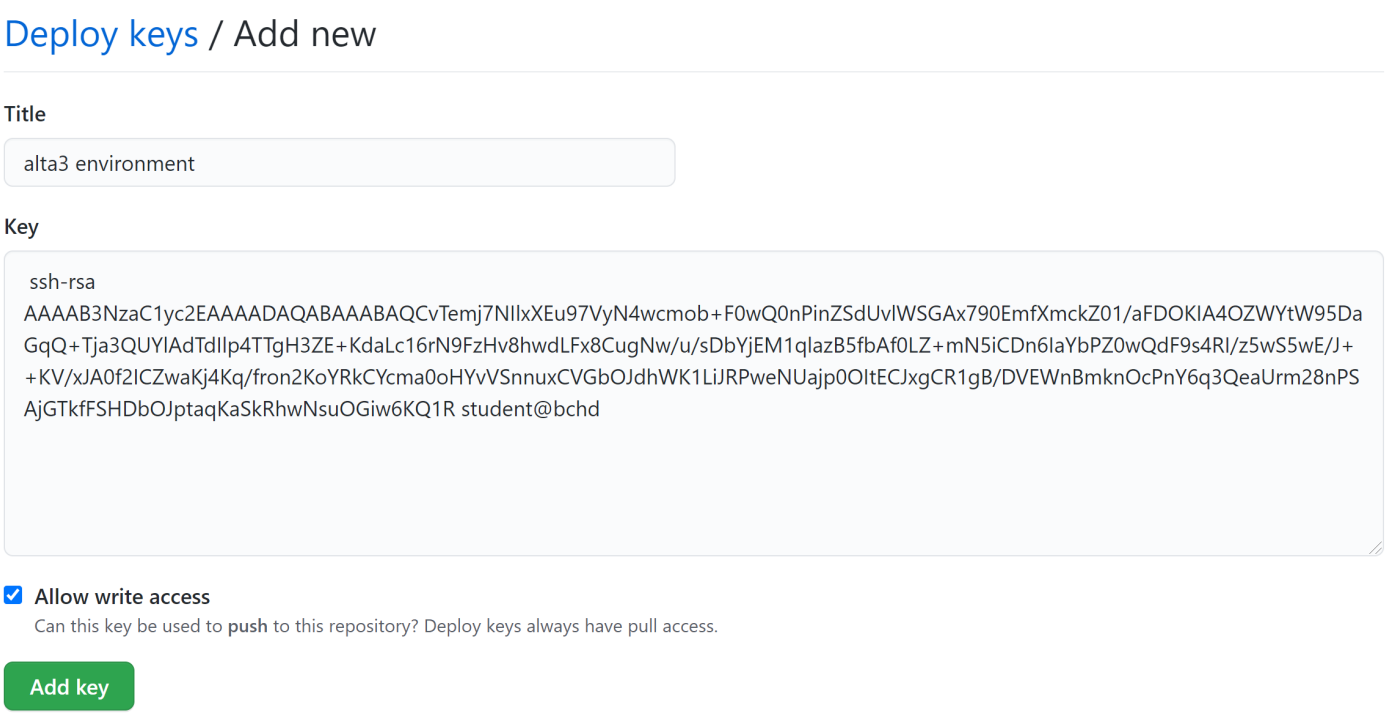
Select all of the output and copy it. NOTE: Yours will be different than the one listed above!

RETURN TO GITHUB NOW

30. At the top of the screen, click on your username to return to your homepage.
31. You should see your repository `mycode`. Click into it.
32. Near the top right of the screen, there is a little gear icon that says **Settings**. Click on that.
33. On the left of the screen, click on **Deploy Keys**. Deploy keys are essentially a token that we set up that allow somebody to read from, and optionally write to, our repository, even if it is private. They just need to have the private key set up on their system as a means of authenticating with the public key stored on github.
34. Your screen should look like the following. Click on the button that says **Add deploy key**.



35. Give your key a name. **alta3 environment** would work. Then paste your key (from **id_rsa_github.pub**) into the large text box. And also click on the checkbox that says **Allow write access**. Then, click **Add key**.



RETURN TO YOUR ALTA3 TMUX ENVIRONMENT NOW

36. Fantastic. Let's set up `git` for first-time use. First of all, let's move back to our home directory.

```
student@bchd:~/.ssh$ cd ~
```

37. Replace the name **Mona Lisa** with **your own!** When you perform a git commit, this is the name that will be applied to the commit (the save).

```
student@bchd:~$ git config --global user.name "Mona Lisa"
```

38. Replace the username portion (only) of the email address **monalisa@users.noreply.github.com** with **your** github username. When you perform a git commit, this is the email that will be applied to the commit (the save). The domain `users.noreply.github.com` with cause emails sent to you to end up in your GitHub account (not your SMTP inbox).

```
student@bchd:~$ git config --global user.email "monalisa@users.noreply.github.com"
```

39. Customize the command below so that the entire `<github-username>` is replaced with only your GitHub username. The angle brackets are **not allowed**.

```
student@bchd:~$ git clone git@github.com:<github-username>/mycode.git
```

> If you run into an error with this step, read the instructions once more and try it again!

40. You should now have a new folder (mycode) in your home directory. Move into it.

```
student@bchd:~$ cd /home/student/mycode/
```

41. Let's edit README.md

```
student@bchd:~/mycode$ vim /home/student/mycode/README.md
```

42. At the top of your `README.md` file, within your project description, add a sentence about wanting to learn how to version control projects with git. (Press the `i` key to enter insert mode).

43. Exit insert mode with `ESC`.

44. Save (write) and exit with `:wq` and then **ENTER**

45. The following steps are ones you should memorize. You'll be issuing these commands all the time. First issue `git status`, which will reveal if you added something and forgot about it.

```
student@bchd:~/mycode$ git status
```

46. Next we'll describe what we want to add to our repo. We'll wildcard this, so everything in the `~/mycode/` directory is added.

```
student@bchd:~/mycode$ git add *
```

47. Time to perform a commit. This makes a new version.

```
student@bchd:~/mycode$ git commit -m "First commit to learn about version controlling"
```

48. Now push your new version to GitHub for tracking. By default, the keyword **origin** points to the repo you cloned the repo from. Enter the following followed by your username and password.

```
student@bchd:~/mycode$ git push origin
```

49. Go up to GitHub, and see if your README.md file has changed. **Remember, we don't want to edit any files in GitHub anymore.** Just validate that GitHub has the new data.

50. You should get in the habit of issuing the following commands each time you have a success, or end for the day:

    - `git status`
    - `git add /home/student/mycode/*`
    - `git commit -m "reason for commit"`
    - `git push origin`

51. Great! Move back to your home directory.

```
student@bchd:~/mycode$ cd /home/student/
```

Troubleshooting

The following items resolve most student issues with connecting to GitHub.

"origin" is defined incorrectly

The variable `origin` needs to be set to use SSH protocol, not HTTP. You can check how it is set by issuing, `cat ~/mycode/.git/config`.

If you see mention of `url = https://github.com/...`, you need to redefine `origin` by issuing the following command from inside the directory `~/mycode/`

`` `git remote set-url origin git@github.com:YourGitHubUsername/mycode` ``

You'll need to replace, `YourGitHubUsername` *with your actual GitHub account name*.

The public key you generated is misnamed

The key-pair you generated should be named, `id_rsa_github` and `id_rsa_github.pub`. If they are named something else, regenerate them with the correct name, and place the new public key on GitHub.

Alternatively, you can update the entry in the file `~/.ssh/config`. There is an entry for `github.com` that indicates the name of the private key to use (private keys *do not* have `.pub` extension). Change the entry the config file to reflect the name of the keypair that *you* created. Save and quit, then try to push to GitHub again.

Histories cannot be merged

If you are seeing this error, the safest and easiest solution is to delete the repository on GitHub.com (but not locally). Create a new `mycode` repo on GitHub *without* a `README.md` (or any other file). This is called a "bare" repository. Try pushing your code to this new repo.

Nothing to push

You need a commit to push to GitHub.com. Get a list of all the commits in your local repository with `git log`. If you don't see anything recent, you might need to perform the `git add` / `git commit` commands before you can get a commit to push to GitHub.

Other Useful Resources

- If you ever run into a problem using git / GitHub, let the instructor know. It is critical to start learning to version control code. You might want to spend some time clicking around on the following guides. They're quite short, and although you might not understand what they're talking about, they'll begin exposing you to the way we 'speak' when using git & version controlling software. Some getting started guides relating to GitHub can be found here: https://guides.github.com/
- Windows and Mac users might also check out the local client GitHub Desktop, available at https://desktop.github.com/ This client makes it **easy** to work through git via a local GUI on your Windows or Mac desktop environment.

5. Methods for Installing Ansible

Lab Objective

The objective of this lab is to learn how to install Ansible, as there are multiple ways to deploy the software. One universal truth is that Ansible only runs on Linux platforms. Each has its own procedure and can result in different Ansible versions being installed. This lab introduces installing the latest stable version of Python and Ansible.

Ansible ONLY needs to be installed on the 'controller'. For us, that is *beachhead* (i.e. `@bchd`), however, since Ansible is already installed here, we'll try installing it on a remote host.

In production, the target hosts (the hosts Ansible will connect to) should have Python installed. The exceptions to this, would be network devices, or APIs, which we will cover later.

Procedure

1. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd
   ```

2. Download the tear-down script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

3. Run the tear-down script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

4. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

5. Run the setup script for this lab.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

6. The script will take a few minutes to execute. Allow it to run to completion.

7. SSH into the `bender` machine, the password is `alta3`

   ```
   student@bchd:~$ ssh bender@10.10.2.3
   ```

8. This machine is a freshly installed Debian / Ubuntu machine. Pretend you just installed Linux, and you want to build an Ansible controller. The first step, update the package installer, apt.

   ```
   bender@bender:~$ sudo apt update -y
   ```

9. Install Python and the package installer, pip.

   ```
   bender@bender:~$ sudo apt install python3-pip -y
   ```

10. The install will fail if pip is not updated.

    ```
    bender@bender:~$ python3 -m pip install --upgrade pip
    ```

11. Now install Ansible using the Python package installer, pip. Note how we call the command. It is the opinion of the author that you **AVOID** using a shortcut like `pip install ansible` in favor of `python3 -m pip install ansible`. This helps avoid installation headaches. The `--user` flag installs it for the local user (`bender`).

    ```
    bender@bender:~$ python3 -m pip install ansible --user
    ```

12. Source your profile as a final step! Without doing this, you'd need to start a new session.

    ```
    bender@bender:~$ . ~/.profile
    ```

13. This will take awhile... be patient, when it completes, Ansible will be installed on the `bender` host. Ensure this is the case.

```
bender@bender:~$ ansible --version
```

14. Answer the following questions:
    - **Q: What is the version of Python supporting Ansible?**
      - *A: This should be at least Python 3.6 or later. At this time, you should never be running Ansible on Python 2.x. All version of Python 2 are now historic.*
    - **Q: What is the version of Ansible you are running?**
      - *A: The version is at the top of the display on the screen*
    - **Q: What is the latest current release of Ansible?**
      - *A: Go to https://github.com/ansible/ansible and click the branch dropdown that says "devel". Scroll down until you see the latest version following the word "Stable-" This is the latest available version of ansible. It should also be the version installed on your system.*

15. Exit from the bender machine.

```
bender@bender:~$ exit
```

16. **CHALLENGE 01 (OPTIONAL)** - SSH to `farnsworth` at the IP address 10.10.2.6 (password `alta3`). This machine is a RedHat CentOS machine. Your goal is to install Ansible on this platform. The steps are nearly identical, however, you'll need to replace the word `apt` with the word `yum`.

17. When you finish be sure to **CLEANUP**! Running the teardown script will delete the `bender` and `farnsworth` machines we built. This will ensure we have a clean environment for subsequent labs.

```
student@bchd:~$ bash max-teardown.sh
```

18. Great job! You have completed this lab.

Reference (READ ONLY)

For anyone running an Apple Mac that wants to upgrade or install Ansible on Python 3.x, use the following steps:

1. Remove the old or current version of Ansible using the inverse way you installed it (example: `python -m pip uninstall ansible`)
2. Check if homebrew is installed with `brew doctor`. You should get a clever response that it is time to brew the strongest potions or some such silliness.
3. If you got a message indicating brew was not recognized, you need to install it. First install xcode `xcode-select --install`
4. Great. Now install homebrew: `ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`
5. Cool. You now should be a potions master. Confirm this with `brew doctor`
6. Now install python3. `brew install python3`
7. Install Ansible to the local user `python3 -m pip install --user ansible`
8. Fix your path! Without this step, you won't be able to use the command `ansible`. *NOTE: Change RZfeeser to your local user* `export PATH="/Users/rzfeeser/Library/Python/3.9/bin:$PATH"`
9. You should now be able to run `ansible --version` and see the latest version of ansible supported by Python 3.x

6. Ansible Host Inventory

Lab Objective

The objective of this lab is to introduce the creation of the Ansible Inventory, and the concept of Ansible Ad-Hoc commands.

The Ansible inventory is a description of hosts organized into groups and or sub-groups. Read more about inventory here:
https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html

Types of Ansible Inventories:

- **Static** - A user defines a list of hosts in a YAML, JSON, or INI format. Other formats exist, such as *commas separated hosts* (CSH), however, the aforementioned are the most popular formats.
- **Dynamic** - A user creates a python script that audits infrastructure, and then "prints" YAML or JSON, which is then used by YAML as the inventory source. This is great for situations where a human could not possibly create a *static* inventory fast enough. Think situations like clouds, or networks, where devices come on and off line in a hurry.
- **Hybrid** - A combination of static and dynamic sources.

The Ansible Ad-Hoc mode allows users to run 'one off' commands without creating playbooks. In this lab, we'll use two modules, `ping` and `command`. The module `ping` tests SSH connectivity and confirms Python is installed on the remote hosts. If the ping is successfully, the remote hosts will respond with a 'pong'. To be clear, this module may be poorly named, as it has nothing to do with ICMP.

Ping Module:
https://docs.ansible.com/ansible/latest/modules/ping_module.html

The module `command` allows a user to run CLI commands on remote hosts. This is the default module run when users are running ansible in Ad-Hoc mode. That is to say, if a user does not indicate a module to use in Ad-Hoc mode, the `command` module will be used.

Command Module:
https://docs.ansible.com/ansible/latest/modules/command_module.html#command-module

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd
```

2. Download the tear-down script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the tear-down script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id bender@10.10.2.3
```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id fry@10.10.2.4
```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. Generally, anything we can do within a playbook, we can do at the CLI with the command `ansible` (as opposed to `ansible-playbook`). This is called 'ad-hoc' mode, and is mostly used for testing or to ensure connectivity. The following asks ansible to target the localhost with the command module (`-m command`) along with argument `"whoami"`. The command module is akin to running commands at the CLI.

```
student@bchd:~$ ansible localhost -m command -a "whoami"
```

11. The results should look something like the following:

```
localhost | CHANGED | rc=0 >>
student
```

12. This might be your first time using a module. A module is a piece of Python code that we can ask Ansible to run for us. Read about the `command` module by issuing the following:

```
student@bchd:~$ ansible-doc command
```

13. Use up and down to scroll, press `q` to quit.

14. Run the CLI command `pwd` via the command module. Notice that we don't actually need to include the `-m command` (the default ad-hoc mode is to use the command module).

```
student@bchd:~$ ansible localhost -a "pwd"
```

15. The results should look something like the following:

```
localhost | CHANGED | rc=0 >>
/home/student
```

16. To do anything interesting, we need to create an inventory. Convention says we should call this file a `hosts` file, although, users can actually call this file anything they like, especially if that name better describes the inventory's contents. Regardless of the name, the inventory file instructs Ansible how to access the hosts we want to configure. The inventory also allows us to organize hosts into groups, which makes it possible to run Ansible against multiple remote hosts.

17. We want to stay organized, so create a directory folder where we can describe our inventory. This technique is a reflection of best practice.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

18. Great! Now create our inventory file:

```
student@bchd:~$ vim ~/ans/inv/dev/myhosts
```

19. An explanation of the file you're about to create is required. Hosts are organized into 'groups'. Hosts can have 'host variables' defined in-line. Any variable that begins with the word `ansible`, usually controls how Ansible runs. For example, the hostname to connect with, or the IP address to connect to. To run ansible against more than one machine, hosts are organized into 'groups', which are created with square-brackets. The host file you're about to create has several of these groups. Sub-groups, or *groups of groups* can also be created with the `:children` tag. Edit your host file as shown below.

```
[humanoid]
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3

[robot]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3

[deliverycrew:children]
humanoid
robot
```

20. Save and exit with `:wq`

21. The host `farnsworth` is set up to use a password. We know that because the variable `user_ssh_pass` is present in the host file. Anytime a password is used to make a connection, the program `sshpass` is required (on the controller). Install it now:

```
student@bchd:~$ sudo apt install sshpass
```

22. Now run Ansible against that hosts file you created. The following command will cause Ansible to SSH into the `bender` machine, and run the command `whoami`

```
student@bchd:~$ ansible robot -i ~/ans/inv/dev/myhosts -a "whoami"
```

23. Try again, only this time target the `humanoid` group. Also, try specifying the `command` group, even though that is the default module to use.

```
student@bchd:~$ ansible humanoid -i ~/ans/inv/dev/myhosts -m command -a "whoami"
```

24. Run help on the Ad-Hoc command to decode what you just did.

```
student@bchd:~$ ansible --help
```

25. Use the help command to answer the following questions:

    - **What does the `-i` flag indicate?**
        - *A: This describes the location of the inventory file.*
    - **What does the `-a` flag indicate?**
        - *A: This implies that the following command in quotes in an argument.*
    - **What does Ansible call the parameter that we used with the value "robot"?**
        - *A: The value "robot" is a group that is defined within the inventory file.*

26. Let's try using a new module, `ping`. This isn't an ICMP ping, but an Ansible ping. It's a kind of test, "Can Ansible SSH to the target location and locate a local instance of Python?" Read about the `ping` module now.

```
student@bchd:~$ ansible-doc ping
```

27. Press up and down to scroll, press `q` to quit.

28. Ensure the `deliverycrew` group is working. This group is actually made up two sub-groups, `humanoid` and `robot`

```
student@bchd:~$ ansible deliverycrew -i ~/ans/inv/dev/myhosts -m ping
```

29. It turns out that you can also describe inventories in additional formats, such as YAML. However, the full list may be found here:

30. If you'd like to try making a YAML inventory, create the following file:

```
student@bchd:~$ vim ~/ans/inv/dev/myhosts.yml
```

31. Create the following file YAML inventory which describes `humanoid` and `robot`. Ansible allows users to describe variables in LOTS of locations. Below, the group `robot` has some "group_vars" mapped to it. We'll study Ansible variable precedence later. For now, just take note.

```
---
all:
    hosts:
            # "ungrouped" hosts would go here
    children:
            humanoid:
                    hosts:
                            fry:
                                    ansible_user: fry
                                    ansible_host: 10.10.2.4
                                    ansible_python_interpreter: /usr/bin/python3
                            zoidberg:
                                    ansible_user: zoidberg
                                    ansible_host: 10.10.2.5
                                    ansible_python_interpreter: /usr/bin/python3
                            farnsworth:
                                    ansible_user: farnsworth
                                    ansible_host: 10.10.2.6
                                    ansible_ssh_pass: alta3
            robot:
                    hosts:
                            bender:
                                    ansible_user: bender
                                    ansible_host: 10.10.2.3
                                    ansible_python_interpreter: /usr/bin/python3
                    # group vars could go below here
                    # all hosts within robot would inherit
                    # (these vars are not used, just examples)
                    vars:
                            gateway_ip: 172.0.0.1
                            storage_type: nfs
                            in_production: false
```

32. Save and exit.
33. Try running the `ping` command with the new inventory.

```
student@bchd:~$ ansible all -i ~/ans/inv/dev/myhosts.yml -m ping
```

34. Try running the `ping` command against *only* the group `robot`

```
student@bchd:~$ ansible robot -i ~/ans/inv/dev/myhosts.yml -m ping
```

35. **CHALLENGE 01** - Ping the group `humanoid` using the inventory `myhosts.yml`
36. Tear down your lab. This will help setup for a clean environment during the next, and subsequent labs.

```
student@bchd:~$ bash max-teardown.sh
```

37. Answer the following questions
    - **Q: Why set up an Ansible inventory?**
        - *A: Setting up static inventory by groups allows flexibility in pre-planning your groups of hosts ahead of time.*
    - **Q: What kind of planning is necessary when designing an inventory?**
        - *A: When setting up our inventory, we should think carefully about how to group the hosts. They may be arranged by geographic area, platform, team, silo, in a single file, or across many files.*
    - **Q: Are there more ways to describe an inventory?**
        - *A: Yes. JSON inventories are common, and typically produced when you ask Ansible to use scripting to generate a 'real-time' inventory. This is known as "dynamic inventory" and covered in a subsequent lab.*
38. Great job! You have completed this lab.

39. If you are tracking your code on GitHub, take a moment and perform a commit.
    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about ansible inventories"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

7. Ad-Hoc Modules and Gather Facts

Lab Objective

The objective of this lab is to demonstrate Ansible's ad-hoc mode, as well as how `gather_facts` works.

Before working with playbooks, a sensible place to start is using ad-hoc mode. An ad-hoc command is something that you might type in to do something really quick, but don't want to save for later. As in, you want to take some quick actions, but write an entire playbook.

For example, if you wanted to power off all of your lab before a Bahamas vacation, you could execute a quick one-liner from your Ansible controller, without writing a playbook.

The issue with this approach is that you're not really producing a 'history' of your work. One of the benefits of keeping the code you run within playbooks, is there is a clear history of the commands and actions being imposed on your infrastructure.

Before starting this lab, you should review the following documentation.

Ansible Ad-Hoc commands: https://docs.ansible.com/ansible/latest/user_guide/intro_adhoc.html

Ansible Gather Facts: https://docs.ansible.com/ansible/latest/user_guide/intro_adhoc.html#gathering-facts

Ansible Setup Module: https://docs.ansible.com/ansible/latest/modules/setup_module.html

Procedure

1. Change the directory to `/home/student/`

   ```
   student@bchd:~$ cd ~
   ```

2. Download the tear-down script (you may already have this):

   ```
   student@bchd:~$ wget https://labs.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

3. Run the tear-down script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

4. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://labs.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

5. Run the setup script for this lab. This may take 2 or 3 minutes to run. Be patient.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

6. To ssh into these machines without a password, we will need to copy over a key.

   ```
   student@bchd:~$ ssh-copy-id bender@10.10.2.3
   ```

7. Type `alta3` when prompted for a password.

8. To ssh into these machines without a password, we will need to copy over a key.

   ```
   student@bchd:~$ ssh-copy-id fry@10.10.2.4
   ```

9. Type `alta3` when prompted for a password.

10. To ssh into these machines without a password, we will need to copy over a key.

    ```
    student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
    ```

11. Type `alta3` when prompted for a password.

12. The fourth machine, `farnsworth` has some issues with copying his key. Therefore, we will connect to this machine with a password. To enable the connection to `farnsworth` with SSH, we'll need to augment the controller with a python package called `sshpass`.

```
student@bchd:~$ sudo apt install sshpass -y
```

13. With ansible, we can target both hosts at once if we describe those hosts within an inventory. Let's stay organized. Create a directory structure, `~/ans/inv/dev/`. Within this folder, we can keep track of the inventory for our development testbeds.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

14. Edit your inventory file. Convention says to call this file `hosts`, but we can call it anything we want.

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

15. Add the following three lines to your `hosts` file. It is okay if you have other ones, inventories may contain multiple groups. In the below example, we named our hosts `h1` and `h2`. The names we apply to hosts within our ansible inventory can be anything we want, as the management IP is defined by `ansible_host`.

```
[research]
h1    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
h2    ansible_host=10.10.2.4 ansible_user=fry      ansible_python_interpreter=/usr/bin/python3
```

16. Save and exit with `:wq`

17. We're about to use the ping module. Try reading about it. This is **not** an ICMP ping, it is an Ansible ping. It tests for connectivity to the remote hosts.

```
student@bchd:~$ ansible-doc ping
```

18. Scroll with the arrow keys. One you are done reading, quit by pressing `q`

19. This module creates a full ssh session to the remote host and runs a very simple python script. If SSH is successful, and the python code can be executed, then we'll see a 'ping and pong' response.

```
student@bchd:~$ ansible research -i ~/ans/inv/dev/hosts -m ping
```

20. Default behavior is that each time a new connection is made, the `~/.ssh/known_hosts` file is checked for a key matching the system we are connecting to. If the key is **not** present, the user is asked if the connection should still continue. If the user indicates `yes`, then the key should be written into the `~/.ssh/known_hosts` file. If that file does not exist, then it will be created. However, this default behavior can be overridden by using the `~/.ssh/config` file. Examine that file now.

```
student@bchd:~$ cat ~/.ssh/config
```

21. Configuration entries for hosts can be on a host-by-host basis, or via a wildcarded, such as `10.*` The option, `StrictHostKeyChecking no` will deactivate the user ever being prompted to accept a key when connecting to a machine for the first time. If you'd like to deactivate this headache for the rest of the course, make sure an entry such as the one the follows is within your `~/.ssh/config` file.

```
Host 10.*
    StrictHostKeyChecking no
    UserKnownHostsFile=/dev/null
```

22. If you ever want to learn more about your connection, you can use the verbosity flags command. The highest level of verbosity is achieved with `-vvvvv`. Try that now.

```
student@bchd:~$ ansible research -i ~/ans/inv/dev/hosts -m ping -vvvvv
```

23. Next, we'll experiment with the setup module, which is also know as *Gathering Facts*, and controlled in playbooks with the keyword, `gather_facts`. By default, Ansible runs this fact finding module on each host before running other tasks. All of the returned values are available as variables in your playbooks **very handy!** Read about this module here: https://docs.ansible.com/ansible/latest/modules/setup_module.html

24. Let's gather facts about a single node `h2`.

```
student@bchd:~$ ansible h2 -i ~/ans/inv/dev/hosts -m setup
```

25. That created a lot of output. Run the command again and pass the output to the less command for scrolling enabled inspection.

```
student@bchd:~$ ansible h2 -i ~/ans/inv/dev/hosts -m setup | less
```

26. Study the structure of the facts.
    - `ansible_all_ipv4_addresses` - followed by information.
    - `ansible_all_ipv6_addresses` - followed by information.
    - `ansible_date_time` - followed by a lot of dictionaries.
    - I wonder if we can access specific facts ONLY? You bet! Let us learn about the `filter` parameter.
27. Type `:q` to quit less.
28. Using filters is an 'easier' way to limit returned information. Issue the command to show how many processor cores are running with, `"filter=ansible_processor_cores"`

    ```
    student@bchd:~$
    ansible research -i ~/ans/inv/dev/hosts -m setup -a "filter=ansible_processor_cores"
    ```
29. See if we can just list **ansible_date_time**.

    ```
    student@bchd:~$ ansible research -i ~/ans/inv/dev/hosts -m setup -a "filter=ansible_date_time"
    ```
30. How about listing the ipv4 default address. If this command does not return anything, it is likely because the target host has a recent update. For example, in March, Ubuntu updated to 18.04. At that time, they changed the way IP address information is displayed. As a result, there has been an issue with this ansible fact failing to populate. If it does work, then it is because ansible addressed this tiny bug. We keep this in the lab because it is always interesting when you find new bugs and see how long it takes a vendor to come out with a solution.

    ```
    student@bchd:~$ ansible research -i ~/ans/inv/dev/hosts -m setup -a "filter=ansible_all_ipv4_addresses"
    ```
31. Hmmm, so we see how to filter for specific stuff, but how are we to know that ***ansible_all_ipv4_addresses*** is a proper filter key? Well, you have to go looking for it.

    ```
    student@bchd:~$ ansible h2 -i ~/ans/inv/dev/hosts -m setup > ~/setup.output
    ```
32. Check out that file we just created.

    ```
    student@bchd:~$ less ~/setup.output
    ```
33. Try answering the following questions:
    - What distribution is the remote linux? What version is it?
    - What version on python is installed on the remote host?
    - How much memory (MB) are on the remote host? How much is free?
    - How many virtual CPUs are available on the remote host?
    - What user shell was used on the remote system?
34. Quit with the escape key, and then, `q`
35. Tear down your lab. This will help setup for a clean environment during the next, and subsequent labs.

    ```
    student@bchd:~$ bash max-teardown.sh
    ```
36. That's it for this lab. If you tracking your work on GitHub, perform the following operations.
    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "ad hoc and gather facts"`
    - `git push origin main`
    - Type in username & password

8. Running your first Playbook

Lab Objective

In this lab, you will run your first Ansible playbook. Running a playbook uses the **ansible-playbook** command, whereas running ad-hoc commands use the **ansible** command. Playbooks are way better for lots of reasons, not the least of which is that they can be version controlled. Best way to learn is to dive in, so Let's get started!

Documentation regarding building Ansible Playbooks can be found here:
https://docs.ansible.com/ansible/latest/user_guide/playbooks.html

Another useful resource is the list of Ansible Keywords. Keywords are used to build playbook logic:
https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html

Procedure

1. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

   ```
   student@bchd:~$ wget https://labs.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

3. Run the teardown script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

4. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://labs.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

5. Run the setup script for this lab.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id bender@10.10.2.3
   ```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id fry@10.10.2.4
   ```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
   ```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

    ```
    student@bchd:~$ mkdir -p ~/ans/inv/dev/
    ```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

    ```
    student@bchd:~$ vim ~/ans/inv/dev/hosts
    ```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit with `:wq`

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. This lab uses the 'apt' module. If you're a Red Hat user, than you know of a nearly identical program called, 'yum'. Both of these programs are analogous to the Google Play Store app or iTunes app. All of these are clients that allow users to install pre-built software. Read the documentation on the 'apt' module.

```
student@bchd:~$ ansible-doc apt
```

16. Press `q` to quit.

17. Use vim to create a file called, `~/ans/playbook-apt.yml`

```
student@bchd:~$ vim ~/ans/playbook-apt.yml
```

18. The following playbook will let us SSH to the target machine, and leverage control of the aptitude tool (apt) via Python (the module takes care of this). All we need to do is pass a minimum number of arguments, and see that 'figlet' is installed (a program that lets us make banners).

```
---
- name: Playbook - Install figlet     # providing meta-data
  hosts: planetexpress:!farnsworth    # resolved in our "inventory" (no farnsworth)
  gather_facts: yes       # default is yes
  connection: ssh         # use the connection plugin type, "ssh" (default)

  tasks:

  # first task
  - name: using apt to install figlet  # providing meta-data for task
    apt:                               # ansible module (apt.py)
        name: figlet                   # name parameter - provides name of the app
        state: present                 # state param - present / absent / latest / etc.
    become: yes                        # run as "sudo"
```

19. Save and exit with `:wq`

20. Answer the following questions about our new playbook:

    - **What are the keywords being used here?**
        - *hosts, tasks, name, connection, and become*
    - **How did you know these were keywords?**
        - *Check the link to ansible keywords at the top of this lab. Then, look at the keywords under **play** and **tasks***
    - **Why is farnsworth being removed?**
        - *Farnsworth is using a CentOS operating system, whereas the other hosts are Ubuntu. CentOS operating systems do not use the apt package installer (they use "yum"). Therefore, including host farnsworth would cause a failure. We will explore other solutions to this problem later on in the course.*

21. Run the playbook and examine the output:

```
student@bchd:~$ ansible-playbook ~/ans/playbook-apt.yml -i ~/ans/inv/dev/
```

22. Confirm that the remote machines now have 'figlet' installed.

```
student@bchd:~$ ssh bender@10.10.2.3
```

23. Type the word `figlet` to ensure figlet is installed.

```
bender@bender:~$ figlet
```

24. Great! Now create a banner by typing something:

```
my ascii text banner
```

25. Exit by pressing: `ctrl` + `c`

26. Exit the bender machine

```
bender@bender:~$ exit
```

27. Run the playbook again and examine the output. Notice that this time, ansible didn't feel the need to install figlet *again*. This is because it was already installed.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-apt.yml -i ~/ans/inv/dev/
```

28. Use vim to create another file called, `playbook-apt-absent.yml`

```
student@bchd:~$ vim playbook-apt-absent.yml
```

29. Copy the following into the playbook. Notice that this one has changed the `state` to `absent`.

```
---
- name: Playbook - Install figlet     # providing meta-data
  hosts: planetexpress:!farnsworth    # resolved in our "inventory" (no farnsworth)
  gather_facts: yes       # default is yes
  connection: ssh         # use the connection plugin type, "ssh" (default)

  tasks:

  # first task
  - name: using apt to install figlet  # providing meta-data for task
    apt:                               # ansible module (apt.py)
        name: figlet                   # name parameter - provides name of the app
        state: present                 # state param - present / absent / latest / etc.
    become: yes                        # run as "sudo"
```

30. Save and exit with `:wq`

31. Run the playbook and examine the output.

```
student@bchd:~$ ansible-playbook playbook-apt-absent.yml -i ~/ans/inv/dev/
```

32. This should remove the application `figlet` on the targeted host machines. SSH to one or more of the targets to be sure.

```
student@bchd:~$ ssh bender@10.10.2.3
```

33. Type the word `figlet` to ensure figlet is removed.

```
bender@bender:~$ figlet
```

34. Exit the bender machine

```
bender@bender:~$ exit
```

35. Run the playbook again and examine the output. Notice that this time, ansible didn't feel the need to remove figlet *again*. This is because it was already removed.

```
student@bchd:~$ ansible-playbook playbook-apt-absent.yml -i ~/ans/inv/dev/
```

36. Run the tear down script to stay organized for future labs.

```
student@bchd:~$ bash max-teardown.sh
```

37. Great job! The take away from this lab is the basics of a playbook, along with ansible's predictable nature.

38. That's it for this lab. If you tracking your work on GitHub, perform the following operations.
    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "First playbook"`
    - `git push origin`
    - Type in username & password

9. Debug, Loops, and YAML Lists

Lab Objective

In this lab, you'll work with the debug module, variable recall, loops, and YAML lists. The **debug** module allows us to print to the screen. Recalling variables is useful, as it allows us to make playbooks more dynamic and reusable. Ansible enables the creation of loops, which allows for writing less code. At this time, Ansible would like you to begin updating the old keyword for creating loops, `with_*`, typically `with_items`, to the new keyword `loop`.

A YAML list is simply a method of expressing a series of 'things'; no different than the grocery list you create before you go shopping.

Read about the 3rd and latest edition of the YAML spec here:
https://yaml.org/spec/1.2/spec.html

Curious what keywords are supported in playbooks? A list of Ansible keywords is available here: https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html

Ansible documentation on looping is available here:
https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html

Procedure

1. We'll run a few playbooks against our localhost, so no need to set anything up in this lab.
2. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

3. Let's begin by making a directory to work in.

   ```
   student@bchd:~$ mkdir ~/ans/
   ```

4. Use vim to create a new playbook called, `~/ans/playbook-loop01.yml`

   ```
   student@bchd:~$ vim ~/ans/playbook-loop01.yml
   ```

5. Create the following playbook. The code below invokes the Ansible `debug` module, which allows us to print a message to the screen. The name 'debug' was deliberately chosen over something like, 'print'. The reason is that Ansible is for placing state on target machines, not for displaying output on the screen. However, sometimes it is critical to run a playbook, and get eyes on the current value of a variable during design or refinement (debugging) stages of playbook creation.

```
---
- name: Intro to looping
  hosts: localhost
  gather_facts: no

  vars:   # vars available to all tasks
    ethList:
      - "eth0"
      - "eth1"
      - "eth2"
      - "eth3"

  tasks:
    - name: Demo a tiny loop
      debug:
        # item is always the var mapped to loop
        msg: "The debug module allows us to display to the screen: {{ item }}"
      loop: "{{ ethList }}"

    - name: Demo a second tiny loop
      debug:
        # item is always the var mapped to loop
        msg: "All modules can also have LOCAL var information: {{ item }}"
      loop: "{{ myOtherList }}"
      vars:
        myOtherList:
          - "192.168.70.1"
          - "10.10.1.1"
```

6. Save and exit.

7. Run the playbook. It will only execute against our localhost (controller). We are not connecting to any remote hosts with this playbook, therefore, we do not need to describe an inventory file.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-loop01.yml
```

8. Examine the results of the playbook. Take note on how the loops caused the debug modules to run multiple times.

9. Variables can be mapped in external files, and users may even be prompted for variables at runtime. For now, let's explore a concept called, "Extra Variables". Before we go any further, check out this list: https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable revealing that, 'extra variables' get highest precedence.

10. Issue the follow command, and remap the value of `ethList` via an extra variable assignment at the CLI.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-loop01.yml -e ethList=["1.1.1.1", "2.2.2.2"]
```

11. Hmm, **it looks like that caused an error!** When passing complex variables, such as arrays or dictionaries, it may be wise to pass them in a file. That's because variable passed at the CLI is interpreted as text-strings. That is to say, our square-brackets are not being recognized. Try creating an extra file where we can remap `ethList`.

12. Create a new directory where we can store our variable information. Remember, it pays to stay organized.

```
student@bchd:~$ mkdir -p ~/ans/vars/
```

13. Now create our variable file.

```
student@bchd:~$ vim ~/ans/vars/pblist01.json
```

14. Create the following file.

```
{"ethList":["1.1.1.1","2.2.2.2"]}
```

15. Save and exit.

16. Now try rerunning the command. The `@` is required when referencing an extra-vars file containing json or yaml.

```
student@bchd:~$
ansible-playbook ~/ans/playbook-loop01.yml --extra-vars "@~/ans/vars/pblist01.json"
```

17. On this run of the playbook, the value of `ethList` should be over-ridden by the list items `1.1.1.1` and `2.2.2.2`.

18. Create a new playbook.

```
student@bchd:~$ vim ~/ans/playbook-loop02.yml
```

19. Sometimes it is important to 'pause' a loop between cycles (sometimes useful when troubleshooting racing conditions in network gear and clouds). To control time (in seconds) between execution of each item in a task loop, use the pause directive with `loop_control`. As the name implies, `loop_control` provides finer management control when working with loops.

20. Create a new playbook.

```
- name: Loop with 3 second pause between runs
  hosts: localhost
  gather_facts: no

  tasks:

  - name: Run a debug with 3 second pauses
    debug:
        var: item
    loop:
      - server1
      - server2
      - server3
      - server4
    loop_control:
      pause: 3  # in seconds
```

21. Save and exit.

22. Run the new playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-loop02.yml
```

23. Create a new playbook, `~/ans/playbook-loop03.yml'

```
student@bchd:~$ vim ~/ans/playbook-loop03.yml
```

24. Sometimes you will want to work across more complex data structure, like a list of dictionaries. In those cases, simply use `item.key`, where `key` is replaced by the key from the dictionary you are referencing. To practice this, make your playbook look like the following:

```
- name: Working with simple dictionaries
  hosts: localhost
  gather_facts: no

  tasks:

  - name: Loop across complex data structures
    debug:
      msg: "{{ item.name }} belongs in the {{ item.groups }} aisle"
    loop:
      - { name: 'orange', groups: 'fruit' }
      - { name: 'lemon', groups: 'fruit' }
      - { name: 'cookie', groups: 'snack' }
      - { name: 'carrot', groups: 'vegetable' }
```

25. Save and exit.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-loop03.yml
```

26. Create a new playbook, `~/ans/playbook-loop04.yml`

```
student@bchd:~$ vim ~/ans/playbook-loop04.yml
```

27. In this playbook, we'll examine the, 'old' style of looping called, `with_items`. At the time of writing, this is still supported, but will soon be deprecated (begin to lose favor). Therefore, you should **not** be writing any new playbooks with this keyword, but still need to understand what it does.

```
---
 - name: Looping old-style with_items
   hosts: localhost
   gather_facts: no

   tasks:
    - name: "Learning about the debug module"
      debug:
        msg: "Debug module allows us to print to the screen --> {{ item }}"
      with_items:
      - "255.255.255.0"
      - "192.168.0.1"
      - "192.168.0.2"
```

28. Save and exit.

29. Run the new playbook. It should work very similarly to the last playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-loop04.yml
```

30. Ansible playbooks are written (coded) using keywords. A list of Ansible keywords can be found here: http://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html

31. Answer the following questions:

    - **Q: What are the play-level keywords being used here?**
        - A: name, hosts, gather_facts, tasks
    - **Q: How did you know these were play keywords?**
        - A: Because I reviewed and bookmarked the playbook keywords link in the previous step.

32. If you are tracking your code, issue the following commands to commit to Git Hub.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "YAML and Loops"`
    - `git push origin master`
    - Type in username & password

10. When Conditionals, YAML Dictionaries, and Jinja

Lab Objective

Playbooks are always written in YAML Ain't Markup Language, or YAML. YAML attempts to be a human-readable version of JSON, and therefore, be easier to write. If nothing else, the standard relaxes the requirement for the number of punctuating marks, and replaces those concepts with simple white spacing.

Playbooks can also be enhanced by *conditionals*. The idea behind a *conditional* is classically handled by the `if` keyword in most languages. Within Ansible, it is handled by the `when` keyword.

It will be helpful to review the YAML specification available here:
https://yaml.org/spec/1.2/spec.html

Review all available Ansible keywords here:
https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html

Jina2 Filters are available here:
https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. Download the teardown script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the teardown script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. Ping the each of the below lab hosts.

```
student@bchd:~$ ping -c 1 10.10.2.3
student@bchd:~$ ping -c 1 10.10.2.4
student@bchd:~$ ping -c 1 10.10.2.5
student@bchd:~$ ping -c 1 10.10.2.6
```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id bender@10.10.2.3
```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id fry@10.10.2.4
```

9. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
```

10. Farnsworth has some issues with copying his key, so we'll connect with a password.

11. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

12. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

13. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

14. Save and exit with `:wq`

15. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

16. Ensure you have the `~/ans` directory to work in.

```
student@bchd:~$ mkdir ~/ans/
```

17. Startup an editor and create a file called `~/ans/playbook-dictrecall01.yml`

```
student@bchd:~$ vim ~/ans/playbook-dictrecall01.yml
```

18. Create the following playbook:

```
---
 - name: Intro to Dictionaries
   hosts: localhost
   gather_facts: no

   vars:
     configA: exampleA.cfg    # var mapped to a str
     configB: exampleB.cfg    # var mapped to a str
     licensetypes:            # var mapped to a list of strs
       opentypes:
         - apache2.0
         - gnu public license (GPL)
         - mit license
         - mozilla public license
       paytypes:
         - windows
         - redhat
         - apple

   tasks:
    - name: Recall a key value pair mapping
      debug:
        var: configA        # returns exampleA.cfg

    - name: Recall a key value mapping where the value is a list
      debug:
        var: licensetypes.opentypes   # returns a list of license types

    - name: Display the MIT License
      debug:
        var: licensetypes.opentypes[2]   # returns mit license

    - name: Display Windows
      debug:
        var: licensetypes.paytypes[0]   # returns windows
```

19. Save and exit with `:wq`

20. Run the new playbook until you are certain you understand how we are recalling variable values from within dictionary.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dictrecall01.yml
```

21. Create a new playbook file called, `~/ans/playbook-yaml-tf01.yml`

```
student@bchd:~$ vim ~/ans/playbook-yaml-tf01.yml
```

22. True and False conditionals can be a bit tricky. If you ever pass values in from the CLI, the variables will *always* be read as strings. Strings exist, so they always test True. This is weird, but if Ansible thinks False is actually "False", then it will test as True! The solution to this, is to place a 'pipe', followed by the `bool` Jinja2 filter after the expression. More on this later, but if you're dying to know, Jinja2 is a templating language used for performing some on-the-fly data translations. Read about Jinja2 filters in Ansible here: https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html

23. Create the following playbook:

```
---
- name: Intro to T and F conditionals
  hosts: localhost
  gather_facts: no

  vars:
    bool_var: True  # This will cause "
    msg_var: "TIME TO PANIC!"

  tasks:
  - name:  Runs when bool_var is true
    debug:
      msg: "{{ msg_var }}"
    when: bool_var|bool

  - name: runs when bool_var is false
    debug:
      msg: "DONT PANIC!"
    when: not bool_var|bool
```

24. Save and exit with `:wq`

25. Try running your playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-yaml-tf01.yml
```

26. Pass in a new value for the `bool_var`. Since we are passing the replacement value as an "extra-var", this value always wins. Try it out!

```
student@bchd:~$ ansible-playbook ~/ans/playbook-yaml-tf01.yml -e bool_var=False
```

27. As expected, the extra variable does override the playbook var, as extra variables have the highest precedence.

28. Create a new playbook to further explore jinja filters.

```
student@bchd:~$ vim ~/ans/playbook-jinja-filters-03.yml
```

29. Jinja filters can be used to force more than "just" boolean data types. A common operation might be forcing string data to be an integer ("22" to become 22). In the following playbook, the value of `ansible_facts.lsb.major_release` is forced to be 18 as opposed to, "18".

```
---
- name: Using Jinja within our Playbook
  hosts: planetexpress  # defined within the inventory
  connection: ssh
  gather_facts: true    # required for ansible_facts

  tasks:

  - name: Display all ansible_facts
    debug:
      var: ansible_facts
      verbosity: 2        # this will ONLY run if you include -vv

  - name: What does ansible_facts.os_family contain?
    debug:
      var: ansible_facts.os_family
      verbosity: 1        # this will ONLY run if you include -v

  - name: What does ansible_facts.lsb contain?
    debug:
      var: ansible_facts.lsb
      verbosity: 1        # this will ONLY run if you include -v

  - name: Some task that runs ONLY on hosts that match our conditional
    debug:
      msg: "I am a Debian host that is running ATLEAST ver 18 of my OS"
    when: ansible_facts.os_family == "Debian" and ansible_facts.lsb.major_release | int >= 18
```

30. Save and exit with `:wq`

31. Run the playbook with two `-vv` flags. This will trigger ALL the debug statements to run. Placing verbosity parameters on debug is a common technique for suppressing output that was required during the development process.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-jinja-filters-03.yml -vv
```

32. That was a bit overwhelming. Try running it with just a single `-v` flag.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-jinja-filters-03.yml -v
```

33. Finally, run with no flags.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-jinja-filters-03.yml
```

34. Answer the following questions:

    - **Q: What is the Jinja filter within the last playbook?**
        - *A: int*
    - **Q: What does this Jinja filter do?**
        - *A: Forces a data type to be an integer*
    - **Q: Why was this necessary?**
        - *A: To write an arithmetic test such as,* `>= 18`
    - **Q: Do all Jinja filters "force" data types?**
        - *A: No. The origins of Jinja are in HTML templating, so many filters focus on modifying string data (example: capitalization or lower-casing)*
    - **Q: Why are there no "{{ }}" statements within the "when: " conditionals?**
        - *A: Ansible drops the requirement for the double-quote, mustache-bracket syntax within "when: " conditionals*

35. If you are tracking your code, issue the following commands to commit your code to GitHub.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "Forcing data types with Jinja filters"`

    - `git push origin`
    - Type in username & password

Gennady Frid
gfrid@bloomberg.net

11. ansible.cfg setup

Lab Objective

Demonstrate the ability of a user to configure Ansible with a config file to simplify cli arguments. Before ansible runs, it will read the `ansible.cfg` file. In this file, we can place global configuration settings, that allow us to "type less" when executing an Ansible playbook. This file also can help us leverage how Ansible runs.

The file `ansible.cfg` will be searched for in the following order. If a file is found, Ansible will ignore any remaining sources:

- The environmental variable `ANSIBLE_CONFIG` *(environment variable if set)*
- The file `ansible.cfg` *(in the current directory)*
- `~/.ansible.cfg` *(in the home directory)*
- `/etc/ansible/ansible.cfg` *(last location checked)*

For a detailed version of `ansible.cfg`, refer to the following:
https://github.com/ansible/ansible/blob/devel/examples/ansible.cfg

The latest documentation outlining Configuration of Ansible can be found here:
https://docs.ansible.com/ansible/latest/installation_guide/intro_configuration.html

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. Download the teardown script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the teardown script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. Ping the each of the below lab hosts.

```
student@bchd:~$ ping -c 1 10.10.2.3
student@bchd:~$ ping -c 1 10.10.2.4
student@bchd:~$ ping -c 1 10.10.2.5
student@bchd:~$ ping -c 1 10.10.2.6
```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id bender@10.10.2.3
```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id fry@10.10.2.4
```

9. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
```

10. Farnsworth has some issues with copying his key, so we'll connect with a password.

11. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

12. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

13. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

14. Save and exit with `:wq`

15. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

16. Create a new playbook, `~/ans/playbookcmd.yml`.

```
student@bchd:~$ vim ~/ans/playbookcmd.yml
```

17. Create the following simple playbook.

```
---
- name: A simple playbook to test ansible.cfg
  hosts: planetexpress
  gather_facts: yes

  tasks:

  - name: Issue a date cmd on each remote host
    command: date
```

18. Save and exit with `:wq`. The above playbook is unimpressive, but that is alright. We just want something to test how we are connecting to our remote hosts. After running the gather_facts module (aka setup module), the playbook will issue, `date` on each remote host machine.

19. Try running your playbook. **Without the inventory flag, this should fail.**

```
student@bchd:~$ ansible-playbook ~/ans/playbookcmd.yml
```

20. We didn't supply an inventory. Instead, let's write that information into a file called, `ansible.cfg`. This file contains default values that control how Ansible runs. It can live in a number of locations. Local to the playbook, within your home directory, or in, `/etc/ansible/`. Let's put ours in the home directory. The only caveat is that we need to make it a hidden file (put a dot before the name of the file).

```
student@bchd:~$ vim ~/.ansible.cfg
```

21. Ensure your file has the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

22. Save and exit.

23. Run your playbook. The `ansible.cfg` file should control location of the inventory. The run should succeed.

```
student@bchd:~$ ansible-playbook ~/ans/playbookcmd.yml
```

24. To see defaults Ansible currently has set, run the following. It will dump all of Ansible's current configuration settings. to **quit**, just press the letter `q`

```
student@bchd:~$ ansible-config dump
```

> To exit, press q

25. Great job! This lab is complete. Feel free to leave your `~/.ansible.cfg` file in place. We'll continue to augment this file as we work through our understanding of Ansible.

26. **CHALLENGE 01 (OPTIONAL)** - Finding the Ansible output a little boring? Turn on `cowsay`! The application `cowsay` surround standard out with fun ASCII art 'talking' the output. It is likely you have seen `cowsay`, and not known what it was. So, let's see Ansible use it. Start by installing `cowsay` with the apt utility.

```
student@bchd:~$ sudo apt install cowsay -y
```

27. Turn on cowsay by adding the following line to your `~/.ansible.cfg` file.

```
student@bchd:~$ vim ~/.ansible.cfg
```

28. To **turn on cowsay** set `nocows=0` within `~/.ansible.cfg` (under the heading `[defaults]`).

```
[defaults]
## TURN ON COWSAY
nocows = 0
```

29. For Ansible, the default is to use `cowsay` if it is installed, so try running a playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbookcmd.yml
```

30. Why you don't say, a cowsay! You can turn off cowsay by adding the following line to your `~/.ansible.cfg` file.

```
student@bchd:~$ vim ~/.ansible.cfg
```

31. To **turn off cowsay** set `nocows=1` within `~/.ansible.cfg` (under the heading `[defaults]`).

```
[defaults]
## TURN OFF COWSAY
nocows = 1
```

32. To **turn on cowsay and other random ascii art** set `nocows = 0` and `cow_selection = random` as shown below (under the heading `[defaults]`).

```
[defaults]
## TURN ON COWSAY
nocows = 0
## TURN ON COWS AND OTHER RANDOM ASCII ART
cow_selection = random
```

33. Save and exit when you've decided if cowsay is for you, or not!

12. Building Playbooks - Bootstrap with raw, group and user Modules

Lab Objective

The objective of this lab is to design a multi-step playbook that can perform some work. We'll start by design a playbook that might bootstrap the Indy host with the latest version of Python 3.x. Once we have Python3 installed, we can try leveraging some basic Ansible modules that we might explore in greater detail in later labs.

Read about the `raw` module here: https://docs.ansible.com/ansible/latest/modules/raw_module.html

Read about the `user` module here: https://docs.ansible.com/ansible/latest/modules/user_module.html

Read about the `group` module here: https://docs.ansible.com/ansible/latest/modules/group_module.html

Procedure

1. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

2. Download the tear-down script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

3. Run the tear-down script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

4. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/indy-setup.sh
   ```

5. Run the setup script for this lab.

   ```
   student@bchd:~$ bash indy-setup.sh
   ```

6. Ensure you have a place to work.

   ```
   student@bchd:~$ mkdir -p ~/ans/inv/dev/
   ```

7. Ping the new indy host below.

   ```
   student@bchd:~$ ping -c 1 10.10.2.2
   ```

8. SSH to the lab host *indy* at *10.10.2.2*.

   ```
   student@bchd:~$ ssh indy@10.10.2.2
   ```

9. Python is a requirement for Ansible to run against remote hosts. Determine if Python 2.7+ is installed.

   ```
   indy@indy:~$ python --version
   ```

10. If Python 2.7+ did get installed, remove it with the following command:

    ```
    indy@indy:~$ sudo apt autoremove python -y
    ```

11. Determine if Python 3.x is installed.

    ```
    indy@indy:~$ python3 --version
    ```

12. If Python3 did get installed, remove it with the following command:

    ```
    indy@indy:~$ sudo apt autoremove python3 -y
    ```

13. As we learned in previous labs, for some reason indy is an ophidiophobe and doesn't have Python installed on his system. Before we can use Ansible against this machine, we'll have to fix this.

14. Ensure indy has sudo privileges.

```
indy@indy:~$ sudo whoami
```

15. Exit indy machine.

```
indy@indy:~$ exit
```

16. Great! Now create our inventory file:

```
student@bchd:~$ vim ~/ans/inv/dev/jones
```

17. Edit your host file as shown below.

```
[raiders]
indy      ansible_host=10.10.2.2 ansible_user=indy ansible_ssh_pass=alta3
```

18. Save and exit.

19. In order to use a password from an inventory file, we need to install `sshpass`

```
student@bchd:~$ sudo apt install sshpass
```

20. There's no Python on indy, so the following command will FAIL.

```
student@bchd:~$ ansible -m ping raiders -i ~/ans/inv/dev/jones
```

21. Let's try using an ad-hoc raw command to issue the argument `apt install python -y`. The below command can be read, "Ansible, use the raw module to target the group raiders, a group that can be defined by looking to the directory `~/ans/inv/dev/jones`, and to do this, 'become' sudo.

```
student@bchd:~$ ansible -m raw raiders -i ~/ans/inv/dev/jones -a "apt install python3 -y" -b
```

22. The above command will take a bit to execute. When it finishes, we should have no problem using the ping module to contact the `indy` machine.

```
student@bchd:~$ ansible -m ping raiders -i ~/ans/inv/dev/jones
```

23. If everything worked out, the above command should return a green 'pong' to your 'ping'. Remember, the ping module has nothing to do with ICMP, and in that way, is poorly named. Think of it more as a sanity test that checks if the target host(s) can be contacted via SSH, logged into using the creds provided, and if Python is installed.

24. Create a playbook file called `~/ans/bootstrap-new-user.yml`

```
student@bchd:~$ vim ~/ans/bootstrap-new-user.yml
```

25. Create the following playbook. Notice in the playbook, we still include the `raw` module, despite having installed Python on the indy machine. This is by design, so you can observe the 'lack of coolness' provided by `raw`. No matter how many times you run the playbook, the `raw` module will **always** execute. This is unlike other modules that only execute if they need to. This 'always issue behavior' is undesirable, as it could lead to unanticipated and unexpected results. A lack of predictability is never a good thing when working with automation!

```
---
- name: bootstrapping and new user
  hosts: raiders
  connection: ssh
  gather_facts: False

  tasks:

  # install Python with ansible
  # the raw module will ALWAYS issue your command
  # no idempotence here!
  # therefore, use it sparingly!!
  - name: "Bootstrap python install"
    raw: "apt install -y python3-pip"
    become: yes

  # create a new group
  - name: Create the new group sandbox
    group:
      name: sandbox
      state: present
    become: yes

  # create new user on target system
  - name: Add user 'exampleuser' with group of 'sandbox'
    user:
      name: exampleuser
      comment: "Example User"
      # password: alta3      # pass as hash
      state: present
      shell: /bin/bash       # Defaults to /bin/bash
      system: no             # Defaults to no
      createhome: yes        # Defaults to yes
      # home: /home/exampleuser  # Defaults to /home/<username>
      group: sandbox
    become: yes
```

26. Save and exit with `:wq`

27. Did you notice that setting a password requires a hash? If you'd like, you can run the following command and use the resulting hash.

```
student@bchd:~$ mkpasswd --method=sha-512
```

28. It is possible that the previous command requires the following install to work:

```
student@bchd:~$ sudo apt-get install -y whois
```

29. Try running your playbook.

```
student@bchd:~$ ansible-playbook ~/ans/bootstrap-new-user.yml -i ~/ans/inv/dev/jones
```

30. If you'd like, you can try running it a few more times as well. Notice that no matter how many times you run the playbook, the raw module **always** displays a change. That is a reflection of how the raw module is architected; without any Python 'intelligence' to support the raw module, the best the module can offer is an SSH command, which, if nothing else, would be reflected as a state change within your history.

31. SSH to indy.

```
student@bchd:~$ ssh indy@10.10.2.2
```

32. Determine if the new user creation playbook was successful.

```
indy@indy:~$ sudo su exampleuser
```

33. The prompt should change to something like `exampleuser@indy:/home/indy~$`. Change to your home directory.

```
exampleuser@indy:/home/indy~$ cd
```

34. Now print working directory to ensure you are in `/home/exampleuser`

```
exampleuser@indy~:$ pwd
```

35. It worked. Exit out of here.

```
exampleuser@indy~:$ exit
```

36. Exit from indy.

```
indy@indy:~$ exit
```

37. Let's write a playbook that remains idempotent. With just a little extra work, we can create a playbook that doesn't show "changed" each time.

```
student@bchd:~$ vim ~/ans/bootstrap-new-user-idempotent.yml
```

38. Create the following playbook.

```
---
- name: bootstrapping and new user
  hosts: raiders
  connection: ssh
  gather_facts: False

  tasks:

  # create idempotence where there is none!
  # This task will play nicely with the second task
  - name: "is python installed?"
    raw: which python3
    changed_when: false
    ignore_errors: true
    register: results

  # install Python with ansible
  # the raw module will ALWAYS issue your command
  # no idempotence here!
  # therefore, use it sparingly!!
  - name: "Bootstrap python install"
    raw: "apt install -y python3-pip"
    become: yes
    when: '"python3" not in results.stdout'

  # create a new group
  - name: Create the new group sandbox
    group:
      name: sandbox
      state: present
    become: yes

  # create new user on target system
  - name: Add user 'exampleuser' with group of 'sandbox'
    user:
      name: exampleuser
      comment: "Example User"
      # password: alta3      # pass as hash
      state: present
      shell: /bin/bash       # Defaults to /bin/bash
      system: no             # Defaults to no
      createhome: yes        # Defaults to yes
      # home: /home/exampleuser  # Defaults to /home/<username>
      group: sandbox
    become: yes
```

39. Save and exit with `:wq`

40. Run the new solution.

```
student@bchd:~$ ansible-playbook ~/ans/bootstrap-new-user-idempotent.yml -i ~/ans/inv/dev/jones
```

41. Notice this time, raw did not run. Let's create a playbook that removes the user and group we just added to indy. This is a basic introduction to the concept of creating "roll-backs" (undo routines).

```
student@bchd:~$ vim ~/ans/remove-new-user.yml
```

42. Write the following tiny playbook:

```
---
- name: bootstrapping and new user
  hosts: raiders
  connection: ssh
  gather_facts: False

  tasks:
  # Remove the user 'exampleuser'
  - name: remove a user
    user:
     name: exampleuser
     state: absent
     remove: yes
    become: yes

  - name: Remove the group sandbox
    group:
      name: sandbox
      state: absent
    become: yes
```

43. Save and exit with `:wq`
44. Run the script.

```
student@bchd:~$ ansible-playbook ~/ans/remove-new-user.yml -i ~/ans/inv/dev/jones
```

45. SSH into indy and ensure you can no longer switch users to `exampleuser`

```
student@bchd:~$ ssh indy@10.10.2.2
```

46. The following command should no longer be successful.

```
indy@indy:~$ sudo su exampleuser
```

47. Escape the indy machine, "No time to argue!"

```
indy@indy:~$ exit
```

48. Run the tear-down script to clean up from this lab.

```
student@bchd:~$ bash max-teardown.sh
```

49. **Conclusions:**
    - Python is required on the target hosts if you want to run anything other than the `raw` module.
    - The `raw` module is a bit dull, it always registers a change.
    - OS agnostic modules `user` and `group` can be used to deploy or remove users and groups.
    - Some modules contain an `absent` parameter, which can make roll-backs easier.
50. Great job! You have finished this lab.

13. Ansible Module - mount

Lab Objective

The objective of this lab is to explore the Ansible `mount` module. This module may be used to target the localhost (controller) or remote linux hosts by controlling configured mount points in `/etc/fstab`. Using the `mount` module, we'll create a playbook that creates a "bind mound".

A bind mount is an alternate view of a directory tree. Classically, mounting creates a view of a storage device as a directory tree. A bind mount instead takes an existing directory tree and replicates it under a different point. The directories and files in the bind mount are the same as the original. Any modification on one side is immediately reflected on the other side, since the two views show the same data.

For example, after issuing the Linux command

`mount --bind /some/where /else/where` the directories /some/where and /else/where have the same content.

Unlike a hard link or symbolic link, a bind mount doesn't affect what is stored on the filesystem. It's a property of the live system.

If this is news to you, check out this article on bind mounts before completing this lab: https://unix.stackexchange.com/questions/198590/what-is-a-bind-mount/198591#198591

Read about the `mount` module here (it can do more than 'just' create bind mounts): https://docs.ansible.com/ansible/latest/collections/ansible/posix/mount_module.html

Procedure

1. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

2. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

   ```
   student@bchd:~$ mkdir -p ~/ans/inv/dev/
   ```

3. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/local.host`

   ```
   student@bchd:~$ vim ~/ans/inv/dev/local.host
   ```

4. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following). This entry creates a host called, `beachhead` within the group `storage`.

   ```
   [storage]
   beachhead ansible_host=localhost ansible_connection=local ansible_python_interpreter=/usr/bin/python3
   ```

5. Save and exit with `:wq`

6. Skim the documentation for the mount module:

   ```
   student@bchd:~$ ansible-doc mount
   ```

7. Use up and down to scroll. Press `q` to quit.

8. Skim the documentation for the file module:

   ```
   student@bchd:~$ ansible-doc file
   ```

9. Use up and down to scroll. Press `q` to quit.

10. Create a playbook that we can use to demonstrate the mount module, `playbook-mount.yml`.

    ```
    student@bchd:~$ vim ~/ans/playbook-mount.yml
    ```

11. Create the following playbook.

```
---
- name: Creating a bindmount
  hosts: storage        # group name from the inventory
  gather_facts: no      # no need to run setup (collect facts) on remote machines
  become: yes
  connection: local     # targeting our local machine, no need to SSH

  vars:
    here_there: present # defines the variable "here_there" with value "present"

  tasks:

    # Mount can create a mount point,
    # but so can we, with the 'file' module
    - name: Create the path to the mount point
      file:
              path: /home/student/bindmnt
              state: directory

    - name: Bind mount a volume
      mount:
              path: /home/student/bindmnt # path to the mount point
              src: /tmp                   # dev to be mounted on the path
              opts: bind
              fstype: none
              state: "{{ here_there }}"   # present == device will be configured in /etc/fstab
                                          # absent  == devices will be unmounted and removed from /etc/
fstab
                                          # mounted == device is mounted and configured in /etc/fstab.
                                          #     If mount point is not created, it will be created.
                                          # see documentation for other options
```

12. Save and exit with `:wq`
13. Next run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-mount.yml -i ~/ans/inv/dev/local.host
```

14. Take a peek at `/etc/fstab`. A new entry should be made in the file. However, the `state: present` directive does not trigger a mount.

```
student@bchd:~$ cat /etc/fstab
```

15. Take a look at the mount point, `~/bindmnt/`. You won't see any file yet, as we haven't actually mounted anything, just made an entry in `/etc/fstab`

```
student@bchd:~$ ls ~/bindmnt/
```

16. Run the playbook again, only this time pass an **extra variable** to set the value of `here_there` to `mounted`.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-mount.yml -i ~/ans/inv/dev/local.host -e here_there=mounted
```

17. The playbook should reflect a "change" on the task `Bind mount a volume`
18. Create a file within `/tmp/`

```
student@bchd:~$ touch /tmp/test.file
```

19. Check the contents of `~/bindmnt/`. The file `test.file` should be present (along with other files that existed in `/tmp/`)

```
student@bchd:~$ ls ~/bindmnt/
```

20. Check the contents of `/tmp/`. These contents should be the same as, `~/bindmnt/`. The file `test.file` should also be present.

```
student@bchd:~$ ls /tmp/
```

21. Cool. Remove your work.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-mount.yml -i ~/ans/inv/dev/local.host -e here_there=absent
```

22. Ensure the mount point `~/bindmnt/`, is missing.

```
student@bchd:~$ ls ~/bindmnt/
```

23. Answer the following questions:

    - **Q: What values are possible for "state" within the "mount" module?**
        - *A: mounted, unmounted, present, absent, and remounted.*
    - **Q: What variables have highest precedence?**
        - *A: The extra variables (those set at the CLI) always win precedence.*

24. Great job! You have completed this lab.

25. If you are tracking your code on GitHub, take a moment and perform a commit.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about the mount module"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

14. Ansible Module - shell

Lab Objective

The objective of this lab is to demonstrate two modules, the `shell` Ansible module, as well as the `file` module.

The shell module takes the command name followed by a list of space-delimited arguments. It is almost exactly like the command module but runs the command through a shell `/bin/sh` on the remote node.

The file module set attributes of files, symlinks or directories, alternatively, it remove files, symlinks or directories.

Documentation on the `shell` module can be found here:
http://docs.ansible.com/ansible/latest/shell_module.html

Documentation on the `file` module can be found here:
https://docs.ansible.com/ansible/latest/modules/file_module.html

Procedure

1. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

3. Run the teardown script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

4. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

5. Run the setup script for this lab.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id bender@10.10.2.3
   ```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id fry@10.10.2.4
   ```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
   ```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

    ```
    student@bchd:~$ mkdir -p ~/ans/inv/dev/
    ```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

    ```
    student@bchd:~$ vim ~/ans/inv/dev/hosts
    ```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit.

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

16. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

17. Save and exit with `:wq`

18. Use the `ansible-doc` command to review the documentation for the shell module.

```
student@bchd:~$ ansible-doc shell
```

19. Press `q` to quit.

20. Create a basic playbook file called `playbook-fileshell.yml`

```
student@bchd:~$ vim ~/ans/playbook-fileshell.yml
```

21. Replicate the following playbook:

```
---
- name: A simple file playbook
  hosts: planetexpress
  gather_facts: yes

  tasks:
   - name: make a directory
     file:
       path: ~/output/
       state: directory
```

22. Save and exit with `:wq`

```
student@bchd:~$ cat ~/ans/playbook-fileshell.yml
```

23. Next run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-fileshell.yml -i ~/ans/inv/dev/hosts
```

24. Now let's connect to the remote hosts with secure shell and run an `ls` command to see if the `~/output/` directory exists.

```
student@bchd:~$ ssh bender@10.10.2.3
```

25. Determine if the directory was created.

```
bender@bender:~$ ls
```

26. Now exit.

```
bender@bender:~$ exit
```

27. Check the other machines if you'd like. When you're satisfied it worked, edit the playbook once again.

```
student@bchd:~$ vim ~/ans/playbook-fileshell.yml
```

28. Make the following changes. Below we add the `shell` module which, unlike the `command` module, allows for wildcards, such as `< | > & << >>`.

```
---
- name: A simple file playbook
  hosts: planetexpress
  gather_facts: yes

  tasks:
   - name: make a directory
     file:
       path: ~/output/
       state: directory

   - name: use shell module with wildcarding
     shell: echo $HOME >> ~/output/path.txt
```

29. Save and exit. The `shell` command we added will take the contents of `$HOME` and push the contents into `~/output/path.txt`. In this example, the wildcard is the `>>`. This will not work if you try it with the `command` module.

30. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-fileshell.yml -i ~/ans/inv/dev/hosts
```

31. Now let's connect to the remote hosts with secure shell and see if our file appeared.

```
student@bchd:~$ ssh bender@10.10.2.3
```

32. Determine if the file in the directory was created.

```
bender@bender:~$ cat ~/output/path.txt
```

33. The output should look like the following:

```
 /home/bender/
```

34. The environmental variable is pretty standard on most operating systems. It should reflect the home directory of the current user. Exit the current machine.

```
bender@bender:~$ exit
```

35. Answer the following questions:

    - **Q: What's the difference between the shell and command modules?**
        - *A: The shell module issues the command through a shell, and therefore understands wildcards. Shells can be "tweaked" (customized), which can lead to unpredictable results. Therefore, the shell module should be avoided.*
    - **Q: How else could we copy an ENV value like $HOME into a file?**
        - *A: Ansible has a tool called a 'lookup plugin'. This allows the lookup of many types of data, including environmental variables.*

36. The following challenge is optional. If you are not interested in performing it, you may skip to the end of the lab.

37. **CHALLENGE 01 (OPTIONAL - DIFFICULT)** - When starting out, you'll often find yourself reaching for `raw`, `shell`, and the `command` modules. However, they are often **not** necessary. Try rewriting the above playbook using the `copy` module and a `lookup()` plugin to achieve the same result, but **without** invoking the `shell` module. *Note: We have not yet studied using copy or lookup plugins, you'll need to look ahead or do some googling if you want to complete this challenge exercise.*

38. **CHALLENGE 01 (SOLUTION) -** The following is a solution to the above challenge. Unlike using the shell module, this solution retains idempotenece. We will study the "copy" module and "lookup plugins" in greater detail later.

```
---
- name: Copy environmental var value into a file without use of shell module
  hosts: planetexpress   # defined in inventory
  gather_facts: yes
  connection: ssh

  tasks:

   - name: make a directory
     file:
       path: ~/output/
       state: directory

  # NOT idempotent (will run every time)
  #- name: use shell module with wildcarding
  #  shell: echo $HOME >> ~/output/path.txt

  # idempotent solution
   - name: use copy to avoid using a shell command
     copy:
       content: "{{ lookup('env', 'HOME') }}"      # this is a, "lookup plugin"  ("left side" of the shell command)
       dest: ~/output/path.txt                     # ("right side" of the shell command)
       force: yes                                  # always overwrite the file if the content is different
```

39. When you're done with this lab, clean up.

```
student@bchd:~$ bash max-teardown.sh
```

40. If you are tracking your code with GitHub. Take a moment, and make a commit.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "working with shell module"`
    - `git push https://github.com/your-account-name/ansible-dev`
    - Type in username & password

15. Ansible Module - copy

Lab Objective

The objective of this lab is to demonstrate the Ansible `copy` Module. The `copy` module copies a file from the local (controller) or remote host machine(s) to a location on the remote machine(s). If you want to harvest files from remote hosts, use the fetch module to return copies to the local box (controller).

Read about the copy module here:

https://docs.ansible.com/ansible/latest/modules/copy_module.html

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the teardown script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id bender@10.10.2.3
```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id fry@10.10.2.4
```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit.

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. Just to stay organized, let's create a directory for files that have no jinja2 in them.

```
student@bchd:~$ mkdir ~/ans/files/
```

16. Let's create Fry's resume and save it in our new "files" directory.

```
student@bchd:~$ vim ~/ans/files/fry.txt
```

17. Create a silly file, such as the following. It doesn't matter what it contains.

```
Name Philip J. Fry I (Fry)
Born August 14, 1974
Height 5ft 9in
Weight: 180 lbs
Native to the 20th century who was cryogenically frozen seconds into the year 2000, having fallen in just as 1999 ended.
He was revived in 2999 and subsequently became a delivery boy for the Planet Express Company. He is the protagonist of
Futurama.
```

18. Save and exit with `:wq`

19. See if your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

20. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

21. Save and exit with `:wq`

22. Skim the documentation for the copy module:

```
student@bchd:~$ ansible-doc copy
```

23. Press `q` to quit.

24. Create a playbook that we can use to demonstrate the copy module, `playbook-copy.yml`.

```
student@bchd:~$ vim ~/ans/playbook-copy.yml
```

25. Create the following playbook.

```
---
- name: Copying files from controller to hosts
  hosts: planetexpress

  tasks:

    - name: Copy Fry's resume to hosts
      copy:
        src: ~/ans/files/fry.txt   # on the controller
        dest: ~/fry.txt      # home directory on hosts
```

26. Save and exit.

27. Next run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-copy.yml
```

28. Now let us connect to the one of the remote hosts with secure shell to see the change we made. Once you ssh to the host, run `ls` to see the new file and `cat fry.txt` to view the contents. Password is *alta3* if you need it.

```
student@bchd:~$ ssh bender@10.10.2.3 (then exit)
student@bchd:~$ ssh fry@10.10.2.4 (then exit)
student@bchd:~$ ssh zoidberg@10.10.2.5 (then exit)
student@bchd:~$ ssh farnsworth@10.10.2.6 (then exit)
```

29. Verify you are back on the beachhead host. From beachhead, create a new playbook, `playbook-copy2.yml`.

```
student@bchd:~$ vim ~/ans/playbook-copy2.yml
```

30. Create the following playbook. In this playbook we eliminate farnsworth from the lineup with `planetexpress:!farnsworth` in the hosts line. This is necessary as farnsworth doesn't define the variable `ansible_all_ipv4_addresses`.

```
---
- name: Copying content from controller to hosts
  hosts: planetexpress:!farnsworth
  gather_facts: yes # default is yes

  tasks:
    - name: Copy gather fact var data into hosts
      copy:
        content: "{{ ansible_all_ipv4_addresses }}" # facts gathered from host
        dest: ~/ipv4info.txt      # home directory on hosts
        backup: yes   # make backup of original
        force: yes    # file always copied when content is diff
```

31. Save and exit.

32. In this playbook, we copied 'content' instead of 'src'. This technique becomes more interesting when we get into using variables (copying the content of variables into a file is a useful technique). In this example, a file called, `ipv4info.txt` is created containing that hosts IPv4 info harvested from the gather_facts module.

33. Try running script.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-copy2.yml
```

34. Now let's connect to the one of the remote hosts with secure shell to see the change we made. (Once you ssh to the host, run `ls` to see the new file and `cat ipv4info.txt` to view the contents).

```
student@bchd:~$ ssh bender@10.10.2.3 (then exit)
student@bchd:~$ ssh fry@10.10.2.4 (then exit)
student@bchd:~$ ssh zoidberg@10.10.2.5 (then exit)
student@bchd:~$ ssh farnsworth@10.10.2.6 (should not have the file! Then exit)
```

35. Verify you are back on the beachhead host.

36. Answer the following questions:
    - **Q: What's a popular use for the copy module?**
        - *A: Moving data from a playbook environment into a static file for long term storage.*
    - **Q: Is the copy module used when building playbooks for storage or network applications?**
        - *A: All the time. As you'll learn, when running against an API or network connection, Ansible runs the copy module in a "local" mode. What that means is the "dest" value becomes the Ansible controller. It becomes VERY useful anytime you need to get data "out" of your playbook environment and into a static file.*
37. Great job! You have completed this lab.
38. If you are tracking your code on GitHub, take a moment and perform a commit.
    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about the copy module"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

16. Ansible Module - apt

Lab Objective

The objective of this lab is to demonstrate the Ansible `apt` Module. The `apt` module allows Ansible to connect to the remote hosts and leverage control of (preferably) aptitude, or alternative, apt-get. Both of these tools are software management tools, just like `yum`, or the `Google Play Store`. These applications allow the installation of fully compiled software, and dependencies, on remote hosts.

Use of `aptitude` and `apt-get` are generally restricted to Debian machines. This will cause issues, as the host Farnsworth is a CentOS RedHat machine, which expects to leverage the use of `yum` to deploy software.

Read the documentation for the `apt` module here: https://docs.ansible.com/ansible/latest/modules/apt_module.html

In this lab, we will leverage use of a variable defined by the `gather_facts` module known as `ansible_distribution`. Many of the possible values for `ansible_distribution` are found here:
https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#ansible-distribution

Procedure

1. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

3. Run the teardown script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

4. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

5. Run the setup script for this lab.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id bender@10.10.2.3
   ```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id fry@10.10.2.4
   ```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
   ```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

    ```
    student@bchd:~$ mkdir -p ~/ans/inv/dev/
    ```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

    ```
    student@bchd:~$ vim ~/ans/inv/dev/hosts
    ```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have at least the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit.

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

16. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

17. Save and exit with `:wq`

18. Let's review usage of a few modules before we continue. The `apt` module allows us to install packages with the `apt` program on remote hosts.

```
student@bchd:~$ ansible-doc apt
```

19. If necessary, press `q` to quit the ansible-doc utility.

20. Create a basic playbook file called `~/ans/playbook-apt.yml`.

```
student@bchd:~$ vim ~/ans/playbook-apt.yml
```

21. Create the following playbook.

```
---
- name: Our first play
  hosts: planetexpress

  tasks:

    - name: Install steam locomotive on ALL machines
      apt:
        name: sl
        state: present
      become: yes # run as sudo
```

22. Save and exit.

23. The above playbook is equivalent to, `sudo apt install sl`.

24. Next run the playbook. We should have one failure on farnsworth, as he is **not** a Debian machine (this host is running CentOS RedHat, which expects to use the `yum` package installer.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-apt.yml
```

25. Connect to the one of the remote hosts with secure shell to see the change we made.

```
student@bchd:~$ ssh fry@10.10.2.4
```

26. Try out the newly installed application.

```
fry@fry:~$ sl
```

27. Choo-Choo! Turns out `sl` is short for steam locomotive. Exit from the ssh session.

```
fry@fry:~$ exit
```

28. We can take a few approaches in how we 'fix' our failing farnsworth. The first might be just to remove him from the host description. We can do that by altering how we describe our hosts. Create a new playbook.

```
student@bchd:~$ vim ~/ans/playbook-apt2.yml
```

29. Ensure this playbook contains the following:

```
---
- name: Our first play
  hosts: planetexpress:!farnsworth

  tasks:

    - name: Install steam locomotive on ALL machines
      apt:
        name: sl
        state: present
      become: yes # run as sudo
```

30. Save and exit.

31. Run the new playbook. This one runs on all hosts in the planetexpress group, EXCEPT, for farnsworth. This method works, but only if we know in advance that the host farnsworth has an unique host OS.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-apt2.yml
```

32. There is another method we could try, which is using a `when` condition. We can reach for `when` if we'd like to construct a conditional. One caveat is that when working within the `when` line it is not necessary to use the double `{{ var }}` technique for calling on variables. Run the following command to reveal a difference between the host `farnsworth` and the rest of the crew:

```
student@bchd:~$ ansible planetexpress -m setup -a "filter=ansible_distribution"
```

33. Looks like `farnsworth` has a value of `CentOS` for `ansible_distribution`, whereas the others have `Ubuntu`. Perfect! Create a new playbook.

```
student@bchd:~$ vim ~/ans/playbook-apt3.yml
```

34. Create the following playbook with the `when` conditional:

```
---
- name: Our first play
  hosts: planetexpress
  gather_facts: yes # runs by default

  tasks:

    - name: Install steam locomotive on ALL machines
      apt:
        name: sl
        state: present
      become: yes # run as sudo
      when: ansible_distribution == "Ubuntu"
```

35. Save and exit.

36. The variable `ansible_distribution` is defined when the `gather_facts` (aka `setup`) module runs. If a host is found to have an Ubuntu distribution, than the task will run. However, it will **not** run if the host is a CentOS RedHat machine (such as host farnsworth). Many of the possible values of `ansible_distribution` may be found here: https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#ansible-distribution

```
student@bchd:~$ ansible-playbook ~/ans/playbook-apt3.yml
```

37. The playbook should execute on all hosts, except the farnsworth host.

38. Answer the following questions:

    - **Q: What is apt?**
        - *A: This is short for aptitude. This is the package installer for Ubuntu and Debian based Linux hosts. Think of it like the Windows Store application or iTunes store application. It allows an admin to install software (apps).*
    - **Q: What if I use a non-Debian or non-Ubuntu Linux system?**
        - *A: Ansible also has a "yum", "pk5", and "package" module (among others). You would select the module that is able to control the package manager on your target infastructure.*

39. Great job! You have completed this lab.

40. If you are tracking your code on GitHub, take a moment and perform a commit.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about the apt module"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

17. Ansible Module - yum

Lab Objective

Here we'll demonstrate the `yum` Ansible module.

To fully understand this lab, understand that yum is a package manager that allows access to free software (like the Apple Store or Google Play store).

Also understand that CentOS is a Fedora RedHat 'clone'.

In this lab, you'll use CentOS, and yum, to install Extra Packages for Enterprise Linux (or EPEL) which is a high quality set of additional packages for Enterprise Linux. Per the website, "EPEL packages are usually based on their Fedora counterparts and will never conflict with or replace packages in the base Enterprise Linux distributions. EPEL uses much of the same infrastructure as Fedora, including buildsystem, bugzilla instance, updates manager, mirror manager and more."

In short, CentOS is a free RedHat clone, and by linking yum to EPEL, we have access to many of the same software packages as we would need to pay to gain access to on a native RedHat system. Understand, we'll issue all of our commands from Beachhead, which is an Ubuntu machine.

Some reading that might help you understand a bit more about this lab:

The Ansible `yum` module: https://docs.ansible.com/ansible/latest/modules/yum_module.html

Extra Packages for Enterprise Linux (EPEL) - https://fedoraproject.org/wiki/EPEL

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the teardown script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id bender@10.10.2.3
```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id fry@10.10.2.4
```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit.

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

16. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

17. Save and exit with `:wq`

18. Let's review usage of a few modules before we continue.

```
student@bchd:~$ ansible-doc yum
```

19. If necessary, press `q` to quit the ansible-doc utility.

20. Create a basic playbook file called `~/ans/playbook-yum.yml`.

```
student@bchd:~$ vim ~/ans/playbook-yum.yml
```

21. Create the following playbook.

```
---
- name: Install sl with yum
  hosts: planetexpress

  tasks:

    - name: Install steam locomotive with yum
      yum:
        name: sl
        state: present
      become: yes # run as sudo
```

22. Save and exit.

23. The above playbook is equivalent to, `sudo yum install sl`.

24. Next run the playbook. We should have one success on farnsworth, as he **is** a CentOS RedHat machine and using the yum package install utility.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-yum.yml
```

25. Connect to the one of the remote hosts with secure shell to see the change we made.

```
student@bchd:~$ ssh farnsworth@10.10.2.6
```

26. Try out the newly installed application.

```
farnsworth@farnsworth:~$ sl
```

27. Choo-Choo! Turns out `sl` is short for steam locomotive. Exit from the ssh session.

```
farnsworth@farnsworth:~$ exit
```

28. The best 'fix' we can apply to our playbook is to use a `when` statement. We could also augment our playbook to include an `apt` module, to install `sl` on the other `Ubuntu` based machines. Create a new playbook.

```
student@bchd:~$ vim ~/ans/playbook-yum2.yml
```

29. Ensure this playbook contains the following:

```
---
- name: Our first play
  hosts: planetexpress
  gather_facts: yes # runs by default

  tasks:

    - name: Install steam locomotive on CentOS machines
      yum:
        name: sl
        state: present
      become: yes # run as sudo
      when: ansible_distribution == "CentOS"

    - name: Install steam locomotive on Ubuntu machines
      apt:
        name: sl
        state: present
      become: yes # run as sudo
      when: ansible_distribution == "Ubuntu"
```

30. Save and exit.

31. The variable `ansible_distribution` is defined when the `gather_facts` (aka `setup`) module runs. If a host is found to have an CentOS distribution, than the `yum` task will run. However, if the `ansible_distribution` is found to be `Ubuntu`, the `apt` package installer will run.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-yum2.yml
```

32. The playbook should execute on all hosts, however the yum task should only execute on the farnsworth host, and the apt task on the remaining hosts.

33. Create another basic playbook file called `playbook-yum3.yml`.

```
student@bchd:~$ vim ~/ans/playbook-yum3.yml
```

34. Create the following playbook. This one shows a fairly practical step, which is installing the, "Extra Packages for Enterprise Linux". You can read about that project here, https://fedoraproject.org/wiki/EPEL however, the TLDR is, when you use CentOS in place of RHEL, you don't have access to the RHEL repositories. the EPEL serves as a replacement for these, 'pay to access' software repositories.

```
---
 - name: Install EPEL libraries in YUM
   hosts: planetexpress
   become: yes # run entire playbook with sudo

   tasks:
    - name: "Install epel-release"
      yum:
        name: epel-release
        state: present
      when: ansible_distribution == "CentOS"

    - name: "Update yum repos and Install package X"
      yum:
        update_cache: True
        name:
            - fail2ban
            - moon-buggy
        state: latest
      when: ansible_distribution == "CentOS"
```

35. Save and exit.

36. Notice we also are installing a list of applications: fail2ban and moon-buggy. The former is an application that prevents users from SSHing to a box if they fail the SSH authentication "X" number of times. The second application, moon-buggy, is a game. Using a list of applications with the `yum` module (as well as the `apt` module) is actually preferred (as it is more efficient) over using a `loop` operation.

37. Next run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-yum3.yml
```

38. Now let's connect to the remote host with secure shell to see the change we made.

```
student@bchd:~$ ssh farnsworth@10.10.2.6
```

39. Verify the newly installed application, moon-buggy.

```
[farnsworth@farnsworth ~]$ moon-buggy
```

40. Use `CTRL` + `C` to exit.

41. Exit from the ssh session.

```
[farnsworth@farnsworth ~]$ exit
```

42. Answer the following questions:

    - **Q: What is yum?**
        - *A: This is the package installer for RedHat based Linux hosts. Think of it like the Windows Store application or iTunes store application. It allows an admin to install software (apps).*
    - **Q: What if I use a non-RedHat based Linux system?**
        - *A: Ansible also has a "apt", "pk5", and "package" module (among others). You would select the module that is able to control the package manager on your target infrastructure.*

43. Great job! You have completed this lab.

44. If you are tracking your code on GitHub, take a moment and perform a commit.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about the yum module"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

18. Ansible Module - get_url

Lab Objective

The objective of this lab is to give an overview of the `get_url` Ansible Module. The primary purpose for this module might be downloading some files from an URL resource as it may download files from HTTP, HTTPS, or FTP to the remote server. The remote host must have direct access to the remote resource (the controller will not proxy the resource to the remote host).

Read the documentation of the `get_url` module here:
http://docs.ansible.com/ansible/latest/get_url_module.html

Procedure

1. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

3. Run the teardown script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

4. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

5. Run the setup script for this lab.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id bender@10.10.2.3
   ```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id fry@10.10.2.4
   ```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
   ```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

    ```
    student@bchd:~$ mkdir -p ~/ans/inv/dev/
    ```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

    ```
    student@bchd:~$ vim ~/ans/inv/dev/hosts
    ```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit.

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

16. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

17. Save and exit with `:wq`

18. Let's review usage of a few modules before we continue. The `get_url` module allows us to get data from across the web.

```
student@bchd:~$ ansible-doc get_url
```

19. If necessary, press `q` to quit the ansible-doc utility.

20. Let's review usage of a few modules before we continue. The `blockinfile` allows us to ensure a block of text appears within a file.

```
student@bchd:~$ ansible-doc blockinfile
```

21. If necessary, press `q` to quit the ansible-doc utility.

22. Create a basic playbook file called `~/ans/geturl-playbook01.yml`

```
student@bchd:~$ vim ~/ans/geturl-playbook01.yml
```

23. Create the following solution. This playbook will download the archived file from the target URL.

```
---
- name: Getting files
  hosts: planetexpress
  gather_facts: no

  tasks:
  - name: "Download virus protection software to install on hosts"
    get_url:
      url: http://www.clamav.net/downloads/production/clamav-0.99.3.tar.gz
      dest: ~/clamav-0.99.3.tar.gz
```

24. Save and exit with `:wq`

25. Next run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/geturl-playbook01.yml
```

26. Now let's connect to the one of the remote hosts with secure shell to see the change we made.

```
student@bchd:~$ ssh bender@10.10.2.3
```

27. Check to see if the newly downloaded gzipped file is on the remote hosts.

```
bender@bender:~$ ls clamav-0.99.3.tar.gz
```

28. Exit from the ssh session.

```
bender@bender:~$ exit
```

29. What if the file was 'protected' with a username, like `sammy` and a password, like `p@55wurd!`? Could Ansible still download the file? We can set up an Apache file sever with a secure directory, `/var/www/html/admin/` on one of our remote hosts, like `fry` with a playbook, and then try using Ansible (again) to download the secured file.

```
student@bchd:~$ vim ~/ans/geturl-setupserver-playbook02.yml
```

30. Create the following playbook that will create our secure file server on the `fry` machine.

```
---
- name: Setup apache2 server
  hosts: fry
  gather_facts: no
  become: yes

  tasks:
  - name: Install apache2 server and ensure pip is installed
    apt:
            name:
                    - apache2
                    - python3-pip
            state: present

  - name: Move configuration block to secure the file
    blockinfile:
            insertafter: EOF
            path: /etc/apache2/apache2.conf
            block: |
                    Alias /admin /var/www/html/admin

                    <Directory /var/www/html/admin>
                      <Files secure.txt>
                        AuthType basic
                        AuthName "Secured Files area"
                        AuthUserFile /etc/apache2/.htpasswd
                        Require user sammy
                      </Files>
                      order allow,deny
                      deny from all
                      satisfy any
                    </Directory>

  - name: create the admin/ folder
    file:
            state: directory
            path: /var/www/html/admin/

  - name: create a protected file we can download
    copy:
            content: "This is a super secret file!\nIf you can read it, you hacked the gibson!"
            dest: /var/www/html/admin/secure.txt
            mode: u=rw,g=r,o=r

  # required for htpasswd to be controlled by python3
  - name: install passlib
    pip:
            name: passlib
            state: present

  # Apache webserver uses htpasswd to make passwords, it so happens ansible has a module to
  # edit this password file. In production, passwords should always be encrypted with vault!
  - name: set a password via htpasswd
    htpasswd:
            path: /etc/apache2/.htpasswd
            name: sammy
            password: p@55wurd!
            owner: root
            group: root
```

```
  - name: restart service
    shell: "/etc/init.d/apache2 restart"
```

31. Save and exit with `:wq`

32. Run your playbook.

```
student@bchd:~$ ansible-playbook ~/ans/geturl-setupserver-playbook02.yml
```

33. Test the playbook with the following curl command. This tries to use the username "sammy" and password "p@55wurd!" to authenticate.

```
student@bchd:~$ curl http://10.10.2.4/admin/secure.txt --user "sammy:p@55wurd!"
```

34. You should get the content of the file `secure.txt` on the screen. If you get an HTTP failure code (4xx), something isn't setup correctly. Assuming everything is going well, let's write a playbook that tries to download the same file to our controller.

```
student@bchd:~$ vim ~/ans/geturl-playbook03.yml
```

35. To download a file to the controller, we'll need to run our playbook in `local` connection mode.

```
---
- name: Get-URL Download the Protected File
  hosts: localhost  # target ourselves
  connection: local # we do not want to SSH anywhere. Run these actions locally

  tasks:
    # in this task we alert Ansible to expect a 401 response
    # the 'register' keyword saves the response sent back to the uri module
    - name: Access the WebSite and make sure the URL is live
      uri:
        url: http://10.10.2.4/admin/secure.txt
        status_code: 401
      register: validateurl

    - name: "INFO: HTTP Response for the URL"
      debug:
              var: validateurl.msg

    - name: Access the same URL with Basic Authentication and Download the file
      get_url:
        url: http://10.10.2.4/admin/secure.txt
        url_password: p@55wurd!
        url_username: sammy
        dest: ~/ansible-downloaded-the-secret-file.txt

    - name: Display the File contents
      command: >
         cat ~/ansible-downloaded-the-secret-file.txt
      register: results

    - name: "INFO: Display the File Contents"
      debug:
              var: results.stdout_lines
```

36. Save and exit with `:wq`

37. Run your playbook.

```
student@bchd:~$ ansible-playbook ~/ans/geturl-playbook03.yml
```

38. The playbook demonstrated that the file was downloaded, but confirm with the following command.

```
student@bchd:~$ cat ~/ansible-downloaded-the-secret-file.txt
```

Great! That's it for this lab.

39.
40. **CHALLENGE 01 (OPTIONAL)** - Rewrite the server playbook so that the user "sammy" and password "p@55wurd!", are defined by variables.
41. **CHALLENGE 02 (OPTIONAL)** - Secure the plain-text passwords (and possibly usernames). You may eliminate them with `vars_prompt`, or secure them with Ansible Vault. Both of these techniques are covered within other labs.
42. Run the tear-down script to tidy up.

```
student@bchd:~$ bash max-teardown.sh
```

43. Answer the following questions:
    - **Q: What is the application for the get_url module?**
      - *A: This module allows Ansible to download content from the web. That content could be HTML, JSON, XML, configuration files, or other data.*
    - **Q: Does Ansible have other ways to download data?**
      - *A: Yes! Many. You could use the uri module, and register the results. You could also find success with the lookup plugin. Both of these techniques you'll learn in later labs. And still many more exist. The merits of each will be discussed later on.*
44. Great job! You have completed this lab.
45. If you are tracking your code on GitHub, take a moment and perform a commit.
    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about the get-url module"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

19. Ansible Module - file

Lab Objective

The objective of this lab is to learn about the `file` module, which can be used to create files, directories, and more.

In this lab, you'll create `~/station_docs/` on some target host machines. A common mistake is to assume modules outputting data will create non-existent directories. Instead, it is much more common that the module will simply fail. Therefore, it is important to understand how directories might be created.

Procedure

1. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

3. Run the teardown script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

4. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

5. Run the setup script for this lab.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id bender@10.10.2.3
   ```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id fry@10.10.2.4
   ```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
   ```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

    ```
    student@bchd:~$ mkdir -p ~/ans/inv/dev/
    ```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

    ```
    student@bchd:~$ vim ~/ans/inv/dev/hosts
    ```

12. **Notice that this inventory has a unique entry for the var** `fileuser`. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following).

    ```
    [planetexpress]
    bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3 fileuser=bender
    fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3 fileuser=fry
    zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3 fileuser=zoidberg
    farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3 fileuser=farnsworth
    ```

Save and exit.

13.

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

16. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

17. Save and exit with `:wq`

18. Let's review usage of a few modules before we continue. The `file` module allows us to ensure a file or directory exists or is removed. *Note: If you tell Ansible to access a directory that does not exist, the playbook will throw an error!*

```
student@bchd:~$ ansible-doc file
```

19. If necessary, press `q` to quit the ansible-doc utility.

20. Create a basic playbook file called `playbook-file.yml`

```
student@bchd:~$ vim ~/ans/playbook-file.yml
```

21. Create the following playbook. Notice that this playbook uses the `loop` keyword.

```
---
- name: A playbook showing the file module
  hosts: planetexpress

  vars:
    # the variable "st" is mapped to "directory"
    st: directory

  tasks:
  - name: "Create directory where Space Station documents will reside"
    become_user: root
    become: true
    file:
      path: "/home/{{ item }}/station_docs/"
      state: "{{ st }}"
      owner: "{{ item }}"
      group: "{{ item }}"
      mode: "0755"
    loop: [fry, bender, zoidberg]
```

22. Save and exit.

23. Run the playbook, **expect it to fail**

```
student@bchd:~$ ansible-playbook ~/ans/playbook-file.yml
```

24. As you can see, there were errors with this approach. Do you know why? The `{{ item }}` will start an iteration for the items in the list and try to make that change to each host. The hosts, however, do not have one another's hostname as a user or a group, so then you receive that error.

25. Not only did it not complete cleanly, it had some unintended consequences. SSH over to bender, and check out the results.

```
student@bchd:~$ ssh bender@10.10.2.3
```

26. List the local directories.

```
bender@bender:~$ ls
```

27. List the directories in `/home/` on the bender machine.

```
bender@bender:~$ ls /home/
```

28. Interesting. So we can see a Space Station Docs, but the way we wrote our loop, it also created additional home directories on our machine. Bender, now also has a `/home/fry/` and `/home/zoidberg` directory on it (the same error occurred on the other machines as well). However, they lack the new directory `station_docs`. We didn't want this to happen. We just wanted `station_docs` to appear in each user's home directory.

```
bender@bender:~$ exit
```

29. To "undo" the damage, rerun the playbook, but change the value of `st` to "absent".

```
student@bchd:~$ ansible-playbook ~/ans/playbook-file.yml -e "st=absent"
```

30. Create a new playbook.

```
student@bchd:~$ vim ~/ans/playbook-file02.yml
```

31. We can use a host variable to fix this. Edit the `~/ans/playbook-file02.yml` so that it looks like the following:

```
---
- name: A playbook showing the file module
  hosts: planetexpress

  tasks:
   - name: "Create directory where Space Station documents will reside"
     become_user: root
     become: true
     file:
        # the var "fileuser" is a host variable
        path: "/home/{{ fileuser }}/station_docs/"
        state: directory
        owner: "{{ fileuser }}"
        group: "{{ fileuser }}"
        mode: "0755"
```

32. Save and exit with `:wq`

33. Run the playbook again.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-file02.yml
```

34. This time, it works! Now let's connect to the remote hosts with secure shell to see the changes we made. First, to `bender` to ensure the Space Station directory has been created.

```
student@bchd:~$ ssh bender@10.10.2.3
```

35. List the files and folders on the `bender` machine.

```
bender@bender:~$ ls
```

36. Exit `bender` machine.

```
bender@bender:~$ exit
```

37. Now to check on the `fry` machine.

```
student@bchd:~$ ssh fry@10.10.2.4
```

38. List the files and folder on the `fry` machine.

```
fry@fry:~$ ls
```

39. Exit the `fry` machine

```
fry@fry:~$ exit
```

40. Finally, check out the `zoidberg` machine.

```
student@bchd:~$ ssh zoidberg@10.10.2.5
```

41. List the files and folders on the `zoidberg` machine.

```
zoidberg@zoidberg:~$ ls
```

42. Exit from the ssh session back to the beachhead host.

```
zoidberg@zoidberg:~$ exit
```

43. So what we just learned, was that a host variable is mapped ONLY to the host beside which it appears in the inventory file. Cool!

44. Create a new playbook.

```
student@bchd:~$ vim ~/ans/playbook-file03.yml
```

45. Create the following solution

```
---
- name: A playbook showing the file module
  hosts: planetexpress

  tasks:
   - name: Create directory where Space Station documents will reside
     become_user: root
     become: true
     file:
        # the var "fileuser" is a host variable
        path: "/home/{{ fileuser }}/station_docs/"
        state: directory
        owner: "{{ fileuser }}"
        group: "{{ fileuser }}"
        mode: "0755"

   - name: Copy content into a file with the year-mo-day
     copy:
        # the var "fileuser" is a host variablen
        # ansible_date_time.date is defined by gather_facts
        dest: "/home/{{ fileuser }}/station_docs/navdata-{{ ansible_date_time.date }}.txt"
        content: "Earth\nMars\nSaturn\nEuropa\n"
```

46. Save and exit.

47. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-file03.yml
```

48. Confirm that the files were created by logging into `bender` machine.

```
student@bchd:~$ ssh bender@10.10.2.3
```

49. List the files and folders on the `bender` machine.

```
bender@bender:~$ ls station_docs/
```

50. View the contents of the document in `station_docs`

```
bender@bender:~$ cat station_docs/navdata-*
```

51. Exit `bender` machine.

```
bender@bender:~$ exit
```

52. Ensure the `fry`, and `zoidberg` have their own versions of this document.

53. Make one last example of this playbook.

```
student@bchd:~$ vim ~/ans/playbook-file04.yml
```

54. Read about the `ansible_user` variable here: https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html Then, create the following solution.

```
---
- name: A playbook showing the file module
  hosts: planetexpress

  tasks:
   - name: Create directory where Space Station documents will reside
     become_user: root
     become: true
     file:
        # the var "ansible_user" is a host variable AND connection variable
        path: "/home/{{ ansible_user }}/station_docs/"
        state: directory
        owner: "{{ ansible_user }}"
        group: "{{ ansible_user }}"
        mode: "0755"

   - name: "Create directory where Space Station documents will reside"
     copy:
        # the var "ansible_user" is a host variable AND connection variable
        # ansible_date_time.date is defined by gather_facts
        dest: "/home/{{ ansible_user }}/station_docs/navdata-{{ ansible_date_time.date }}.txt"
        content: "Earth\nMars\nSaturn\nEuropa"
```

55. Save and exit.

56. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-file04.yml
```

57. Answer the following questions:

    - **Q: What is the application for the file module?**
        - *A: To create files or folders (directories) or remove them.*
    - **Q: Is this module used often?**
        - *A: All the time! There is another great module to check out, 'stat', which can return checksums and build conditionals when trying to determine IF a file or folder exists. However, if you need to ensure a file or folder is created, reach for 'file'.*

58. Great job! You have completed this lab.

59. If you are tracking your code on GitHub, take a moment and perform a commit.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about the file module"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

20. Ansible Module - git

Lab Objective

The objective of this lab is to demonstrate another useful module, `git`. This module can be used to work with the git tool, and as such, has a pre-requisite, which is that the git utility is installed on the hosts executing the Ansible code.

If you've forgotten, Ansible works by connecting to machines, and then moving Python code to those machines to execute. In most cases, the machines executing python code would be your 'remote hosts'.

Review the git documentation here:
http://docs.ansible.com/ansible/latest/git_module.html

Procedure

1. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

3. Run the teardown script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

4. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

5. Run the setup script for this lab.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id bender@10.10.2.3
   ```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id fry@10.10.2.4
   ```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
   ```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

    ```
    student@bchd:~$ mkdir -p ~/ans/inv/dev/
    ```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

    ```
    student@bchd:~$ vim ~/ans/inv/dev/hosts
    ```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit.

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

16. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

17. Save and exit with `:wq`

18. Check out the module documentation.

```
student@bchd:~$ ansible-doc git
```

19. If necessary, use `q` to quit the ansible-doc utility.

20. Create a basic playbook file called `modulegit.yml`

```
student@bchd:~$ vim ~/ans/modulegit.yml
```

21. Create the following playbook:

```
---
- name: Learning how to use git
  hosts: planetexpress

  tasks:

  - name: "Create a git archive from a repository"
    become_user: root
    become: true
    git:
        repo: https://github.com/ansible/ansible-examples.git
        dest: /home/student/ansible-examples/
        archive: /tmp/READMEmd.zip
```

22. Save and exit.

23. Run your playbook. In this example, we'll just point to the inventory directory. Ansible will scan the entire directory for files containing an entry for `planetexpress`

```
student@bchd:~$ ansible-playbook ~/ans/modulegit.yml -i ~/ans/inv/dev/
```

24. Now let's connect to the remote hosts with secure shell to see the repository and archive we brought down.

```
student@bchd:~$ ssh bender@10.10.2.3
```

25. Display the new repo on the `bender` machine.

```
bender@bender:~$ ls /home/student/ansible-examples/
```

26. Cool. So the directory `ansible-examples` was even created for us. This doesn't always happen with Ansible. Typically, if a folder does not exist, the playbook will encounter a `failure`. To create directories, see the `file` module. Exit the `bender` machine.

```
bender@bender:~$ exit
```

27. For good measure, also check fry:

```
student@bchd:~$ ssh fry@10.10.2.4

fry@fry:~$ ls /home/student/ansible-examples/
```

28. Exit the `fry` machine.

```
fry@fry:~$ exit
```

29. If you'd like, repeat the process for `zoidberg@10.10.2.5` and `farnsworth@10.10.2.6`

30. Takeaways are as follows:

    - The git module can be used to interact with the git software tool
    - The git tool is required to be installed on the hosts executing the Ansible git module (remote hosts)
    - The git module creates the directory if it does not already exist.

31. Answer the following questions:

    - **Q: What is the application for the git module?**
        - *A: To check out code from a repository like GitHub, GitLab, or BitBucket.*
    - **Q: Does this module let me create git commits? Or just clone code?**
        - *A: Just clone code. The process of creating a commit SHOULD involve attention to any warning messages that may pop-up, as well as writing a detailed message of why you are creating the commit. In short, git commits are not a process that should be fully automated.*

32. Great job! You have completed this lab.

33. If you are tracking your code on GitHub, take a moment and perform a commit.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about the git module"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

21. Ansible Module - template

Lab Objective

The objective of this lab is to demonstrate the use of the `template` Ansible Module. The `template` module expects to move a file (template) formatted with Jinja2 formatting, to a target location. This target location might include the localhost. Jinja2 allows for many template 'tricks' that make it a very useful module for creating those things with dynamic configurations (like config files and web pages).

In this lab we'll try building a `/etc/fstab` file. Fstab is your Linux operating system's file system table. The fstab file is read by the `mount` command, which happens automatically at boot time to determine the overall file system structure, and thereafter when a user executes the mount command to modify that structure. It is the duty of the system or storage administrator to properly create and maintain the fstab file. So, let's learn how one technique for manufacturing this kind of file with Ansible.

Important go skim the documentation for template: https://docs.ansible.com/ansible/latest/modules/template_module.html

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the teardown script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id bender@10.10.2.3
```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id fry@10.10.2.4
```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit.

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

16. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

17. Save and exit with `:wq`

18. Read about the template module. The template module can be used to render incomplete templates into finished files. The blanks in the templates are "filled in" via variables provided by the playbook.

```
student@bchd:~$ ansible-doc template
```

19. If necessary, press `q` to quit the ansible-doc utility.

20. Create a templates directory.

```
student@bchd:~$ mkdir ~/ans/templates/
```

21. New to Jinja2 and templating? Check out https://jinja.palletsprojects.com/en/3.0.x/

22. Create a config file for the planetexpress ship under `~/ans/templates/` and call it `fstab.j2`. In this fictitious example, we want a **finished** file called `fstab`. We add the `.j2` extension to indicate that this file is a *template*. Essentially, it is 'unfinished'.

```
student@bchd:~$ vim ~/ans/templates/fstab.j2
```

23. Create the following file. This template expects the variable `fstab_entries` to be defined in order to run.

```
# {{ansible_managed}}
#
#
# /etc/fstab format:
#
#  device  mount_point  file_system_type  options  backup_operation  file_system_check_order
#

{% if fstab_entries is defined %}

{% for fs in fstab_entries %}
{{fs.device}} {{fs.mount_point}} {{fs.file_system}} {{fs.options}} {{fs.backup}} {{fs.file_sys_check}}
{% endfor %}

{% else %}
# No data was found when ansible attempted to generate /etc/fstab
{% endif %}
```

24. Save and exit with `:wq`
25. Answer the following questions:
    - **Q: What is the significance of a mustache bracket and percentage sign?**
        - *A: This indicates 'jinja' logic, and will not be rendered.*
    - **Q: What is the for statement?**
        - *A: This is creating a "loop" across the list contained within 'fstab_entries'*
26. Great, now we need a playbook that will render our template, and push it to our remote hosts.

```
student@bchd:~$ vim ~/ans/template-playbook01.yml
```

27. Create the following:

```
---
- name: Exploring the template module and jinja expressions
  hosts: planetexpress
  connection: ssh    # default "ssh"
  gather_facts: no   # default "yes" - runs the "setup" module on remote hosts

  # these variables are used to generate /etc/fstab from a template
  vars:
          # fstab_entries is a dictionary containing a list
          # each list entry is a new line within /etc/fstab
          fstab_entries:
                  # each list item is a dictionary containing (6) key:values
                  # debugfs /sys/kernel/debug debugfs noauto 0 0
                  - device: debugfs
                    # specify the mount point where disk should be mounted
                    mount_point: /sys/kernel/debug
                    # file system type
                    file_system: debugfs
                    # mount options, separated by commas
                    options: noauto
                    # 1 for dump utility backup partition
                    # 0 turns this off
                    backup: 0
                    # determines the order that fsck checks the dev/partition for errors at boot time
                    # 0 means fsck should not check a file system
                    file_sys_check: 0
                  - device: /dev/sda2
                    mount_point: /
                    file_system: ext3
                    options: acl,user_xattr
                    backup: 1
                    file_sys_check: 1
                  - device: /dev/sdc1
                    mount_point: /novi_disk
                    file_system: ext3
                    options: acl,user_xattr,usrquota,grpquota
                    backup: 0
                    file_sys_check: 2


  tasks:
    # here we create file we might use for configurating /etc/fstab
    # or ensuring /etc/fstab is in compliance
    - name: Create a dynamic /etc/fstab file from a static template
      template:
        src: templates/fstab.j2    # name of the template on ansible controller
        dest: ~/fstab_example   # name of the completed file to be
                                #    placed on the target system
```

28. Save and exit with `:wq`

29. Next, run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/template-playbook01.yml
```

30. Now let's connect to the one of the remote hosts with secure shell to see the change we made.

```
student@bchd:~$ ssh bender@10.10.2.3
```

31. Notice in our playbook, we targeted `~/fstab_example` as the destination. In production, we would want to use `/etc/fstab`.

```
bender@bender:~$ cat ~/fstab_example
```

32. Answer the following questions:
    - **Q: How many entries are in the file?**
        - *A: 3 active entries are in the /etc/fstab file*
    - **Q: Did we define the variable ansible_managed?**
        - *A: No. This variable may be defined, but it is also auto defined by Ansible.*
    - **Q: Why would you want to include a reminder that some file had been generated by Ansible?**
        - *A: To prevent others (or possibly yourself) from attempting to ever edit the file manually. Ansible may rewrite it!*

33. Exit the bender machine.

```
bender@bender:~$ exit
```

34. If you'd like, you can ssh to fry, zoidberg, and farnsworth to see their files as well.

35. Create a new playbook.

```
student@bchd:~$ vim ~/ans/template-playbook02.yml
```

36. Create the following:

```
---
- name: Exploring the template module and jinja expressions
  hosts: planetexpress
  connection: ssh    # default "ssh"
  gather_facts: no   # default "yes" - runs the "setup" module on remote hosts

  # these variables are used to generate /etc/fstab from a template
  vars_files:
    - vars/fstab.yml

  tasks:
    # here we create file we might use for configurating /etc/fstab
    # or ensuring /etc/fstab is in compliance
    - name: Create a dynamic /etc/fstab file from a static template
      template:
        src: templates/fstab.j2    # name of the template on ansible controller
        dest: ~/fstab_example   # name of the completed file to be
                                #    placed on the target system
```

37. Save and exit with `:wq`

38. Now create a file containing all of the variables that were previously within the playbook. Best practice says this file should be under a `vars` folder.

```
student@bchd:~$ mkdir -p ~/ans/vars/
```

39. Create the file `~/ans/vars/fstab.yml`

```
student@bchd:~$ vim ~/ans/vars/fstab.yml
```

40. Make the following file:

```
# fstab_entries is a dictionary containing a list
# each list entry is a new line within /etc/fstab
fstab_entries:
        # each list item is a dictionary containing (6) key:values
        # debugfs /sys/kernel/debug debugfs noauto 0 0
        - device: debugfs
          # specify the mount point where disk should be mounted
          mount_point: /sys/kernel/debug
          # file system type
          file_system: debugfs
          # mount options, separated by commas
          options: noauto
          # 1 for dump utility backup partition
          # 0 turns this off
          backup: 0
          # determines the order that fsck checks the dev/partition for errors at boot time
          # 0 means fsck should not check a file system
          file_sys_check: 0
        - device: /dev/sda2
          mount_point: /
          file_system: ext3
          options: acl,user_xattr
          backup: 1
          file_sys_check: 1
        - device: /dev/sdc1
          mount_point: /novi_disk
          file_system: ext3
          options: acl,user_xattr,usrquota,grpquota
          backup: 0
          file_sys_check: 2
```

41. Save and exit with `:wq`

42. Next, run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/template-playbook02.yml
```

43. We didn't change anything, other than where our variables were stored. Therefore, the playbook should report all GREEN (no changes).

44. **CHALLENGE 01** - Edit the file `~/ans/vars/fstab.yml`. Create a new list entry within the variable `fstab_entries` so that (at least) one new line is added to `~/fstab_example`. When you've made your changes, rerun the playbook and confirm everything is still working.

45. Answer the following questions:

    - **Q: What is the application for the template module?**
      - *A: Anytime you need to dynamically populate a template into a finished file. Popular case uses might include rendering HTML page data, a configuration file for an application, or running configuration for a network device.*
    - **Q: Where did the double curly bracket thing come from?**
      - *A: Jinja2 is a Python library invented by the HTTP Framework engine, Django. It was the Django project that decided variables would be surrounded by double curly brackets, whereas logic statements would be a double curly bracket followed by a percentage sign. The project is fully described at https://jinja.palletsprojects.com/en/2.11.x/*

46. Great job! You have completed this lab.

47. If you are tracking your code on GitHub, take a moment and perform a commit.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about the template module"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

22. Deploying Web Services with Ansible

Lab Objective

This lab will demonstrate building a basic webservice with Ansible, using the basic build-in modules (those packaged with Ansible's installation.

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. Download the teardown script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the teardown script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. Ping the each of the below lab hosts.

```
student@bchd:~$ ping -c 1 10.10.2.3
student@bchd:~$ ping -c 1 10.10.2.4
```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id bender@10.10.2.3
```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id fry@10.10.2.4
```

9. The `ansible.cfg` should describe your default inventory location (typically, you want to set default to development).

```
student@bchd:~$ vim ~/.ansible.cfg
```

10. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

11. Save and exit with `:wq`

12. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

13. Edit your inventory file.

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

14. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following). Notice that in this inventory, we've created a group called, `web` that includes `bender` and `fry`

```
[web]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
```

15. Save and exit.

16. Create a basic playbook file called `playbook-handler01.yml`

```
student@bchd:~$ vim ~/ans/playbook-handler01.yml
```

17. Replicate the following playbook. Know that `apache2` is the Ubuntu name for the `httpd` service you'd find on an RHEL or CentOS box.

```
---
- name: Apache server installed
  hosts: web
  gather_facts: no
  become: yes

  tasks:

  # the package module tries to select
  # yum or apt or pkg5 (etc) automatically
  - name: latest Apache version installed
    package:
      name: apache2
      state: latest

  - name: Apache enabled and running
    service:
      name: apache2
      enabled: yes
      state: started
```

18. Save and exit with `:wq`

19. Run the playbook. It should deploy the Apache `httpd` webservice on each of the hosts described by `web`

```
student@bchd:~$ ansible-playbook ~/ans/playbook-handler01.yml -i ~/ans/inv/dev/hosts
```

20. First, cURL the `bender` webserver, and ensure the Apache webservice is responding. You should get an HTTP response.

```
student@bchd:~$ curl http://10.10.2.3
```

21. Next, cURL the `fry` webserver, and ensure the Apache webservice is responding.

```
student@bchd:~$ curl http://10.10.2.4
```

22. Let's update our playbook.

```
student@bchd:~$ vim ~/ans/playbook-handler01.yml
```

23. Add the following tasks to our playbook.

```
---
- name: Apache server installed
  hosts: web
  gather_facts: no
  become: yes

  tasks:

  # the package module tries to select
  # yum or apt or pkg5 (etc) automatically
  - name: latest Apache version installed
    package:
      name: apache2
      state: latest

  - name: Apache enabled and running
    service:
      name: apache2
      enabled: yes
      state: started

  # Copy index.html into the service
  - name: copy index.html
    copy:
      src: ~/ans/files/index.html
      dest: /var/www/html/
```

24. Save and exit.

25. Let's also create an `index.html` file that can be moved into our Apache server. Best practice says this should live in a folder called `files`. Make that now.

```
student@bchd:~$ mkdir -p ~/ans/files/
```

26. Create `index.html`

```
student@bchd:~$ vim ~/ans/files/index.html
```

27. Make `index.html` look like the following:

```
<body>
<h1>Apache Webserver is up and running!</h1>
Way to automate with Ansible!
</body>
```

28. Save and exit.

29. Great. Try running the playbook again.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-handler01.yml -i ~/ans/inv/dev/hosts
```

30. Once again, cURL the `bender` webserver. The returned page should now be our `index.html` page

```
student@bchd:~$ curl http://10.10.2.3
```

31. Don't forget to cURL the `fry` webserver.

```
student@bchd:~$ curl http://10.10.2.4
```

32. While we can change the content being returned by the service, we can't change the configuration of the service without restarting it. Update the playbook again to include two new tasks, the `get_url` and `service` modules.

```
student@bchd:~$ vim ~/ans/playbook-handler01.yml
```

33. Make the following additions to the end of the playbook:

```
---
- name: Apache server installed
  hosts: web
  gather_facts: no
  become: yes

  tasks:

  # the package module tries to select
  # yum or apt or pkg5 (etc) automatically
  - name: latest Apache version installed
    package:
      name: apache2
      state: latest

  - name: Apache enabled and running
    service:
      name: apache2
      enabled: yes
      state: started

  # Copy index.html into the service
  - name: copy index.html
    copy:
      src: ~/ans/files/index.html
      dest: /var/www/html/

  # if dest is directory download every time
  # but only replace if destination is different
  - name: Download a copy of apache2.conf
    get_url:
      url: https://raw.githubusercontent.com/rzfeeser/alta3files/master/apache2.conf
      dest: /etc/apache2/

  - name: restart_apache
    service:
      name: apache2
      state: restarted
```

34. Save and exit.

35. Great. Try running the playbook. The playbook should run successfully.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-handler01.yml -i ~/ans/inv/dev/hosts
```

36. Run the playbook 2 or 3 more times, then answer the following questions:

    - **Q: If I run the playbook again, will I not ALWAYS cause a service restart for apache2?**
      - *A: Yes! And that is poor design. We ONLY want to restart the service IF a configuration file has been changed. This is best accomplished with a "handler", and will be addressed in a later lab.*
    - **Q: Is there a way to shorten this playbook so it is "less complex" or "easier to share" with others?**
      - *A: Yes. This playbook is actually very short, and would still be difficult to copy & paste. Solutions can be transformed into "roles", which are easy-to-invoke and share Ansible solutions. They are a bit like functions in Python. We'll study roles a bit later.*

37. Great job! You have completed this lab.

38. If you are tracking your code on GitHub, take a moment and perform a commit.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about deploying web-services with Ansible"`

- `git push https://github.com/your-account-name/ansible-dev master`
- Type in username & password

23. Roles and ansible-galaxy

Lab Objective

The objective of this lab is to introduce the Ansible Galaxy. Ansible Galaxy is both a website, and a tool that installs with Ansible. The website serves as a place to show off the work you create with the Ansible Galaxy client, which is roles and collections.

A role is a highly organized set of Ansible code. The organization is across a directory hierarchy.

A collection is content distribution, where the content is designed to augment Ansible's capabilities. A collection could include a role, but may also include additional modules, or other plugins.

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. Download the teardown script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the teardown script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. Ping the each of the below lab hosts.

```
student@bchd:~$ ping -c 1 10.10.2.3
student@bchd:~$ ping -c 1 10.10.2.4
```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id bender@10.10.2.3
```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id fry@10.10.2.4
```

9. The `ansible.cfg` should describe your default inventory location (typically, you want to set default to development).

```
student@bchd:~$ vim ~/.ansible.cfg
```

10. Your file should reflect the following values. Notice the new addition of `roles_path`:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no

# controlling WHERE ansible-galaxy installs / looks for roles
roles_path      = /home/student/ans/roles/
```

11. Save and exit with `:wq`
12. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

13. Edit your inventory file.

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

14. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following). Notice that in this inventory, we've created a group called, `web` that includes `bender` and `fry`

```
[web]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
```

15. Save and exit.
16. Consider the following playbook. We created this in a previous lab. No need to copy it yourself, but this is what we want to turn into a role. It is a solution that deploys a webservice. Some small changes have been made, such the addition of a `vars` section that maps the variable `pw` to `pass123`.

```
---
- name: Setup apache2 server
  hosts: fry
  gather_facts: no
  become: yes

  vars:
    pw: pass123

  tasks:
  - name: Install apache2 server and ensure pip is installed
    apt:
            name:
                    - apache2
                    - python3-pip
            state: present

  - name: Move configuration block to secure the file
    blockinfile:
            insertafter: EOF
            path: /etc/apache2/apache2.conf
            block: |
                    Alias /admin /var/www/html/admin

                    <Directory /var/www/html/admin>
                      <Files secure.txt>
                        AuthType basic
                        AuthName "Secured Files area"
                        AuthUserFile /etc/apache2/.htpasswd
                        Require user sammy
                      </Files>
                      order allow,deny
                      deny from all
                      satisfy any
                    </Directory>

  - name: create the admin/ folder
    file:
            state: directory
            path: /var/www/html/admin/

  - name: create a protected file we can download
    copy:
            content: "This is a super secret file!\nIf you can read it, you hacked the gibson!"
            dest: /var/www/html/admin/secure.txt
            mode: u=rw,g=r,o=r

  # required for htpasswd to be controlled by python3
  - name: install passlib
    pip:
            name: passlib
            state: present

  # Apache webserver uses htpasswd to make passwords, it so happens ansible has a module to
  # edit this password file. In production, passwords should always be encrypted with vault!
  - name: set a password via htpasswd
    htpasswd:
            path: /etc/apache2/.htpasswd
            name: sammy
            password: "{{ pw }}"
```

```
            owner: root
            group: root

  - name: restart service
    shell: "/etc/init.d/apache2 restart"
```

17. Let's try creating a role. Use the `ansible-galaxy` command to create a role directory structure.

```
student@bchd:~$ ansible-galaxy role init ~/ans/roles/alta3.webservice
```

18. Install tree to "see" the role directory structure.

```
student@bchd:~$ sudo apt install tree -y
```

19. Move into the role directory.

```
student@bchd:~$ cd ~/ans/roles/
```

20. Run tree.

```
student@bchd:~/ans/roles$ tree
```

21. These are all the default directories and files. Notice most files are called `main.yml`. When a role is invoked, the first file that is read is `tasks/main.yml`. All we need to do is put our tasks in this file.

```
student@bchd:~/ans/roles$ vim ~/ans/roles/alta3.webservice/tasks/main.yml
```

22. Copy and paste the following tasks into this file.

```
- name: Install apache2 server and ensure pip is installed
  apt:
          name:
                  - apache2
                  - python3-pip
          state: present

- name: Move configuration block to secure the file
  blockinfile:
          insertafter: EOF
          path: /etc/apache2/apache2.conf
          block: |
                  Alias /admin /var/www/html/admin

                  <Directory /var/www/html/admin>
                    <Files secure.txt>
                      AuthType basic
                      AuthName "Secured Files area"
                      AuthUserFile /etc/apache2/.htpasswd
                      Require user sammy
                    </Files>
                    order allow,deny
                    deny from all
                    satisfy any
                  </Directory>

- name: create the admin/ folder
  file:
          state: directory
          path: /var/www/html/admin/

- name: create a protected file we can download
  copy:
          content: "This is a super secret file!\nIf you can read it, you hacked the gibson!"
          dest: /var/www/html/admin/secure.txt
          mode: u=rw,g=r,o=r

# required for htpasswd to be controlled by python3
- name: install passlib
  pip:
          name: passlib
          state: present

# Apache webserver uses htpasswd to make passwords, it so happens ansible has a module to
# edit this password file. In production, passwords should always be encrypted with vault!
- name: set a password via htpasswd
  htpasswd:
          path: /etc/apache2/.htpasswd
          name: sammy
          password: "{{ pw }}"
          owner: root
          group: root

- name: restart service
  shell: "/etc/init.d/apache2 restart"
```

23. Save and exit with `:wq`

24. Now let's define some a default value for our var, `pw`.

```
student@bchd:~/ans/roles$ vim ~/ans/roles/alta3.webservice/vars/main.yml
```

25. Make the following addition.

```
pw: pass123
```

26. Save and exit with `:wq`

27. We're done! If we wanted we could clean up the unused folders. These could be utilized if we wished to make a more complicated role, but for now, let's use our work in a playbook.

```
student@bchd:~/ans/roles$ cd ~/ans/
```

28. Create a new playbook.

```
student@bchd:~/ans$ vim playbook-webservice-role.yml
```

29. Create the following solution that invokes our role.

```
- name: Using a role to deploy a webservice
  hosts: web
  gather_facts: no
  become: yes

  roles:
    - alta3.webservice
```

30. Save and exit with `:wq`

31. Great. Try running the playbook. The playbook should run successfully.

```
student@bchd:~$ ansible-playbook playbook-webservice-role.yml -i ~/ans/inv/dev/hosts
```

32. Access the secure file with the following curl command. This command targets the bender server, but you could change the IP address and also target fry (10.10.2.4).

```
student@bchd:~$ curl http://10.10.2.3/admin/secure.txt --user "sammy:pass123"
```

33. Remember, Ansible is **state management**. Therefore, we should be able to change the state of our webservers by rerunning the playbook, and simply providing the new values we wish to apply. In this case, let's change the password from `pass123` to `shrubbery`. To do this, we don't need to edit the role-- just overwrite the precedence of the var `pw`. This can be done at the command line via an "extra" variable.

```
student@bchd:~$ ansible-playbook playbook-webservice-role.yml -i ~/ans/inv/dev/hosts -e pw=shrubbery
```

34. Try accessing the file again. On both the bender and fry machines, the password has now been changed to `shrubbery`.

```
student@bchd:~$ curl http://10.10.2.3/admin/secure.txt --user "sammy:shrubbery"
```

35. Answer the following questions:

    - **Q: What is the purpose of a role?**
        - *A: A role is a bit like a function. Once we have perfected a playbook, we can organize it as a role. Since the role is "just" a directory, we could then post it for download. Ansible provides a website called the Ansible Galaxy to do this, but you could also use an internal storage tool, such as GitHub, GitLab, or BitBucket.*
    - **Q: Can more than one role appear in a playbook?**
        - *A: Yes. This is our first introduction to roles, so we are keeping things simple. However, playbooks may contain multiple roles, as well as additional tasks. A later lab goes into detail on the many sections a playbook might have, as well as the order in which they execute.*

36. Great job! You have completed this lab.

37. If you are tracking your code on GitHub, take a moment and perform a commit.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`

- `git commit -m "learning about deploying roles with Ansible"`
- `git push https://github.com/your-account-name/ansible-dev master`
- Type in username & password

24. Ansible Galaxy

Lab Objective

The objective of this lab is to provide greater clarity on "Ansible Galaxy". Ansible Galaxy is comprised of two parts; a website (galaxy.ansible.com), and a CLI tool (`ansible-galaxy`). The relationship here is analogous to that of Python's *pypi.org* (webiste), and Python's *pip*. One is a visual aide for sharing code, the other is a CLI tool for harvesting that code.

The Ansible Galaxy website allows users to share roles and other Ansible plugins via collections. Effectively, galaxy.ansible.com points to repos on GitHub where content is actually stored.

The Ansible Galaxy CLI tool, *ansible-galaxy*, allows users to interact with roles described on the website. Using the CLI tool, users can install, create and manage roles locally.

Visit the Ansible Galaxy here: https://galaxy.ansible.com

Procedure

1. Open a webpage to https://galaxy.ansible.com
2. Create a new directory to work in.

```
student@bchd:~$ mkdir -p ~/ans/roles
```

3. The `ansible-galaxy` can be used to download roles to the path specified by the environment variable ANSIBLE_ROLES_PATH. When Ansible is first installed it defaults to `/etc/ansible/roles`, which requires root privileges. Override this by setting the environment variable in your session, defining `roles_path` in an `ansible.cfg` file, or by using the `-roles-path` option. To start, set `roles_path` within `~/.ansible.cfg`.

```
student@bchd:~$ vim ~/.ansible.cfg
```

4. Copy and paste the following into `~/.ansible.cfg`. This will control where Ansible Galaxy downloads roles to. By the way, it is alright if other values exist within config file.

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no

# controlling WHERE ansible-galaxy installs / looks for roles
roles_path      = /home/student/ans/roles/
```

5. Save and exit with `:wq`
6. Let's try out our work. Roles can be installed from Ansible-Galaxy. There is plenty to explore. Let's start with one from an Alta3 Research instructor, Zach Feeser. Check out the home page here: https://galaxy.ansible.com/rzfeeser/ansible_role_minecraft
7. Looks like this role installs a minecraft server on the machine it is run on. Cool! Try installing the role with the ansible-galaxy command found on the webpage.

```
student@bchd:~$ ansible-galaxy install rzfeeser.ansible_role_minecraft
```

8. Check out the results of our work. Move into the directory we described within `ansible.cfg`.

```
student@bchd:~$ cd /home/student/ans/roles
```

9. See that the role `rzfeeser.ansible_role_minecraft` is there.

```
student@bchd:~/ans/roles$ ls -ll
```

10. Exploring complex directories can be made simpler with `tree`. Install tree now.

```
student@bchd:~/ans/roles$ sudo apt install tree -y
```

11. Run tree.

```
student@bchd:~/ans/roles$ tree ~/ans/roles/rzfeeser.ansible_role_minecraft
```

12. Note that the role name, `rzfeeser.ansible_role_minecraft` is simply the name of the parent directory. All of the subsequent directories are related to that role. Roles typically begin with `rolename/tasks/main.yml` (but not always). Check that file out now:

```
student@bchd:~/ans/roles$ cat rzfeeser.ansible_role_minecraft/tasks/main.yml
```

13. Looks like `main.yml` would point us to the file `deployonubuntu.yml`, provided we were running on a Debian system. The variable `ansible_os_family` is defined by `gather_facts: yes`

```
student@bchd:~/ans/roles$ cat rzfeeser.ansible_role_minecraft/tasks/deployonubuntu.yml
```

14. Explore the rest of the role if you'd like.

15. Try installing another role from galaxy. This one allows users to Expand a Volume on a Dell PowerMax Flash Array via a WWN. *Note: Ansible still requires some augmentation to work with Dell EMC PowerMax, but once configured, this solution would work*

```
student@bchd:~/ans/roles$ ansible-galaxy install rzfeeser.dell_powermax_expand_vol_by_wwn
```

16. See that the role `rzfeeser.dell_powermax_exapand_vol_by_wwn` is there.

```
student@bchd:~/ans/roles$ ls -ll
```

17. Run tree on the new role.

```
student@bchd:~/ans/roles$ tree ~/ans/roles/rzfeeser.dell_powermax_expand_vol_by_wwn
```

18. If storage solutions are of interest to you, you can check out this role. Start by looking at `tasks/main.yml`. *Note: Other labs will focus on Ansible for storage applications. Therefore, if storage is of some interest, don't spend too much time examining this role now.*

19. The ansible-galaxy has a nice command line trick that allows us to spawn a new role template. Type the following.

```
student@bchd:~/ans/roles$ ansible-galaxy role init ~/ans/roles/my_role
```

20. Explore the various folders and standard role structure the ansible-galaxy template just created for you. In this case, our role is named, `my_role`. Run tree to see the standard structure for a role.

```
student@bchd:~/ans/roles$ tree ~/ans/roles/my_role
```

21. Answer the following questions:

    - **Q: Must all roles be posted to galaxy.ansible.com?**
        - *A: No. But if you don't use galaxy.ansible.com, you should use some sort of internal SCM, like GitHub, GitLab or BitBucket.*
    - **Q: How do I install the program ansible-galaxy?**
        - *A: It installs when you install ansible.*

22. Great job! You have completed this lab.

23. If you are tracking your code on GitHub, take a moment and perform a commit.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about the ansible galaxy tool"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

25. Ansible Collections

Lab Objective

New in Ansible v2.9, collections are the future of content delivery for Ansible. Traditionally, module creators have had to wait for their modules to be marked for inclusion in an upcoming Ansible release or had to add them to roles, which made consumption and management more difficult. In short, as devs we were kind of misusing roles because Ansible didn't offer an easy way to allow consumers to add new functionality to Ansible. With Ansible collections, pertinent roles and documentation, creators now have a chance of keeping up with demand. For example, a public cloud provider can offer new functionality of an existing service or a new service altogether, along with offering consumers an easy way to update Ansible allowing automation of the new functionality.

For the automation consumer, this means that fresh content is continuously made available for consumption. Managing content in this manner also becomes easier as modules, plugins, roles, and docs are packaged and tagged with a collection version. Modules can be updated, renamed, improved upon; roles can be updated to reflect changes in module interaction; docs can be regenerated to reflect the edits and all are packaged and tagged together.

Review an example structure of a collection before proceeding with this lab https://github.com/bcoca/collection

Procedure

1. Let's start by exploring what it takes to build a collection.

```
student@bchd:~$ mkdir -p ~/ans/collections/
```

2. Move into the new directory

```
student@bchd:~/ans/collections$ cd ~/ans/collections
```

3. The application tree has nothing to do with Ansible, but will let us "see" directory structures easier. Install tree now.

```
student@bchd:~/ans/collections$ sudo apt install tree -y
```

4. Just like a role, the collection skeleton can be started with the `ansible-galaxy` command. If you'd like, replace `adalovelace` with your name.

```
student@bchd:~/ans/collections$ ansible-galaxy collection init adalovelace.myfirstcollection
```

5. In the above command, `adalovelace` is a "namespace". Don't read into it too much, it ends up just being a folder named adalovelace that is used to keep things organized. The collection itself will be called `myfirstcollection`.

6. Run tree so we can see what just happened.

```
student@bchd:~/ans/collections$ tree
```

7. Here is a short rundown of each file and folder within a collection. *Note: This is a bit of a moving target. If you see additional files and folders, or fewer, "Don't panic!" Collections are an evolving concept. We expect this tool to generate different content over the next few months or years*:

    - README.md: What is this collection about
    - galaxy.yml: Contains metadata about your collection
    - docs/: local documentation for the collection
    - playbooks/: playbooks reside here
        - playbooks/tasks/: this holds 'task list files' for include_tasks/import_tasks usage
    - plugins/: all ansible plugins and modules go here, each in its own subdir
        - (example) plugins/modules/: ansible modules
        - (example) plugins/lookups/: lookup plugins
        - (example) plugins/filters/: Jinja2 filter plugins
        - plugins/... rest of plugins
        - README.md: a description of the new functionality provided by the plugins
    - roles/: directory for ansible roles

8. A few of those files mentioned didn't appear when we ran the `ansible-galaxy` command, and that's OK. You can add directories as necessary. A good reference for how you might structure an Ansible collection can be found here https://github.com/bcoca/collection

9. Collections are a flexible concept. They're focus is **content delivery**. That might mean playbooks, plugins, roles, or more.

10. In a previous lab you might have written your own Ansible role. If you have, try coping it into the `roles/` folder... and you're done! Yep, a collection is that easy! The following command is an example of how to copy the role `alta3.webservice` into our new collection, re-titling it `webservice`. *Note: Roles in collections should not have dot notation*. If you didn't do this lab, go do it now before you go any further. Alternatively, if you titled your role differently, you may need to tweak the command below.

```
student@bchd:~/ans/collections$ cp -r ~/ans/roles/alta3.webservice ~/ans/collections/adalovelace/myfirstcollection/roles/
webservice
```

11. *Note: A collection may contain multiple roles.*

12. Great. Now the role `webservice` only exists within our collection.

13. Let's package an example playbook with our collection as well. Create a playbook directory.

```
student@bchd:~/ans/collections$ mkdir ~/ans/collections/adalovelace/myfirstcollection/playbook/
```

14. Create an example playbook.

```
student@bchd:~/ans/collections$ vim ~/ans/collections/adalovelace/myfirstcollection/playbook/playbook-example01.yml
```

15. Great. Create the following playbook

```
---
- name: use role distributed with collection
  hosts: new_servers       # host we want to run the collection against
  connection: ssh  # default connection type
  gather_facts: no # no need to collect ansible_facts
  become: yes

  # this is the collection we want to make part of this playbook namespace
  collections:
          - adalovelace.myfirstcollection

  # this role is within the collection adalovelace.myfirstcollection
  roles:
          - webservice

  tasks:
          - name: Tasks run after roles
            debug:
                    msg: "Great job! You used a role within the collection adalovelace.myfirstcollection"
```

16. One you've put your collection together, it is time to `build` the collection. This must be done from the folder containing `galaxy.yml`. This process creates a tarball and compresses the file. The command is always run in the relative root of your collection.

```
student@bchd:~/ans/collections$ cd ~/ans/collections/adalovelace/myfirstcollection/
```

17. Now run the `build` command to create the collection.

```
student@bchd:~/ans/collections$ ansible-galaxy collection build --output-path ~/ans
```

18. The result is a version controlled tar ball (1.0.0.tar.gz). This version is a reflection of the value found within `galaxy.yml`. This tar ball is mainly intended to upload to Galaxy as a distribution method, but you can use it directly to install the collection on target systems.

> It is worth noting, that if you see anything related to the now-deprecated `Mazer tool` for any of your collections, delete any and all files it added to your releases/ directory before you build your collection with ansible-galaxy.

19. Move into the `~/ans` directory.

```
student@bchd:~/ans/collections$ cd ~/ans
```

20. Confirm your collection is present. It will end in 1.0.0.tar.gz.

```
student@bchd:~/ans$ ls
```

21. Try installing your collection to make it usable to the system. The command below will install it to one of the configured collection directories (defined in `~/.ansible.cfg`).

```
student@bchd:~/ans$ ansible-galaxy collection install adalovelace-myfirstcollection-1.0.0.tar.gz
```

22. If you want to install your collection to a target directory, you can use `-p /path/to/collection/directory/`. If your playbook is adjacent to the directory, it will find the collection, otherwise, it is necessary to update the collection search paths by editing `~/.ansible.cfg` and adding the new directory to `collections_paths`.

23. Great! Let's make a playbook.

```
student@bchd:~/ans$ vim ~/ans/collection01-playbook.yaml
```

24. Create the following playbook (very close to the example playbook we included with the collection).

```
- name: use role distributed with collection
  hosts: fry       # host we want to run the collection against
  connection: ssh  # default connection type
  gather_facts: no # no need to collect ansible_facts
  become: yes      # run as sudo

  # this is the collection we want to make part of this playbook namespace
  collections:
          - adalovelace.myfirstcollection

  # this role is within the collection adalovelace.myfirstcollection
  roles:
          - webservice

  tasks:
          - name: Tasks run after roles
            debug:
                    msg: "Great job! You used a role within the collection adalovelace.myfirstcollection"
```

25. Save and exit with `:wq`.

26. Try running the playbook.

```
student@bchd:~/ans$ ansible-playbook collection01-playbook.yaml
```

27. Once a collection has been installed, you can reference any of its components using a Fully Qualified Collection Name (FQCN). **Read-only the following example.**

```
- name: Use collection components without using the keyword "collection"
  hosts: all

  tasks:
    - my_namespace.my_collection.mymodule:
        option1: value
```

28. This technique works for roles or any type of plugin distributed within the collection. **Read-only the following example.**

```
- name: Using collection components

  hosts: all
  tasks:
    # use role1 from adalovelace.myfirstcollection
    - import_role:
        name: adalovelace.myfirstcollection.role1

    # use a custom module "mymodule" located in namespace buzzaldrin within the collection apollo
    - name: Use the custom "mymodule"
      buzzaldrin.apollo.mymodule:
        option1: value

    - name: Use custom lookup and filter plugins from the collection
      debug:
        msg: '{{ lookup("adalovelace.myfirstcollection.lookup1", 'param1')| adalovelace.myfirstcollection.filter1 }}'
```

29. Using the `collections` keyword simply prevents excessive typing. **Read-only the following example.**

```
- name: Using the collections keyword
  hosts: all

  # multiple collections may be imported
  collections:
   - adalovelace.myfirstcollection
   - buddaldrin.apollo

  tasks:
    # use role1 from adalovelace.myfirstcollection
    - import_role:
        name: role1

    # use a custom module "mymodule" located in namespace buzzaldrin within the collection apollo
    - name: Use the custom "mymodule" from within the buzzaldrin namespace
      mymodule:
        option1: value

    - name: You still need to use FQCN for non-action plugins
      debug:
        msg: '{{ lookup("adalovelace.myfirstcollection.lookup1", 'param1')| adalovelace.myfirstcollection.filter1 }}'
```

30. The "collections" keyword creates a search path for non namespaced plugin references. It does not import roles or anything else. Notice that you still need the FQCN for non-action or module plugins.

31. If you'd like to publish your collection to Ansible Galaxy, you can do so with `ansible-galaxy collection publish`. Once you publish a version to Galaxy, it cannot be changed. So, make sure you're careful about what you publish. For more information on publishing a collection to Galaxy, see https://docs.ansible.com/ansible/latest/dev_guide/developing_collections.html#publishing-collections

32. **CHALLENGE 01 (OPTIONAL)** - Try building a collection for the custom modules available on https://github.com/rzfeeser/ansible-custom-modules-nasa-api

33. **SOLUTION 01** - Create a collection with `collection init`. Clone the repository. Copy the library files into the collection at `plugins/modules/`. Perform a `collection build` and then a `collection install` to access the collection.

34. Answer the following questions:

    - **Q: What are collections?**
      - *A: Collections make it easy to distribute content. Dell could release a new product, and on the same day, release a collection that allows Ansible to support that new product. Collections make it easy to augment your ansible controller.*
    - **Q: Will I need to make collections?**
      - *A: Not necessarily. It is important to understand how they work, and how to use them, but you may never need to actually build one. Remember, they are for content delivery. The question is similar to, "Do I have to write libraries in Python to be successful*

with python?" Of course the answer to that question is "No." You can go very far with Python without ever designing a library for import.

35. Great job! You have completed this lab.

36. If you are tracking your code on GitHub, take a moment and perform a commit.

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about collections"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

26. Ansible for Dell EMC PowerMax Storage

Lab Objective

*This lab is a **case study**, and is designed to reflect acquired knowledge. The lab environments do not include targets for this lab, or access to the equipment may lay outside the current environment. Reading through the lab can still offer valuable information on using Ansible for Dell EMC PowerMax Storage.*

The objective of this lab to learn how to automate the Dell PowerMax storage array with Ansible.

The following links will help you get started augmenting your Ansible controller to support Dell EMC PowerMax Storage arrays.

Dell EMC PowerMax Collection: https://galaxy.ansible.com/dellemc/powermax

Dell EMC PowerMax GitHub Repo:
https://github.com/dell/ansible-powermax/

Dell EMC PowerMax Documentation:
https://github.com/dell/ansible-powermax/tree/master/dellemc_ansible/docs

The PyU4V Python Client:
https://pyu4v.readthedocs.io/en/latest/

Procedure

1. Read through the above comments. When you're ready, you'll need to install the following packages on the Ansible controller.
2. Install the following client to interact with Unisphere. It is actually important that you stick to the version referenced in the installation guide, despite it being woefully behind the client currently available (at the time of writing the install guide said to install version 3.1.5 when version 9.4+ was available).

```
student@bchd:~$ python3 -m pip install PyU4V==3.1.5
```

3. With Ansible 2.10+, we are encouraged to augment the Ansible controller via collections. Read about that now https://galaxy.ansible.com/dellemc/powermax
4. Install the collection now.

```
student@bchd:~$ ansible-galaxy collection install dellemc.powermax -c
```

5. Use the following command to learn a bit more about where ansible is installed.

```
student@bchd:~$ ansible --version
```

6. Create the `~/ans/` and `~/ans/vars/` folders in the correct location for your system.

```
student@bchd:~$ mkdir -p ~/ans/vars/
```

7. Create a file for the credentials.

```
student@bchd:~$ vim ~/ans/vars/credsvault.yml
```

8. Place the following information in the new file.

```
---
dell:
        unispherehost: 10.126.70.28
        universion: "90"
        verifycert: no
        user: smc
        password: smc
```

9. Save and exit with `:wq`

10. Create a playbook to communicate with the Dell EMC Storage Array.

```
student@bchd:~$ vim ~/ans/playbook-dellpowermax01.yml
```

11. Create the following:

```
---
- name: PowerMax Dell Playbook v1
  hosts: localhost
  gather_facts: no

  collections:
    - dellemc.powermax

  tasks:
  - name: Try running our new gather_facts module
    dellemc_powermax_gatherfacts:
      unispherehost: 10.126.70.19
      universion: "90"
      verifycert: False
      user: smc
      password: smc
    register: results

  - name: what JSON was returned
    debug:
      var: results

  - name: Provide serial number for next gather facts
    dellemc_powermax_gatherfacts:
      unispherehost: 10.126.70.19
      universion: "90"
      verifycert: False
      user: smc
      password: smc
      serial_no: "{{ results.Arrays[0] }}"
      gather_subset:
        - sg
        - srp
        - pg
        - host
    register: arrayresults

  - name: dump out to the screen
    debug:
      var: arrayresults
```

12. Save and exit with `:wq`

13. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dellpowermax01.yml
```

14. **OPTIONAL STEP** - If you get an error such as, `libselinux-python not installed`, try the next command. It helped me overcome the issue when running Ansible in a CentOS 6 environment with a python3 installation.

```
student@bchd:~$ python3 -m pip install selinux
```

15. Write a new playbook that places the information in an external file:

```
student@bchd:~$ vim ~/ans/playbook-dellpowermax02.yml
```

16. Create the following solution:

```
---
- name: Gather configuration data
  hosts: localhost
  connection: local

  collections:
    - dellemc.powermax

  tasks:
  - name: Get a list of arrays (serial_no.)
    dellemc_powermax_gatherfacts:
        unispherehost: 10.126.70.28
        universion: "90"
        verifycert: no
        user: smc
        password: smc
    register: arrays

  - name: Display the data that was just gathered
    debug:
            var: arrays
            verbosity: 1

  - name: Pull all available facts
    dellemc_powermax_gatherfacts:
        unispherehost: 10.126.70.28
        universion: "90"
        verifycert: no
        user: smc
        password: smc
        serial_no: "{{ item }}" # maps to an array (avail if you run the module without it)
        gather_subset: [ vol, srp, sg, pg, host, hg, port, mv ]
    loop: "{{ arrays.Arrays }}"
    register: results

  - name: Ensure a directory exists to store a fact report inside
    file:
            dest: reporting/
            state: directory

  - name: Create a local JSON file with current fact statistics
    copy:
            dest: "reporting/{{ ansible_date_time.date }}-powermax_facts.json"
            content: "{{ results|to_json }}"
```

17. Save and exit with `:wq`

18. Run the new playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dellpowermax02.yml
```

19. With this next run, we'll use the credential file we created earlier.

```
student@bchd:~$ vim ~/ans/playbook-dellpowermax03.yml
```

20. This next file will read in variables stored within, `vars/credsvault.yml`.

```
---
- name: Gather configuration data
  hosts: localhost
  connection: local

  collections:
    - dellemc.powermax

  vars_files:
    - vars/credsvault.yml

  tasks:
  - name: Get a list of arrays (serial_no.)
    dellemc_powermax_gatherfacts:
        unispherehost: "{{ dell.unispherehost }}"
        universion: "{{ dell.universion }}"
        verifycert: "{{ dell.verifycert }}"
        user: "{{ dell.user }}"
        password: "{{ dell.password }}"
    register: arrays

  - name: Display the data that was just gathered
    debug:
            var: arrays
            verbosity: 1

  - name: Pull all available facts
    dellemc_powermax_gatherfacts:
        unispherehost: "{{ dell.unispherehost }}"
        universion: "{{ dell.universion }}"
        verifycert: "{{ dell.verifycert }}"
        user: "{{ dell.user }}"
        password: "{{ dell.password }}"
        serial_no: "{{ item }}" # maps to an array (avail if you run the module without it)
        gather_subset: [ vol, srp, sg, pg, host, hg, port, mv ]
    loop: "{{ arrays.Arrays }}"
    register: results

  - name: Ensure a directory exists to store a fact report inside
    file:
            dest: reporting/
            state: directory

  - name: Create a local JSON file with current fact statistics
    copy:
            dest: "reporting/{{ ansible_date_time.date }}-powermax_facts.json"
            content: "{{ results|to_json }}"
```

21. Save and exit with `:wq`

22. Try running the new playbook

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dellpowermax03.yml
```

23. If that worked, then we should encrypt the credential file with `ansible-vault`.

```
student@bchd:~$ ansible-vault encrypt ~/ans/vars/credsvault.yml --ask-vault-pass
```

24. Use the password `alta3`

25. Try running the playbook again, be sure to include the `--ask-vault-pass` flag to let Ansible know you have a password to decrypt the credentials. *Note: In production, it is possible to use a key file in place of a string password when working with ansible-vault*

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dellpowermax03.yml --ask-vault-pass
```

26. Provide the password `alta3`
27. The playbook should execute the same as it did previously.
28. In this next playbook, let's create a playbook that does some work (makes some changes). To begin with, create a variable file we can store some dynamic values in.

```
student@bchd:~$ vim ~/ans/vars/storagedetails.yml
```

29. Create the following variable file.

```
---
storage:
        sg_name: ansible_sg
        vols:
                - vol_name: ansible_001
                  size: 1
                  cap_unit: GB
                - vol_name: ansible_002
                  size: 2
                  cap_unit: GB
                - vol_name: ansible_003
                  size: 3
                  cap_unit: GB
        initiators:
                - 10000000abcffaee
                - 10000000defaabcc
        host_name: ansible_host
        portgroup_name: ansible_pg
        port_list:
                - director_id: FA-1D
                  port_id: 24
        mv_name: ansible_mv
```

30. Save and exit with `:wq`
31. Now for a playbook that will provision some storage.

```
student@bchd:~$ vim ~/ans/playbook-dellpowermax04.yml
```

32. The playbook is mostly self-describing. Review the following playbook. Look for ways each module interacts with the next.

```
---
- name: Gather configuration data
  hosts: localhost
  connection: local

  collections:
    - dellemc.powermax

  vars_files:
    - vars/credsvault.yml
    - vars/storagedetails.yml

  tasks:
  - name: Get a list of arrays (serial_no.)
    dellemc_powermax_gatherfacts:
        unispherehost: "{{ dell.unispherehost }}"
        universion: "{{ dell.universion }}"
        verifycert: "{{ dell.verifycert }}"
        user: "{{ dell.user }}"
        password: "{{ dell.password }}"
    register: arrays

  - name: Display the data that was just gathered
    debug:
            var: arrays
            verbosity: 1

  - name: Create a storage group
    dellemc_powermax_storagegroup:
        unispherehost: "{{ dell.unispherehost }}"
        universion: "{{ dell.universion }}"
        verifycert: "{{ dell.verifycert }}"
        user: "{{ dell.user }}"
        password: "{{ dell.password }}"
        serial_no: "{{ arrays.Arrays[0] }}" # maps to an array (avail if you run the module without it)
        service_level: "bronze"
        state: "present"
        sg_name: "{{ storage.sg_name }}"

  - name: Create a volume for storage group
    dellemc_powermax_storagegroup:
        unispherehost: "{{ dell.unispherehost }}"
        universion: "{{ dell.universion }}"
        verifycert: "{{ dell.verifycert }}"
        user: "{{ dell.user }}"
        password: "{{ dell.password }}"
        serial_no: "{{ arrays.Arrays[0] }}" # maps to an array (avail if you run the module without it)
        service_level: "bronze"
        state: "present"
        sg_name: "{{ storage.sg_name }}"
        volumes: "{{ storage.vols }}" # this is a list
        vol_state: "present-in-group"

  - name: "Create host {{ storage.host_name }}"
    dellemc_powermax_host:
        unispherehost: "{{ dell.unispherehost }}"
        universion: "{{ dell.universion }}"
        verifycert: "{{ dell.verifycert }}"
        user: "{{ dell.user }}"
        password: "{{ dell.password }}"
```

```
        serial_no: "{{ arrays.Arrays[0] }}" # maps to an array (avail if you run the module without it)
        host_name: "{{ storage.host_name }}"
        initiators: "{{ storage.initiators }}"
        state: present
        initiator_state: "present-in-host"

    - name: "Create port group {{ storage.portgroup_name }}"
      dellemc_powermax_portgroup:
        unispherehost: "{{ dell.unispherehost }}"
        universion: "{{ dell.universion }}"
        verifycert: "{{ dell.verifycert }}"
        user: "{{ dell.user }}"
        password: "{{ dell.password }}"
        serial_no: "{{ arrays.Arrays[0] }}" # maps to an array (avail if you run the module without it)
        portgroup_name: "{{ storage.portgroup_name }}"
        state: present
        ports: "{{ storage.port_list }}"
        port_state: "present-in-group"

    - name: "Create masking view {{ storage.mv_name }}"
      dellemc_powermax_maskingview:
        unispherehost: "{{ dell.unispherehost }}"
        universion: "{{ dell.universion }}"
        verifycert: "{{ dell.verifycert }}"
        user: "{{ dell.user }}"
        password: "{{ dell.password }}"
        serial_no: "{{ arrays.Arrays[0] }}" # maps to an array (avail if you run the module without it)
        mv_name: "{{ storage.mv_name }}"
        portgroup_name: "{{ storage.portgroup_name }}"
        host_name: "{{ storage.host_name }}"
        sg_name: "{{ storage.sg_name }}"
        state: present
```

33. Save and exit with `:wq`

34. Run the playbook solution

```
student@bchd:~$ vim ~/ans/playbook-dellpowermax04.yml --ask-vault-pass
```

35. Provide the password `alta3`

36. Great job! That's it for this lab.

27. Ansible for Dell EMC PowerScale (Isilon)

Lab Objective

*This lab is a **case study**, and is designed to reflect acquired knowledge. The lab environments do not include targets for this lab, or access to the equipment may lay outside the current environment. Reading through the lab can still offer valuable information on using Ansible for Dell EMC Isilon Storage.*

Dell EMC updated Isilon with the name PowerScale. You'll find references to both names, however, both names reference the same product.

The objective of this lab to learn how to automate the Dell EMC PowerScale (Isilon) storage array with Ansible.

The following links will help you get started augmenting your Ansible controller to support Dell EMC PowerScale (Isilon) Storage arrays.

Dell EMC PowerScale (Isilon) GitHub Repo:
https://github.com/dell/ansible-isilon

Dell EMC PowerScale (Isilon) Documentation:
https://github.com/dell/ansible-isilon/blob/master/dellemc_ansible/docs/

The isi-sdk Python Client:
https://pypi.org/project/isi-sdk-8-1-1/

Example PowerMax, PowerScale (Isilon), PowerStore, VPLEX, (and more) Playbooks: https://github.com/dell/ansible-storage-automation

Procedure

1. Read through the above comments. When you're ready, you'll need to install the following packages on the Ansible controller.
2. Install the following client to interact with Dell EMC PowerScale (Isilon). It is actually important that you stick to the version referenced in the installation guide.

```
student@bchd:~$ python3 -m pip install isi_sdk_8_1_1
```

3. With Ansible 2.10+, we are encouraged to augment the Ansible controller via collections. Read about that now https://galaxy.ansible.com/dellemc/isilon
4. Install the collection now.

```
student@bchd:~$ ansible-galaxy collection install dellemc.isilon -c
```

5. You could also use the following command to learn a bit more about where ansible is installed.

```
student@bchd:~$ ansible --version
```

6. Create the `~/ans/` and `~/ans/vars/` folders in the correct location for your system.

```
student@bchd:~$ mkdir -p ~/ans/vars/
```

7. Create a file for the credentials.

```
student@bchd:~$ vim ~/ans/vars/credsvault-isi.yml
```

8. Place the following information in the new file.

```
---
# credentials for isilon
onefs_host: 192.168.3.11      # IP address of FQDN of the Isilon cluster
isilon_port: 8080             # defaults to 8080
api_user: root                # mandatory - username for the REST API
api_password: "password1234"  # mandatory - password for the REST API
verify_ssl: False             # mandatory - bool
```

9. Save and exit with `:wq`
10. Create a playbook to communicate with the Dell EMC Storage Array.

```
student@bchd:~$ vim ~/ans/playbook-dellisilon01.yml
```

11. Create the following:

```
---
- name: Isilon gather facts
  hosts: localhost
  connection: local

  # access the dellemc.isilon namespace
  collections:
          - dellemc.isilon

  vars_files:
          - vars/credsvault-isi.yml

  tasks:
          - name: Gather facts on Isilon cluster
            dellemc_isilon_gatherfacts:
                    onefs_host: "{{ onefs_host }}"
                    port_no: "{{ isilon_port }}"
                    verify_ssl: "{{ verify_ssl }}"
                    api_user: "{{ api_user }}"
                    api_password: "{{ api_password }}"
                    gather_subset:
                            - attributes
                            - access_zones
                            - nodes
                            - providers
                            - users
                            - groups
            register: results

          # display what was just collected
          - name: Debug the results
            debug:
                    var: results
```

12. Save and exit with `:wq`
13. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dellisilon01.yml
```

14. If that worked, then we should encrypt the credential file with `ansible-vault`.

```
student@bchd:~$ ansible-vault  encrypt ~/ans/vars/credsvault-isi.yml --ask-vault-pass
```

15. Use the password `alta3`
16. Try running the playbook again, be sure to include the `--ask-vault-pass` flag to let Ansible know you have a password to decrypt the credentials. *Note: In production, it is possible to use a key file in place of a string password when working with ansible-vault.*

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dellisilon01.yml --ask-vault-pass
```

17. Provide the password `alta3`
18. The playbook should execute the same as it did previously.
19. Let's try doing some work. To begin, create a new variable file with some data.

```
student@bchd:~$ vim ~/ans/vars/storagedetails-isi.yml
```

20. Place the following in the new file.

```
# vars for isilon access
---
base_path: "/ifs/globaluni"
access_control: "0755"
access_zone: "System"
domain: "alta3.com"
state: "present"

students:
        - peterParker
        - lukeCage
        - natashaRomanoff
        - wandaMaximoff
        - scottLang

teachers:
        - tonyStark
        - steveRogers
        - thorOdinson
        - bruceBanner
        - carolDanvers
```

21. Save and exit with `:wq`

22. Write a new playbook that places the information in an external file:

```
student@bchd:~$ vim ~/ans/playbook-dellisilon02.yml
```

23. Create the following solution:

```
---
- name: Create Unix home directory from storagedetails-isi.yml data in PowerScale/Isilon
  hosts: localhost
  connection: local

  collections:
          - dellemc.isilon

  vars_files:
          - vars/credsvault-isi.yml
          - vars/storagedetails-isi.yml

  tasks:
    - name: Create Filesystem with Quota for students
      dellemc_isilon_filesystem:
              onefs_host: "{{ onefs_host }}"
              port_no: "{{ isilon_port }}"
              verify_ssl: "{{ verify_ssl }}"
              api_user: "{{ api_user }}"
              api_password: "{{ api_password }}"
              path: "{{base_path}}/students/{{item}}"
              owner:
                      name: "root"
                      provider_type: 'file'
              access_control: "{{ access_control }}"
              # quota:  # quota is a featured supported by some ilison licenses
              #         include_snap_data: False
              #         include_data_protection_overhead: False
              #         #advisory_limit_size: 2
              #         soft_limit_size: 5
              #         hard_limit_size: 10
              #         cap_unit: 'GB'
              #         quota_state: 'present'
              recursive: True
              state: "{{ state }}"
      loop: "{{ students }}"


    - name: Create Filesystem with Quota for Teachers
      dellemc_isilon_filesystem:
              onefs_host: "{{ onefs_host }}"
              port_no: "{{ isilon_port }}"
              verify_ssl: "{{ verify_ssl }}"
              api_user: "{{ api_user }}"
              api_password: "{{ api_password }}"
              path: "{{base_path}}/teachers/{{item}}"
              owner:
                      name: "root"
                      provider_type: 'file'
              access_control: "{{ access_control }}"
              #quota:
              #        hard_limit_size: 100
              #        cap_unit: 'GB'
              #        quota_state: 'present'
              recursive: True
              state: "{{ state }}"
      loop: "{{ teachers }}"
```

24. Save and exit with `:wq`

25. Run the new playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dellisilon02.yml --ask-vault-pass
```

26. Provide the password `alta3`

27. Great! All of our storage has been provisioned. Now lets **remove** that provisioning. Use the same command, only provide an extra variable via `-e state=absent`

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dellisilon02.yml --ask-vault-pass -e state=absent
```

28. Provide the password `alta3`

29. When the playbook finishes, all of the provisioned storage should be removed.

30. Great. Now that we have a playbook interacting with the Dell EMC platform that is able to apply, and remove, a change, let's write a playbook that uses **prechecks** to ensure we're about to operate on the "correct" platform. *Note: We are trying to emulate the way a human engineer works within an enterprise*

```
student@bchd:~$ vim ~/ans/playbook-dellisilon03.yml
```

31. The following playbook has three (3) precheck conditions. The first, the playbook will fail (and exit) when there are not **exactly** 4 devices enrolled within the array. The second, ensures that the version running is 8.2.1.0. The third, ensures there are no current failures on the isilon array. If any changes fail, the changes will be rolled back. Create the following:

```
---
- name: Create Unix home directory from storagedetails-isi.yml data in PowerScale/Isilon
  hosts: localhost
  connection: local
  gather_facts: no

  collections:
          - dellemc.isilon

  vars_files:
          - vars/credsvault-isi.yml
          - vars/storagedetails-isi.yml

  tasks:

    - name: Precheck Operations - Isilon / PowerScale
      block:
              - name: Gather facts on Isilon cluster
                dellemc_isilon_gatherfacts:
                        onefs_host: "{{ onefs_host }}"
                        port_no: "{{ isilon_port }}"
                        verify_ssl: "{{ verify_ssl }}"
                        api_user: "{{ api_user }}"
                        api_password: "{{ api_password }}"
                        gather_subset:
                                - attributes
                register: results


              - name: PRECHECK - Correct number of devices enrolled
                fail:
                        msg: "Precheck fail."
                when: results.Attributes.Config.devices|length != 4


              - name: PRECHECK - Ensure correct release version running on Cluster
                fail:
                        msg: "Precheck failure."
                when: item.release != "v8.2.1.0"
                loop: "{{ results.Attributes.Cluster_Version.nodes }}"
                loop_control:
                        label: "{{ item.release }}"


              - name: PRECHECK - Ensure no current failures on the Isilon / Powerscale Cluster
                fail:
                        msg: "Precheck failure."
                when: results.Attributes.Cluster_Version.errors | length != 0


    - name: Change Operation
      block:
            - name: CHANGE - Create Filesystem with Quota for students
              dellemc_isilon_filesystem:
                      onefs_host: "{{ onefs_host }}"
                      port_no: "{{ isilon_port }}"
                      verify_ssl: "{{ verify_ssl }}"
                      api_user: "{{ api_user }}"
                      api_password: "{{ api_password }}"
                      path: "{{base_path}}/students/{{item}}"
```

```
                owner:
                        name: "root"
                        provider_type: 'file'
                access_control: "{{ access_control }}"
                recursive: True
                state: "{{ state }}"
          loop: "{{ students }}"


        - name: CHANGE - Create Filesystem with Quota for Teachers
          dellemc_isilon_filesystem:
                onefs_host: "{{ onefs_host }}"
                port_no: "{{ isilon_port }}"
                verify_ssl: "{{ verify_ssl }}"
                api_user: "{{ api_user }}"
                api_password: "{{ api_password }}"
                path: "{{base_path}}/teachers/{{item}}"
                owner:
                        name: "root"
                        provider_type: 'file'
                access_control: "{{ access_control }}"
                recursive: True
                state: "{{ state }}"
          loop: "{{ teachers }}"

        - name: CHANGE - End of Change operation
          debug:
                msg: Change operation SUCCESSFUL.

    rescue:
        - name: RESCUE - Start Rollback
          debug:
                  msg: An error was encountered. Attempting rollback.

        - name: RESCUE - Remove filesystem with Quota for students
          dellemc_isilon_filesystem:
                  onefs_host: "{{ onefs_host }}"
                  port_no: "{{ isilon_port }}"
                  verify_ssl: "{{ verify_ssl }}"
                  api_user: "{{ api_user }}"
                  api_password: "{{ api_password }}"
                  path: "{{base_path}}/students/{{item}}"
                  owner:
                          name: "root"
                          provider_type: 'file'
                  access_control: "{{ access_control }}"
                  recursive: True
                  state: "absent"
          loop: "{{ students }}"


        - name: RESCUE - Remove filesystem with Quota for Teachers
          dellemc_isilon_filesystem:
                  onefs_host: "{{ onefs_host }}"
                  port_no: "{{ isilon_port }}"
                  verify_ssl: "{{ verify_ssl }}"
                  api_user: "{{ api_user }}"
                  api_password: "{{ api_password }}"
                  path: "{{base_path}}/teachers/{{item}}"
                  owner:
```

```
                name: "root"
                provider_type: 'file'
            access_control: "{{ access_control }}"
            recursive: True
            state: "absent"
      loop: "{{ teachers }}"

    - name: RESCUE - End of Rollback
      debug:
            msg: Rollback SUCCESSFUL. No changes were made.
```

32. Save and exit with `:wq`
33. Run the new playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dellisilon03.yml --ask-vault-pass
```

34. Provide the password `alta3`
35. The playbook should execute without errors.
36. Once again, run the playbook to remove the provisioned storage.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dellisilon03.yml --ask-vault-pass -e state=absent
```

37. Provide the password `alta3`
38. The playbook should execute without errors.
39. Answer the following questions:
    - **Q: Why are prechecks important?**
        - *A: Before we get in a car, hop on a motorcycle, or fly a plane, the pilot should always do a walk-around for obvious mechanical failures, adjust mirrors, and review the location of all controls. These prechecks are designed to prevent errors, or mitigate errors should they occur. Same is true before we work on enterprise equipment. Prechecks are designed to help prevent, "Uh-oh" moments.*
    - **Q: What could cause a playbook error?**
        - *A: A poorly written playbook is the most likely culprit. However, it could be a well written playbook, with an undefined or wrongly defined variable. In short, lots.*
40. Create a final playbook that **simulates an error**.

```
student@bchd:~$ vim ~/ans/playbook-dellisilon04.yml
```

41. Create a final playbook that simulates an error. When this error is triggered, **the playbook will rollback** the changes we made.

```
---
- name: Create Unix home directory from storagedetails-isi.yml data in PowerScale/Isilon
  hosts: localhost
  connection: local
  gather_facts: no

  collections:
          - dellemc.isilon

  vars_files:
          - vars/credsvault-isi.yml
          - vars/storagedetails-isi.yml

  tasks:

    - name: Precheck Operations - Isilon / PowerScale
      block:
              - name: Gather facts on Isilon cluster
                dellemc_isilon_gatherfacts:
                        onefs_host: "{{ onefs_host }}"
                        port_no: "{{ isilon_port }}"
                        verify_ssl: "{{ verify_ssl }}"
                        api_user: "{{ api_user }}"
                        api_password: "{{ api_password }}"
                        gather_subset:
                                - attributes
                register: results


              - name: PRECHECK - Correct number of devices enrolled
                fail:
                        msg: "Precheck fail."
                when: results.Attributes.Config.devices|length != 4


              - name: PRECHECK - Ensure correct release version running on Cluster
                fail:
                        msg: "Precheck failure."
                when: item.release != "v8.2.1.0"
                loop: "{{ results.Attributes.Cluster_Version.nodes }}"
                loop_control:
                        label: "{{ item.release }}"


              - name: PRECHECK - Ensure no current failures on the Isilon / Powerscale Cluster
                fail:
                        msg: "Precheck failure."
                when: results.Attributes.Cluster_Version.errors | length != 0


    - name: Change Operation
      block:
            - name: CHANGE - Create Filesystem with Quota for students
              dellemc_isilon_filesystem:
                      onefs_host: "{{ onefs_host }}"
                      port_no: "{{ isilon_port }}"
                      verify_ssl: "{{ verify_ssl }}"
                      api_user: "{{ api_user }}"
                      api_password: "{{ api_password }}"
                      path: "{{base_path}}/students/{{item}}"
```

```
                owner:
                        name: "root"
                        provider_type: 'file'
                access_control: "{{ access_control }}"
                recursive: True
                state: "{{ state }}"
          loop: "{{ students }}"


        - name: CHANGE - Create Filesystem with Quota for Teachers
          dellemc_isilon_filesystem:
                onefs_host: "{{ onefs_host }}"
                port_no: "{{ isilon_port }}"
                verify_ssl: "{{ verify_ssl }}"
                api_user: "{{ api_user }}"
                api_password: "{{ api_password }}"
                path: "{{base_path}}/teachers/{{item}}"
                owner:
                        name: "root"
                        provider_type: 'file'
                access_control: "{{ access_control }}"
                recursive: True
                state: "{{ state }}"
          loop: "{{ teachers }}"

        - name: Force a failure
          shell:
                cmd: "/bin/false"

        - name: CHANGE - End of Change operation
          debug:
                msg: Change operation SUCCESSFUL.

    rescue:
        - name: RESCUE - Start Rollback
          debug:
                  msg: An error was encountered. Attempting rollback.

        - name: RESCUE - Remove filesystem with Quota for students
          dellemc_isilon_filesystem:
                  onefs_host: "{{ onefs_host }}"
                  port_no: "{{ isilon_port }}"
                  verify_ssl: "{{ verify_ssl }}"
                  api_user: "{{ api_user }}"
                  api_password: "{{ api_password }}"
                  path: "{{base_path}}/students/{{item}}"
                  owner:
                          name: "root"
                          provider_type: 'file'
                  access_control: "{{ access_control }}"
                  recursive: True
                  state: "absent"
          loop: "{{ students }}"


        - name: RESCUE - Remove filesystem with Quota for Teachers
          dellemc_isilon_filesystem:
                  onefs_host: "{{ onefs_host }}"
                  port_no: "{{ isilon_port }}"
                  verify_ssl: "{{ verify_ssl }}"
```

```
                api_user: "{{ api_user }}"
                api_password: "{{ api_password }}"
                path: "{{base_path}}/teachers/{{item}}"
                owner:
                        name: "root"
                        provider_type: 'file'
                access_control: "{{ access_control }}"
                recursive: True
                state: "absent"
          loop: "{{ teachers }}"

        - name: RESCUE - End of Rollback
          debug:
                msg: Rollback SUCCESSFUL. No changes were made.
```

42. Save and exit with `:wq`

43. When you run the playbook, **it will fail, and then rollback the changes it is attempting to make**.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-dellisilon04.yml --ask-vault-pass -e state=absent
```

44. Answer the following questions:

    - **Q: Why are rollbacks important?**
        - *A: We want to write automation that tries to do work, and if it is not successful, puts things back the way they were. This is the way a human would work, it is the way we want to write our code.*
    - **Q: How can this technique be expanded?**
        - *A: By using 'import_playbook' or 'import_role' within a block / rescue condition (like above) you could achieve a "cleaner" solution.*

45. Great! That's it for this lab.

28. Ansible for Dell EMC Unity

Lab Objective

*This lab is a **case study**, and is designed to reflect acquired knowledge. The lab environments do not include targets for this lab, or access to the equipment may lay outside the current environment. Reading through the lab can still offer valuable information on using Ansible for Dell EMC Unity.*

The objective of this lab to learn how to automate the Dell EMC Unity with Ansible.

There is no current Unity collection. To use Unity, we will need to clone the git repository, then edit the ansible.cfg file to point to the new library: https://github.com/dell/ansible-unity

The SDK client used with Dell EMC Unity is called storops. The entry in pypi is: https://pypi.org/project/storops/

Procedure

1. Read through the above comments. When you're ready, start in your home folder.

```
student@bchd:~$ cd
```

2. We need to install the Python SDK for Dell EMC Unity on the Ansible controller. Use pip to do that now.

```
student@bchd:~$ python3 -m pip install storops
```

3. No collection on this product, so we'll need to do a bit more work to get Ansible to use our new modules. Clone the git repo containing the new Ansible modules. These are contained in a folder called `library`.

```
student@bchd:~$ git clone https://github.com/dell/ansible-unity
```

4. Create the `~/ans/` and `~/ans/vars/` folders in the correct location for your system.

```
student@bchd:~$ mkdir -p ~/ans/vars/
```

5. Create an ansible.cfg file with (at least) the following settings. This will point ansible to the location of the new modules we just downloaded. *Note: This is additional library and utility locations, and will not nullify the other default search locations for modules and utilities that may be installed with Ansible. Additional library or utility locations may be added with the colon used as a separator.*

```
[defaults]
# location of our new modules
library = ~/ansible-unity/dellemc_ansible/unity/library
# location of utilities required to use new modules
module_utils    = ~/ansible-unity/dellemc_ansible/utils/
```

6. Save and exit with `:wq`

7. Create a file for the credentials.

```
student@bchd:~$ vim ~/ans/vars/credsvault-unity.yml
```

8. Place the following information in the new file. In production, you'll need to replace the following information with your own hostname, username and password.

```
---
# unity credentials
unispherehost: 10.10.4.9
username: admin
password: qwerty123
verifycert: false
```

9. Save and exit with `:wq`

10. Create a playbook to communicate with the NetApp Storage Array.

```
student@bchd:~$ vim ~/ans/playbook-unity01.yml
```

11. Create the following:

```
---
- name: Connect to Dell EMC Unity
  hosts: localhost
  connection: local
  gather_facts: no

  vars_files:
          - vars/credsvault-unity.yml

  tasks:
          - name: Get information of Unity array.
            dellemc_unity_gatherfacts:
              unispherehost: "{{unispherehost}}"
              username: "{{username}}"
              password: "{{password}}"
              verifycert: "{{verifycert}}"
            register: results

          - name: display facts
            debug:
              var: results
```

12. Save and exit with `:wq`

13. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-unity01.yml
```

14. If that worked, then we should encrypt the credential file with `ansible-vault`.

```
student@bchd:~$ ansible-vault encrypt ~/ans/vars/credsvault-unity.yml --ask-vault-pass
```

15. Use the password `alta3`

16. Try running the playbook again, be sure to include the `--ask-vault-pass` flag to let Ansible know you have a password to decrypt the credentials. *Note: In production, it is possible to use a key file in place of a string password when working with ansible-vault.*

```
student@bchd:~$ ansible-playbook ~/ans/playbook-unity01.yml --ask-vault-pass
```

17. Provide the password `alta3`

18. The playbook should execute the same as it did previously.

19. A list of current Dell EMC Unity modules are available on the github page (click the "Content" button) https://github.com/dell/ansible-unity/tree/master/dellemc_ansible/unity/library

20. Great! That's it for this lab.

29. Ansible and Dell ECS S3

Lab Objective

The Dell EMC ECS S3 service mimics the Amazon AWS API. The similarities are such that the `amazon.aws.aws_s3` module not only works, but is the recommended module to use when automating the Dell ECS S3 API. This lab outlines how to automate the Dell ECS S3 with the `aws_s3` module.

Just as with the other Amazon modules, you'll need to install boto3 with `python3 -m pip install boto3`

Lab Procedure

1. On your Ansible controller, we can place those keys within a file that you won't accidentally commit to a git repo. Your home directory is fine.

   ```
   vim ~/cred.ecs_s3
   ```

2. Fill out the credential file as follows:

   ```
   ---
   access_key: "--REMOVED--"     # place your Access Key ID here
   secret_key: "--REMOVED--"     # place your Secret Key here
   ```

3. Save and exit with `:wq`

4. **IMPORTANT** Without wasting any time, we want to encrypt the `~/cred.zon` file with `ansible-vault`

   ```
   ansible-vault encrypt ~/cred.ecs_s3 --ask-vault-pass
   ```

5. Make up a password that you won't forget. You'll need it in a moment.

6. Install the latest collection.

   ```
   ansible-galaxy install amazon.aws
   ```

7. Take a peek at the module we will focus on for using against the DellEMC ECS S3 API.

   ```
   ansible-doc amazon.aws.aws_s3
   ```

8. Looks like in order to use it, we need the `boto3` client installed. Install that now.

   ```
   python3 -m pip install boto3
   ```

9. Great! All that is left is to write a playbook.

   ```
   vim ~/ans/dellemc-ecs-s3-playbook01.yml
   ```

10. Create the following play.

```
---
- name: Create an s3 bucket on AWS with Ansible
  hosts: localhost
  gather_facts: False
  connection: local

  collections:
          - amazon.aws

  vars:
      ECS_host: ## to ECS host e.g. 192.168.33.15:9021   ## use the data API port
      bucketName: ducktoaster77 ## name of bucket to be created

  vars_files:
          - ~/cred.ecs_s3
  tasks:
          - name:
            amazon.aws.aws_s3:
            aws_s3:
              bucket: "{{ bucketName }}"
              mode: create    # change to absent to remove
              permission: public-read
              s3_url: "https://{{ ECS_host }}/"
              validate_certs: no
              encrypt: no
              rgw: true
              aws_access_key: "{{ username }}"
              aws_secret_key: "{{ password }}"
            register: results_CreateBucket
```

11. Save and exit with `:wq`

12. Try running the playbook.

```
ansible-playbook ~/ans/dellemc-ecs-s3-playbook01.yml --ask-vault-pass
```

13. When prompted, type the password you invented a few steps ago to encrypt your `~/cred.ecs_s3` file.

> A common failure for creating S3 buckets, is that the name must be unique. If `ducktoaster77` has already been taken, you might need to create `koolaidchistmas62`

14. Now that your bucket has been created, you should be able to view it on you account, as well as interact with it.

15. Great job! That's it for this lab.

30. Ansible for Pure Flash Array

Lab Objective

*This lab is a **case study**, and is designed to reflect acquired knowledge. The lab environments do not include targets for this lab, or access to the equipment may lay outside the current environment. Reading through the lab can still offer valuable information on using Ansible for Pure Flash Array.*

The objective of this lab to learn how to automate the Pure Flash Array with Ansible.

The following links will help you get started augmenting your Ansible controller to support Pure Flash Storage Arrays.

Pure Storage Array Collection on GitHub:
https://github.com/Pure-Storage-Ansible/FlashArray-Collection

Pure Storage Array Collection on Ansible Galaxy:
https://galaxy.ansible.com/purestorage/flasharray

The purestorage Python Client:
https://pypi.org/project/purestorage/

This blog is a bit dated, but it is still a good resource for getting started: https://blog.purestorage.com/purely-technical/using-ansible-to-manage-your-pure-infrastructure/

Procedure

1. Read through the above comments. When you're ready, you'll need to install the following packages on the Ansible controller.
2. Install the following client to interact with Pure Flash Storage Array.

   ```
   student@bchd:~$ python3 -m pip install purestorage
   ```

3. With Ansible 2.10+, we are encouraged to augment the Ansible controller via collections. Read about that now https://galaxy.ansible.com/purestorage/flasharray
4. Install the collection now. *Note: If you need to control the installation path, use the -p (path) argument.*

   ```
   student@bchd:~$ ansible-galaxy collection install purestorage.flasharray -c
   ```

5. You could also use the following command to learn a bit more about where ansible is installed.

   ```
   student@bchd:~$ ansible --version
   ```

6. Create the `~/ans/` and `~/ans/vars/` folders in the correct location for your system.

   ```
   student@bchd:~$ mkdir -p ~/ans/vars/
   ```

7. Create a file for the credentials.

   ```
   student@bchd:~$ vim ~/ans/vars/credsvault-pure.yml
   ```

8. Place the following information in the new file. In production, you'll need to replace the following information with your own api_token for a privileged user, as well as the IP address of your Pure Flash Array.

   ```
   ---
   # credentials for pure flash array
   api_token: "e31060a7-21fc-e277-6240-25983c6c4592"       # api token of privileged user
   fa_url: "10.10.10.2"                                    # IP of Flash Array
   ```

9. Save and exit with `:wq`
10. Create a playbook to communicate with the Dell EMC Storage Array.

    ```
    student@bchd:~$ vim ~/ans/playbook-pure01.yml
    ```

11. Create the following:

```
---
- name: Connect to Pure Storage Flash Array
  hosts: localhost
  connection: local
  gather_facts: no

  vars_files:
          - vars/credsvault-pure.yml

  collections:
          - purestorage.flasharray

  tasks:
          - name: Try changing banner on Pure Storage Flash Array
            purefa_banner:
              banner: "Banner over\ntwo lines"
              state: present
              fa_url: "{{ fa_url }}"
              api_token: "{{ api_token }}"
```

12. Save and exit with `:wq`
13. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-pure01.yml
```

14. If that worked, then we should encrypt the credential file with `ansible-vault`.

```
student@bchd:~$ ansible-vault encrypt ~/ans/vars/credsvault-pure.yml --ask-vault-pass
```

15. Use the password `alta3`
16. Try running the playbook again, be sure to include the `--ask-vault-pass` flag to let Ansible know you have a password to decrypt the credentials. *Note: In production, it is possible to use a key file in place of a string password when working with ansible-vault.*

```
student@bchd:~$ ansible-playbook ~/ans/playbook-pure01.yml --ask-vault-pass
```

17. Provide the password `alta3`
18. The playbook should execute the same as it did previously.
19. A list of current Pure Flash Storage Array modules are available on the content page (click the Content button) https://galaxy.ansible.com/purestorage/flasharray
20. Answer the following questions:
    - **Q: Is a collection the only way to install modules, like those for Pure Flash Array?**
      - *A: No. You can clone the git repo, and then point to the location within your ansible.cfg file.*
    - **Q: When were collections introduced?**
      - *A: They were introduced with Ansible 2.9, which was available in early-ish 2020. They're a new concept that continues to evolve.*
21. Great! That's it for this lab.

31. Ansible for NetApp

Lab Objective

*This lab is a **case study**, and is designed to reflect acquired knowledge. The lab environments do not include targets for this lab, or access to the equipment may lay outside the current environment. Reading through the lab can still offer valuable information on using Ansible for NetApp.*

The objective of this lab to learn how to automate the NetApp with Ansible.

There are currently 6 NetApp Collections. Each one augments the Ansible Controller with different abilities. All of them are listed on GitHub for NetApp Collections:
https://github.com/ansible-collections/netapp

Here is the collection for the **ontap** product:
https://galaxy.ansible.com/netapp/ontap

Here is the collection for the **StorageGrid** product: https://galaxy.ansible.com/netapp/storagegrid

The netapp Python Client is an installation requirement on the controller:
https://pypi.org/project/netapp-lib/

Procedure

NetApp OnTap

1. Read through the above comments. When you're ready, you'll need to install the following packages on the Ansible controller.
2. Install the following client to interact with the NetApp ontap.

```
student@bchd:~$ python3 -m pip install netapp-lib
```

3. With Ansible 2.10+, we are encouraged to augment the Ansible controller via collections. Read about that now https://galaxy.ansible.com/netapp/ontap
4. Install the collection now. This collection is for the NetApp ontap. A total of 6 collections exist for NetApp. See the GitHub repo for a listing of all collections. *Note: If you need to control the installation path, use the -p (path) argument.*

```
student@bchd:~$ ansible-galaxy collection install netapp.ontap -c
```

5. If necessary, use the following command to learn a bit more about where ansible is installed.

```
student@bchd:~$ ansible --version
```

6. Create the `~/ans/` and `~/ans/vars/` folders in the correct location for your system.

```
student@bchd:~$ mkdir -p ~/ans/vars/
```

7. Create a file for the credentials.

```
student@bchd:~$ vim ~/ans/vars/credsvault-netapp-ontap.yml
```

8. Place the following information in the new file. In production, you'll need to replace the following information with your own hostname, username and password.

```
---
# netapp credentials
hostname: "na-vsim"
username: "admin"
password: "admins_password"
```

9. Save and exit with `:wq`
10. Create a playbook to communicate with the NetApp Storage Array.

```
student@bchd:~$ vim ~/ans/playbook-netapp01.yml
```

11. Create the following:

```
---
- name: Playbook to gather facts on NetApp OnTap
  hosts: localhost
  connection: local
  gather_facts: no

  vars_files:
          - vars/credsvault-netapp-ontap.yml

  collections:
          - netapp.ontap

  tasks:
          - name: Gather facts on NetApp OnTap platform
            na_ontap_info:
              state: info
              hostname: "{{ hostname }}"
              username: "{{ username }}"
              password: "{{ password }}"
            register: results


          - name: display the gathered facts
            debug:
              msg: "{{ results.ontap_info }}"
```

12. Save and exit with `:wq`

13. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-netapp01.yml
```

14. If that worked, then we should encrypt the credential file with `ansible-vault`.

```
student@bchd:~$ ansible-vault encrypt ~/ans/vars/credsvault-netapp-ontap.yml --ask-vault-pass
```

15. Use the password `alta3`

16. Try running the playbook again, be sure to include the `--ask-vault-pass` flag to let Ansible know you have a password to decrypt the credentials. *Note: In production, it is possible to use a key file in place of a string password when working with ansible-vault.*

```
student@bchd:~$ ansible-playbook ~/ans/playbook-netapp01.yml --ask-vault-pass
```

17. Provide the password `alta3`

18. The playbook should execute the same as it did previously.

19. A list of current Pure Flash Storage Array modules are available on the collection page (click the "Content" button) https://galaxy.ansible.com/netapp/ontap

NetApp StorageGrid

1. With Ansible 2.10+, we are encouraged to augment the Ansible controller via collections. Read about that now https://galaxy.ansible.com/netapp/storagegrid

2. Install the collection now. This collection is for the NetApp StorageGrid. A total of 6 collections exist for NetApp. See the GitHub repo for a listing of all collections. *Note: If you need to control the installation path, use the -p (path) argument.*

```
student@bchd:~$ ansible-galaxy collection install netapp.storagegrid -c
```

3. If necessary, use the following command to learn a bit more about where ansible is installed.

```
student@bchd:~$ ansible --version
```

4. Create the `~/ans/` and `~/ans/vars/` folders in the correct location for your system.

```
student@bchd:~$ mkdir -p ~/ans/vars/
```

5. Create a file for the credentials.

```
student@bchd:~$ vim ~/ans/vars/credsvault-netapp-sg.yml
```

6. Place the following information in the new file. In production, you'll need to replace the following information with your own hostname, username and password.

```
---
# netapp credentials
sgurl: "https://sgadmin.example.com"
username: "root"
password: "storagegrid123"
```

7. Save and exit with `:wq`

8. Create a playbook to communicate with the NetApp StorageGrid.

```
student@bchd:~$ vim ~/ans/playbook-netapp02.yml
```

9. Create the following:

```
---
- name: Playbook to gather facts on NetApp StorageGrid
  hosts: localhost
  connection: local
  gather_facts: no

  vars_files:
        - vars/credsvault-netapp-sg.yml

  collections:
        - netapp.storagegrid

  tasks:
        - name: Get Grid Authorization token
          uri:
            url: "{{ sgurl }}/api/v3/authorize"
            method: POST
            body: {
              "username": "{{ username }}",
              "password": "{{ password }}",
              "cookie": false,
              "csrfToken": false
            }
            body_format: json
            validate_certs: false
          register: auth

        # now that we have an auth token, we can comm with the NetApp StorageGrid
        - name: Gather StorageGRID Grid info
          na_sg_grid_info:
            api_url: "{{ sgurl }}"
            auth_token: "{{ auth.json.data }}"
            validate_certs: false
          register: sg_grid_info

        - name: Show the facts gathered from the NetApp StorageGrid
          debug:
                  var: sg_grid_info
```

10. Save and exit with `:wq`

11. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-netapp02.yml
```

12. If that worked, then we should encrypt the credential file with `ansible-vault`.

```
student@bchd:~$ ansible-vault encrypt ~/ans/vars/credsvault-netapp-sg.yml --ask-vault-pass
```

13. Use the password `alta3`

14. Try running the playbook again, be sure to include the `--ask-vault-pass` flag to let Ansible know you have a password to decrypt the credentials. *Note: In production, it is possible to use a key file in place of a string password when working with ansible-vault.*

```
student@bchd:~$ ansible-playbook ~/ans/playbook-netapp02.yml --ask-vault-pass
```

15. Provide the password `alta3`

16. The playbook should execute the same as it did previously.

17. That's it for this lab. Good luck using Ansible to automate your NetApp devices

32. Loops and Mapping YAML Vars Files in Playbooks

Lab Objective

The objective of this lab is to become familiar with the primary component of Ansible, known as a "playbook". Playbooks contain your Ansible code, and made of (at least) two sections. These two sections are `hosts`, and `tasks`. The `hosts` are the "who" to run against, whereas the `tasks` are the "what" you want to do.

It is common practice to design a playbook that describes a host "inventory" as well as variables within external files. This design allows for a more dynamic (reusable) playbook. In this lab, we'll try reading variables from a few different places, and observe how they effect our playbook.

This lab makes use of two hosts. The machine you run your Ansible code (the Ansible controller), and a Linux Ubuntu host, `indy`. Our goal is to use Ansible to make some changes to host `indy`.

It isn't true when running Ansible against APIs or network devices, but when using Ansible to configure Linux hosts, we will want Python installed on our those hosts. That's because (whenever possible) Ansible wants to *push* out work to do. The work is blocks of Python code to be executed. The intention, is that the Python code will bring our target machine into a desired state (if it isn't already).

So, what do you do if your target Linux host is lacking Python? Find out in this lab!

Procedure

1. **You are currently on the Ansible controller.** Confirm the latest version of Ansible is installed.

   ```
   student@bchd:~$ ansible --version
   ```

2. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

3. Download the tear-down script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

4. Run the tear-down script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

5. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/indy-setup.sh
   ```

6. Run the setup script for this lab. **This script will take 3 to 4 minutes to complete.** When it finishes, the script will have deployed an Ubuntu Linux host, `indy` for us to use as a target for Ansible.

   ```
   student@bchd:~$ bash indy-setup.sh
   ```

7. After the script finishes *again, it will take about 3 to 4 minutes. Be patient.* Ping the new indy host @ 10.10.2.2.

   ```
   student@bchd:~$ ping -c 1 10.10.2.2
   ```

8. Try to SSH to the lab host *indy* at *10.10.2.2*.

   ```
   student@bchd:~$ ssh indy@10.10.2.2
   ```

9. Python is a requirement for Ansible to run against remote hosts. Determine if Python 2.7+ is installed. *Note: Python 2.7+ should not appear on any machines at this point in time. It hit end of life in January 2020.*

   ```
   indy@indy:~$ python --version
   ```

10. Determine if Python 3.x is installed (it is).

    ```
    indy@indy:~$ python3 --version
    ```

11. If you're a fan of Indiana Jones, you know Indy is an ophidiophobe (afraid of snakes). So let's do Indy a favor, and uninstall all versions of Python from his system.

```
indy@indy:~$ sudo apt remove python* -y
```

12. Ensure indy has sudo privileges (the response should be `root`).

```
indy@indy:~$ sudo whoami
```

13. Great! Exit `indy` machine.

```
indy@indy:~$ exit
```

14. Now that we're back on the Ansible controller, we want to create our inventory file. Start by creating a directory `~/ans/inv/` to stay organized.

```
student@bchd:~$ mkdir -p ~/ans/inv/
```

15. The inventory, `jones` describes the group or groups of machines we want Ansible to control. If you have never used `vim`, refer back to the `vim` instructional lab for help. *Note: vim, vi, or nano are all fine to use.*

```
student@bchd:~$ vim ~/ans/inv/dev/jones
```

16. Edit your host file as shown below. Careful to include the `[raiders]` heading.

```
[raiders]
indy      ansible_host=10.10.2.2 ansible_user=indy
```

17. Save and exit with `:wq`

18. Before we write a playbook, read about a bit of the code we're going to call on. The `raw` module allows Ansible to control a target machines over SSH (no Python). We'll use this module to install Python on `indy`.

```
student@bchd:~$ ansible-doc raw
```

19. Up and down to scroll, `q` to quit.

20. We will also use the `group` module. This requires python be installed on the `indy` machine first.

```
student@bchd:~$ ansible-doc group
```

21. Up and down to scroll, `q` to quit.

22. We will also use the `user` module. This module also requires python be installed on the `indy` machine first.

```
student@bchd:~$ ansible-doc user
```

23. Up and down to scroll, `q` to quit.

24. Fantastic! Time to create a playbook called, `playbook01.yml`

```
student@bchd:~$ vim ~/ans/playbook01.yml
```

25. Create the following playbook. In this playbook we'll hardcode variables directly into our playbook. We can do this with the keyword `vars`.

```
---
# Learning about Playbooks
- name: bootstrapping and new user creation
  hosts: raiders        # our group in our inventory
  gather_facts: no

  vars:
    newuser1: marian
    newuser2: belloq
    newuser3: willie
    newuser4: elsa

  tasks:
  # use the raw module
  - name: update apt with latest repo info
    raw: "apt upgrade -y"
    become: yes

  - name: install py3 and pip3 with ansible raw
    raw: "apt install -y python3-pip"
    become: yes

  - name: create new group
    group:
      name: indymovies
      state: present
    become: yes

  - name: Add newuser1 to target hosts
    user:
      name: "{{ newuser1 }}"
      state: present
      group: indymovies
    become: yes

  - name: Add newuser2 to target hosts
    user:
      name: "{{ newuser2 }}"
      state: present
      group: indymovies
    become: yes

  - name: add newuser3 to target hosts
    user:
      name: "{{ newuser3 }}"
      state: present
      group: indymovies
    become: yes

  - name: add newuser4 to target hosts
    user:
      name: "{{ newuser4 }}"
      state: present
      group: indymovies
    become: yes
```

26. Exciting times! Run your Ansible playbook. Be sure to include the inventory flag to point to your inventory (jones).

```
student@bchd:~$ ansible-playbook ~/ans/playbook01.yml -i ~/ans/inv/dev/jones
```

27. All that yellow on the screen means Ansible modified the state of the target machine. Test to see if everything worked by SSHing to the host `indy`.

```
student@bchd:~$ ssh indy@10.10.2.2
```

28. Determine if the new user creation playbook was successful. If it was, we should be able to switch to one of the new users, such as `marian`.

```
indy@indy:~$ sudo su marian
```

29. Exit back to the indy account.

```
$ exit
```

30. If you'd like, switch to one (or more) of the new users such as `belloq`, `willie`, or `elsa`.

31. Exit the indy machine to return to the Ansible controller.

```
indy@indy:~$ exit
```

32. Great! Next, let's clean up that last playbook with the keyword `loop`. Loop will allow us to loop across our data, if we just present it in a slightly different manner. Create the playbook `playbook01loop.yml`

```
student@bchd:~$ vim ~/ans/playbook01loop.yml
```

33. Create the following playbook.

```
---
- name: bootstrapping and new user creation
  hosts: raiders
  gather_facts: no

  vars:
    newusers:
    - marian
    - belloq
    - willie
    - elsa

  tasks:
  - name: update apt with latest repo info
    raw: "apt upgrade -y"
    become: yes

  - name: install py3 and pip3 with ansible raw
    raw: "apt install -y python3-pip"
    become: yes

  - name: create new group
    group:
      name: indymovies
      state: present
    become: yes

  - name: Add newusers to target hosts
    user:
      name: "{{ item }}"
      state: present
      group: indymovies
    become: yes
    loop: "{{ newusers }}"
```

34. Save and exit with `:wq`

35. Run your code.

```
student@bchd:~$ ansible-playbook ~/ans/playbook01loop.yml -i ~/ans/inv/dev/jones
```

36. Notice in the play recap, that our 7 tasks were reduced to 4. That's because a single task is *looping*, and in effect, turned 4 tasks into a single task. Let's try the same thing, only move our data into a variable file. This can be done with `vars_files`. To start, we'll create our variable file. Stay organized. Create a folder `~ans/vars/` that we can use to stash our vars files in.

```
student@bchd:~$ mkdir ~/ans/vars/
```

37. Create your yaml file, `~/ans/vars/newusers1.yml`.

```
student@bchd:~$ vim ~/ans/vars/newusers1.yml
```

38. Make your file look like the following. This is YAML syntax for a simple list. One of the benefits of mapping data in external files, is we can offer users of the playbook an editable file, without actually requiring them to edit the playbook itself.

```
---
users1:
  - marian
  - belloq
  - willie
  - elsa
```

39. Save and exit with `:wq`

40. Create a playbook file called `~/ans/playbook01filesloop.yml`

```
student@bchd:~$ vim ~/ans/playbook01filesloop.yml
```

41. Create the following playbook. You may remember making one similar to this one previously. This time, we will map the new users to be created in external files.

```
---
- name: bootstrapping and new user
  hosts: raiders
  gather_facts: no

  # vars_files expects a list of 1 or more files
  vars_files:
    - vars/newusers1.yml

  tasks:
  # update the apt catalog
  - name: "update apt with latest repo info"
    raw: "apt upgrade -y"
    become: yes

  # install Python3 and pip3 with ansible
  - name: "Bootstrap python install"
    raw: "apt install -y python3-pip"
    become: yes

  # create a new group
  - name: Create the new group sandbox
    group:
      name: sandbox
      state: present
    become: yes

  # create new user on target system
  - name: Add new users to group 'sandbox'
    user:
      name: "{{ item }}"
      comment: throwaway user account for testing
      state: present
      shell: /bin/bash       # Defaults to /bin/bash
      system: no             # Defaults to no
      createhome: yes        # Defaults to yes
      group: sandbox
    become: yes
    loop: "{{ users1 }}"
```

42. Save and exit with `:wq`

43. Run your code.

```
student@bchd:~$ ansible-playbook ~/ans/playbook01filesloop.yml -i ~/ans/inv/dev/jones
```

44. Suppose we wanted to loop across multiple pieces of data stored in separate files. To learn this technique, create your second yaml file.

```
student@bchd:~$ vim ~/ans/vars/newusers2.yml
```

45. Make your second file look like the following. Again, this is syntax for a simple list.

```
---
users2:
  - walter
  - jonessr
  - laoche
```

46. Save and exit.

47. Create a playbook that can loop across both data structures. At the top of the playbook we describe **two** files within `vars_files`, and then we also use the plus sign to combine our two lists, `user1` and `users2` within the jinja syntax.

```
student@bchd:~$ vim ~/ans/bootstrap-varsfiles.yml

---
- name: bootstrapping and new user
  hosts: raiders
  gather_facts: no

  # vars_files expects a list of 1 or more files
  vars_files:
    - vars/newusers1.yml
    - vars/newusers2.yml

  tasks:
  # update the apt catalog
  - name: "update apt with latest repo info"
    raw: "apt upgrade -y"
    become: yes

  # install Python3 and pip3 with ansible
  - name: "Bootstrap python install"
    raw: "apt install -y python3-pip"
    become: yes

  # create a new group
  - name: Create the new group sandbox
    group:
      name: sandbox
      state: present
    become: yes

  # create new user on target system
  - name: Add new users to group 'sandbox'
    user:
      name: "{{ item }}"
      comment: throwaway user account for testing
      state: present
      shell: /bin/bash       # Defaults to /bin/bash
      system: no             # Defaults to no
      createhome: yes        # Defaults to yes
      group: sandbox
    become: yes
    loop: "{{ users1 + users2 }}"
    ## notice how we combine two separate lists with the above technique
```

48. Save and exit.

49. Try running your playbook.

```
student@bchd:~$ ansible-playbook ~/ans/bootstrap-varsfiles.yml -i ~/ans/inv/dev/jones
```

50. Make sure everything is still working. SSH to indy.

```
student@bchd:~$ ssh indy@10.10.2.2
```

51. Determine if the new user creation playbook was successful. Switch to one (or more) of the new users.

```
indy@indy:~$ sudo su marian
```

52. Exit back to the indy account.

```
marian@indy:/home/indy/$ exit
```

53. Exit the indy machine.

```
indy@indy:~$ exit
```

54. We can continue to practice working with `vars_files` and `loop`. This time, let's describe packages that need installed on the `indy` system. Create the following YAML file in which to store our variable data.

```
student@bchd:~$ vim ~/ans/vars/aptpkgs.yml
```

55. Create the following file:

```
---
# aptpkgs.yml
bootstrap_packages:
 - python3
 - aptitude

apt_pkgs:
 - iputils-ping
 - cowsay
 - sl
 - cmatrix
```

56. Save and exit with `:wq`

57. Create a new playbook.

```
student@bchd:~$ vim ~/ans/playbook01files02.yml
```

58. Make your playbook look like the following:

```
---
- name: Loop over packages to install
  hosts: raiders
  gather_facts: yes

  vars_files:
  - vars/aptpkgs.yml

  tasks:
  ## upgrade apt with latest repo info
  - name: "update apt"
    raw: "apt update -y"
    become: yes

  ## ensure python3 is installed on indy
  - name: "Bootstrap python3 install"
    raw: "apt install -y {{ item }}"
    loop: "{{ bootstrap_packages }}"
    become: yes

  ## use the apt module to deploy apps
  - name: "Apt install packages"
    apt:
      name: "{{ apt_pkgs }}"
    become: yes
```

59. Save and exit with `:wq`

60. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook01files02.yml -i ~/ans/inv/dev/jones
```

61. It make take awhile for this playbook to finish. Ansible is taking control of the `apt` package manager, and installing Now let's connect to the remote host with SSH.

```
student@bchd:~$ ssh indy@10.10.2.2
```

62. Try out the newly installed applications. Try out the popular linux application Cow Say.

```
indy@indy:~$ cowsay "alta3 is great"
```

63. Try out steam locomotive.

```
indy@indy:~$ sl
```

64. Try out `cmatrix`

```
indy@indy:~$ cmatrix -s
```
(-s: "Screensaver" mode, exits on first keystroke)

65. Exit from the ssh session.

```
indy@indy:~$ exit
```

66. Clean up! Tear down the indy machine.

```
student@bchd:~$ bash max-teardown.sh
```

67. Great job! You have completed this lab.

33. Playbook Tags

Lab Objective

The objective of this lab is to demonstrate how Ansible playbooks may incorporate 'tags', which control which sections of a playbook execute at runtime. If you have a large playbook, it may become useful to be able to run only a specific part of it rather than running everything in the playbook. Ansible supports a “tags:” attribute for this reason.

Control which tags do or do not run from the command line, with the `--tags` or `--skip-tags` options. Alternatively, in Ansible configuration settings (`ansible.cfg`), with the TAGS_RUN and TAGS_SKIP options.

For example, if you wanted to just run the “configuration” and “packages” part of a very long playbook, you can use the --tags option on the command line:

```
ansible-playbook example.yml --tags "configuration,packages"
```

On the other hand, if you want to run a playbook without certain tagged tasks, you can use the --skip-tags command-line option:

```
ansible-playbook example.yml --skip-tags "packages"
```

You can see which tasks will be executed with --tags or --skip-tags by combining it with --list-tasks:

```
ansible-playbook example.yml --tags "configuration,packages" --list-tasks
```

The offical documentation on tags can be found here: https://docs.ansible.com/ansible/latest/user_guide/playbooks_tags.html

Procedure

1. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

2. Download the teardown script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

3. Run the teardown script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

4. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

5. Run the setup script for this lab.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

6. Ping the each of the below lab hosts.

   ```
   student@bchd:~$ ping -c 1 10.10.2.3
   student@bchd:~$ ping -c 1 10.10.2.4
   student@bchd:~$ ping -c 1 10.10.2.5
   student@bchd:~$ ping -c 1 10.10.2.6
   ```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id bender@10.10.2.3
   ```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id fry@10.10.2.4
   ```

9. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
```

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

11. Edit your inventory file.

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following).

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender
fry         ansible_host=10.10.2.4 ansible_user=fry
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit.

14. If you haven't yet, check the documentation for template module: http://docs.ansible.com/ansible/latest/template_module.html

15. Edit `~/.ansible.cfg`

```
student@bchd:~$ vim ~/.ansible.cfg
```

16. Make the file look like the following:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
# save keys to a blackhole
record_host_keys = /dev/null
# turn on logging
log_file = /tmp/ansible.log
# set the var ansible_managed
ansible_managed = Ansible managed
```

17. Create a templates directory.

```
student@bchd:~$ mkdir ~/ans/templates/
```

18. Create a config file for the planetexpress ship under `~/ans/templates/` and call it `cargo_manifest.txt.j2`.

```
student@bchd:~$ vim ~/ans/templates/cargo_manifest.txt.j2
```

```
# cargo_manifest.txt
# {{ ansible_managed }}

[cargo]

holding_bay1 = {{ cargo1 }}

holding_bay2 = {{ cargo2  }}

holding_bay3 = {{ cargo3 }}
```

19. Save and exit.

20. Create a second config file for the planetexpress ship under `~/ans/templates/`. Call this second template `navigation_charts.cfg.j2`.

```
student@bchd:~$ vim ~/ans/templates/navigation_charts.cfg.j2
```

21. Create the following within `~/ans/templates/navigation_charts.cfg.j2`

```
# navigation_charts.cfg
# {{ ansible_managed }}

[darkmatter-nav]

star_system = {{ star }}

planet = {{ planet  }}

latitude = {{ lat }}

longitude = {{ long }}
```

22. Save and exit.

23. Create a third and final file for the planetexpress crew under `~/ans/templates/`. Call this third template `orders.txt.j2`.

```
student@bchd:~$ vim ~/ans/templates/orders.txt.j2
```

24. Create the following simple template.

```
# orders.txt
# {{ ansible_managed }}

[orders]
Primary Mission: {{ pmiss }}

Secondary Mission: {{ smiss }}
```

25. Save and exit.

26. Create a basic playbook file called `playbook-crewauto.yml`

```
student@bchd:~$ vim ~/ans/playbook-crewauto.yml
```

27. Copy and paste the following into your new playbook.

```
---
- name: Playbook with tags
  hosts: planetexpress

  vars:
    cargo1: "prizes for claw crane"
    cargo2: "popcorn"
    cargo3: "pillows"

  # this file will fill in the empty values found in the *.j2 files
  vars_files:
    - /home/student/ans/vars/tempvars.yml

  tasks:
    - name: "Deploy orders to the crew"
      template:
        src: /home/student/ans/templates/orders.txt.j2 #name of the template on ansible controller
        dest: ~/orders.txt #name of the completed file to be placed on the target system
      tags:
      - orders

    - name: "Deploy the cargo manifest"
      template:
        src: /home/student/ans/templates/cargo_manifest.txt.j2 #name of the template on ansible controller
        dest: ~/cargo_manifest.txt #name of the completed file to be placed on the target system
      tags:
      - cargo
      - ship_prep

    - name: "Install navigation charts"
      template:
        src: /home/student/ans/templates/navigation_charts.cfg.j2 #name of the template on ansible controller
        dest: ~/navigation_charts.cfg #name of the completed file to be placed on the target system
      tags:
      - nav
      - ship_prep
```

28. Save and exit.

29. Verify the playbook contents with a cat command.

```
student@bchd:~$ cat ~/ans/playbook-crewauto.yml
```

30. Let's make a folder where we can stuff a variables file, we can call it `~/ans/vars/`

```
student@bchd:~$ mkdir ~/ans/vars/
```

31. Create a variable file, where we'll define our remaining variables. We actually already referenced this inside of the playbook. Call this, `~/ans/vars/tempvars.yml`

```
student@bchd:~$ vim ~/ans/vars/tempvars.yml
```

32. Create the following:

```
---
star: "sol"
planet: "Moon of Earth"
lat: "58"
long: "23"
pmiss: "Good news everyone! We're delivering movie theater supplies."
smiss: "Bring back some Diet Slurm Cola"
...
```

Save and exit.

33.

34. Great job. That was a lot of set up, but now we can start running our playbook. Issue the following command. By including the `cargo` tag, we are indicating that we only wish for tasks tagged `cargo` to run.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-crewauto.yml --tags="cargo"
```

35. The playbook should **only** run the setup module (gathering facts) followed by the `Deploy the cargo manifest` task.

36. SSH into each host and confirm that they all now contain a `cargo_manifest.txt` within their home directory.

37. First, SSH to bender.

```
student@bchd:~$ ssh bender@10.10.2.3
```

38. Confirm bender has `cargo_manifest.txt`

```
bender@bender:~$ cat cargo_manifest.txt
```

39. Exit from `bender`

```
bender@bender:~$ exit
```

40. Repeat the procedure above for the other two hosts. SSH into both `fry@10.10.2.4` and `zoidberg@10.10.2.5` confirm that each have the `cargo_manifest.txt` file.

41. Run the playbook again. This time, use the `ship_prep` tag.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-crewauto.yml --tags="ship_prep"
```

42. The playbook should **only** run the setup module (gathering facts) followed by the `Deploy the cargo manifest` task, and finally the `Install navigation charts`. Both of these tasks are tagged with `ship_prep`

43. SSH into each host and confirm that they all still contain `cargo_manifest.txt`, and now also contain `navigation_charts.cfg` within their home directory.

44. First, SSH to bender.

```
student@bchd:~$ ssh bender@10.10.2.3
```

45. Confirm bender has `navigation_charts.cfg`

```
bender@bender:~$ cat navigation_charts.cfg
```

46. Confirm bender still has `cargo_manifest.txt`

```
bender@bender:~$ cat cargo_manifest.txt
```

47. Exit from `bender`

```
bender@bender:~$ exit
```

48. Repeat the procedure above for the other two hosts. SSH into both `fry@10.10.2.4` and `zoidberg@10.10.2.5` confirm that each have a copy of `cargo_manifest.txt`, as well as `navigation_chargs.cfg`

49. One last time, let's run the playbook. This time we will **only** run the tag `orders`, and also use the `limit` command to ensure only `fry` receives them.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-crewauto.yml --tags="orders" --limit="fry"
```

50. The playbook should **only** run the setup module (gathering facts) followed by the `Deploy orders to the crew` task, and this should be limited to only the host `fry`.

51. SSH to fry.

```
student@bchd:~$ ssh fry@10.10.2.4
```

52. Confirm fry has `orders.txt`

```
fry@fry:~$ cat orders.txt
```

53. Confirm fry still has the other two files, `cargo_manifest.txt` as well as `navigation_charts.cfg`.

```
fry@fry:~$ ls
```

54. Exit from `fry`

```
fry@fry:~$ exit
```

55. Repeat the procedure above for the other two hosts. SSH into both `bender@10.10.2.3` and `zoidberg@10.10.2.5` to confirm that each have a copy of `cargo_manifest.txt`, as well as `navigation_chargs.cfg` but do **NOT** have a copy of `orders.txt`.

56. Clean up! Tear down the remote hosts.

```
student@bchd:~$ bash max-teardown.sh
```

57. That was a long one, but way to go. Hopefully, how to use the key word `tags` is now more clear.

34. Handlers & Listeners

Lab Objective

This lab will demonstrate the Ansible handlers. As we've mentioned, modules should be idempotent and can relay when they have made a change on the remote system. Playbooks recognize this and have a basic event system that can be used to respond to change.

These 'notify' actions are triggered at the end of each block of tasks in a play, and will only be triggered once even if notified by multiple different tasks.

Those things listed under a `notify` section of a task are called handlers.

Handlers are lists of tasks, not really any different from regular tasks, that are referenced by a globally unique name, and are notified by notifiers. If nothing notifies a handler, it will not run. Regardless of how many tasks notify a handler, it will run only once, after all of the tasks complete in a particular play.

Check out some basic documentation on handlers: http://docs.ansible.com/ansible/latest/playbooks_intro.html#handlers-running-operations-on-change

Procedure

1. The setup for this lab is very similar to a previous lab. Therefore, if you completed that lab, you may already have some of the configuration files you are being requested to create.
2. Go skim the documentation for handler. Navigate your browser to the handler module documentation
3. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

4. Download the teardown script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

5. Run the teardown script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

6. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

7. Run the setup script for this lab.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

8. Ping the each of the below lab hosts.

   ```
   student@bchd:~$ ping -c 1 10.10.2.3
   student@bchd:~$ ping -c 1 10.10.2.4
   ```

9. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id bender@10.10.2.3
   ```

10. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

    ```
    student@bchd:~$ ssh-copy-id fry@10.10.2.4
    ```

11. The `ansible.cfg` should describe your default inventory location (typically, you want to set default to development).

    ```
    student@bchd:~$ vim ~/.ansible.cfg
    ```

12. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

13. Save and exit with `:wq`

14. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

15. Edit your inventory file.

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

16. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following). Notice that in this inventory, we've created a group called, `web` that includes `bender` and `fry`

```
[web]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
```

17. Save and exit.

18. Let's also create an `index.html` file that can be moved into our Apache server. Best practice says this should live in a folder called `files`. Make that now.

```
student@bchd:~$ mkdir -p ~/ans/files/
```

19. Create `index.html`

```
student@bchd:~$ vim ~/ans/files/index.html
```

20. Make `index.html` look like the following:

```
<body>
<h1>Apache Webserver is up and running!</h1>
Way to automate with Ansible!
</body>
```

21. Save and exit with `:wq`

22. While we can change the content being returned by the service, we can't change the confirmation of the service without restarting it. The problem is that if we hardcode a restart into our playbook, than we will ALWAYS get an interruption of service when the playbook runs, even if it isn't required. The solution is an Ansible handler. Create the following playbook that includes a **handler**.

```
student@bchd:~$ vim ~/ans/playbook-handler01.yml
```

23. Create the following playbook.

```
---
- name: Apache server installed
  hosts: web
  gather_facts: no
  become: yes

  tasks:

  # the package module tries to select
  # yum or apt or pkg5 (etc) automatically
  - name: latest Apache version installed
    package:
      name: apache2
      state: latest

  - name: Apache enabled and running
    service:
      name: apache2
      enabled: yes
      state: started

  # Copy index.html into the service
  - name: copy index.html
    copy:
      src: ~/ans/files/index.html
      dest: /var/www/html/

  # if dest is directory download every time
  # but only replace if destination is different
  - name: Download a copy of apache2.conf
    get_url:
      url: https://raw.githubusercontent.com/rzfeeser/alta3files/master/apache2.conf
      dest: /etc/apache2/
    notify:
        - restart_apache

  # This will ONLY run if the associated tasks run
  # no matter how many times called these tasks
  # will ONLY run once
  handlers:
  - name: restart_apache
    service:
      name: apache2
      state: restarted
```

24. Save and exit.

25. Great. Try running the playbook again.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-handler01.yml -i ~/ans/inv/dev/hosts
```

26. Notice that the file was downloaded to the remote system (the `get_url` task is marked `CHANGED`). Because that task ran, the handler was also called. The handler, in this case, is restarting our service, ensuring our new `apache2.conf` file has taken effect.

27. To observe how a handler works, you'll need to run the playbook a second time.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-handler01.yml -i ~/ans/inv/dev/hosts
```

28. Notice that the handler did **not** run this time. It did not run, because it wasn't called by the `get_url` task. Since our service wasn't needlessly restarted, our web service did not experience a service impact.

29. Run the playbook one more time. Nothing changed, so we won't expect the `get_url` or handler to run.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-handler01.yml -i ~/ans/inv/dev/hosts
```

30. Now let's study the Ansible listener. A better name would of been, "a way to group together multiple handlers that can all be triggered when the listener is notified". In our story, maybe we also want an instance of mysql available that our webserver can interact with. We can accomplish that by installing 'mariaDB'. If the Apache webservice, or the mariaDB client or server need installed or updated, it might be a good idea to restart all off the services, which could be a great application to notify a `listener`. Create a new playbook.

```
student@bchd:~$ vim ~/ans/playbook-handler02-listen.yml
```

31. Write out the following playbook. It is similar to our previous playbook, with some commented additions. The first is to give `package` a list of services to install (`apache2`, `mariadb` server and client). We also added a notify to `restart_webservices`. Within the `handler` section, we use the `listen` keyword to group together handlers by the value `restart_webservices`. Therefore, if the `package` task runs, both the Apache and mysql services will be bounced.

```
---
- name: Apache server installed
  hosts: web
  gather_facts: no
  become: yes

  tasks:

  # the package module tries to select
  # yum or apt or pkg5 (etc) automatically
  # if any of these services need installed or
  # updated, then they ALL get restarted
  - name: latest Apache version installed
    package:
      name:
        - apache2        ## <-- updated
        - mariadb-server ## <-- updated
        - mariadb-client ## <-- updated
      state: latest
    notify:
      - restart_webservices ## <-- updated

  - name: Apache enabled and running
    service:
      name: apache2
      enabled: yes
      state: started

  # Copy index.html into the service
  - name: copy index.html
    copy:
      src: ~/ans/files/index.html
      dest: /var/www/html/

  # if dest is directory download every time
  # but only replace if destination is different
  # https://raw.githubusercontent.com/rzfeeser/
  #              alta3files/master/apache2.conf
  - name: Download a copy of apache2.conf
    get_url:
      url: https://raw.githubusercontent.com/rzfeeser/alta3files/master/apache2.conf
      dest: /etc/apache2/
    notify:
        - restart_apache   # ONLY restart apache if this conf
                           # file needs updated

  # ensure the mysql service is up and running
  - name: mysql (mariadb) is running
    service:
      name: mysql
      enabled: yes
      state: started

  # if this line needs added to my.cnf
  # then ONLY the mysql service needs restarted
  - name: Modify SQL conf file to listen on all interfaces
    lineinfile:
      dest: /etc/mysql/my.cnf
      regexp: "^bind-address"
      line: "bind-address=0.0.0.0"
```

```
    notify:
      - restart_mysql

  # This will ONLY run if the associated tasks run
  # no matter how many times called these tasks
  # will ONLY run once
  handlers:
  - name: restart_apache
    service:
      name: apache2
      state: restarted
    listen: restart_webservices

  ## this is new, restarts mysql
  - name: restart_mysql
    service:
      name: mysql
      state: restarted
    listen: restart_webservices
```

32. Save and exit.

33. Let's trash your two hosts `fry` and `bender`.

```
student@bchd:~$ bash max-teardown.sh
```

34. Rebuild fresh instances of `fry` and `bender`.

```
student@bchd:~$ bash pexpress-setup.sh
```

35. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id bender@10.10.2.3
```

36. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id fry@10.10.2.4
```

37. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
```

38. Okay, run your playbook against your freshly rebuilt hosts.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-handler02-listen.yml -i ~/ans/inv/dev/hosts
```

39. The result should be `ok=8` and `changed=8`. Now, try running that playbook again.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-handler02-listen.yml -i ~/ans/inv/dev/hosts
```

40. This time, the result should be `ok=6` and `changed=0`. Hopefully, the reason is clear; the target machines had already achieved the **state** we described with our Ansible code, so there was no reason to 'notify' any of our handlers.

41. Try triggering a single handler. Edit the playbook so it reads, `bind-address=1.1.1.1`, and re-run. After this run, the result should be `ok=7 changed=2`. That's because the change caused the `lineinfile` module to execute. When it executes, the `restart_mysql` handler is called, thus restarting the single mysql service, not both services.

42. Run the teardown script to remove your remote hosts `fry` and `bender`.

```
student@bchd:~$ bash max-teardown.sh
```

43. Answer the following questions:
    - **Q: What is the purpose of a handler?**
        - *A: Handlers are tasks that are only run if they are notified to run. Notifications are produced if a task within the task section reports a "change". Handlers only run once, even if they are notified to run multiple times. Typically, they are used to perform restarts on an application or system.*
    - **Q: What is a listener?**
        - *A: A listener is a way to "group" together handlers.*
44. Great job! You have completed this lab.
45. If you are tracking your code on GitHub, take a moment and perform a commit.
    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "learning about handlers and listeners"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

35. Ansible Error Handling

Lab Objective

This lab will demonstrate the Ansible error handling. In 2.0, Ansible added a block feature to allow for logical grouping of tasks and even in-play error handling. Most of what you can apply to a single task can be applied at the block level, which also makes it much easier to set data or directives common to the tasks. The `block` may be further augmented by `rescue` and `always`.

block - This is what you want to do. Blocks allow for logical grouping of tasks and in play error handling. Most of what you can apply to a single task (with the exception of loops) can be applied at the block level.

rescue - Why do we fall down? So we can learn to stand back up! Think of rescue as how you "stand back up" or "how to recover". If you do not encounter any failures within the rescue, your playbook will continue.

always - These tasks are always performed, even if your block **and** rescue sections encounter failures, you'll still perform these tasks.

By combining blocked tasks with `rescue` and `always` tasks, it is possible to create highly responsive playbook thats can **rollback** when they encounter an error. We'll explore some of those techniques in this lab.

Read about blocks within Ansible:
https://docs.ansible.com/ansible/latest/user_guide/playbooks_blocks.html

Read about error handling within Ansible:
https://docs.ansible.com/ansible/latest/user_guide/playbooks_error_handling.html

Read about Ansible keywords that may be applied at a block level:
https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html#block

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the teardown script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id bender@10.10.2.3
```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id fry@10.10.2.4
```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit.

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

16. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

17. Save and exit with `:wq`

18. If you haven't already, skim the documentation for apt: https://docs.ansible.com/ansible/latest/user_guide/playbooks_error_handling.html

19. Create a basic playbook file called `~/ans/playbook-error-handler01.yml`

```
student@bchd:~$ vim ~/ans/playbook-error-handler01.yml
```

20. Create the following solution:

```
---
 - name: Learning about blocks
   hosts: planetexpress

   tasks:
    - block:
       - shell: "echo 'My single task ran' > example.txt"
       - shell: "echo 'My second task ran' >> example.txt"
       - shell: "echo 'My third task ran' >> example.txt"
```

21. Save and exit.

22. Next run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-error-handler01.yml
```

23. Now let's connect to the one of the remote hosts with secure shell to see the change we made.

```
student@bchd:~$ ssh bender@10.10.2.3
```

24. Look at the file that was produced.

```
bender@bender:~$ cat example.txt

My single task ran
My second task ran
My third task ran.
```

25. Cool. So, all the tasks within the block executed. Delete the file.

```
bender@bender:~$ rm example.txt
```

26. Exit from the ssh session.

```
bender@bender:~$ exit
```

27. Creae a playbook called, `~/ans/playbook-error-handler02.yml`

```
student@bchd:~$ vim ~/ans/playbook-error-handler02.yml
```

28. Create the following playbook:

```
---
 - name: Learning about block rescue and always
   hosts: planetexpress

   tasks:
    - block:
        - shell: "echo 'My single task ran' > example.txt"
        - shell: "echo 'My second task ran' >> example.txt"
        - shell: "echo 'My third task ran' >> example.txt"
      rescue:
        - shell: "echo 'My rescue code ran. Maybe it undoes whatever the block tried to do' >> example.txt"
      always:
        - shell: "echo 'This always runs. Maybe it is a commit, or an entry in a log' >> example.txt"
```

29. Save and exit.

30. Run the playbook. The `rescue` section of the block should **not** run (or be skipped).

```
student@bchd:~$ ansible-playbook ~/ans/playbook-error-handler02.yml
```

31. Now let's connect to the one of the remote hosts with secure shell to see the change we made.

```
student@bchd:~$ ssh bender@10.10.2.3
```

32. Look at the file that was produced.

```
bender@bender:~$ cat example.txt

My single task ran
My second task ran
My third task ran.
This always runs. Maybe it is a commit, or an entry in a log
```

33. Appears as if all the tasks within the `block` executed, no tasks within the `rescue` section executed, however, the tasks within the `always` section did execute. Delete the file. The `always` tasks will **always** execute, even on a failure! A good tasks to put in the `always` section might be instructions for Ansible to send you an email with the results of the play. *To send email with Ansible, research the mail module*.

```
bender@bender:~$ rm example.txt
```

34. Exit from the ssh session.

```
bender@bender:~$ exit
```

35. Create a new playbook, `~/ans/playbook-error-handler03.yml`. In this playbook, we will insert an error.

```
student@bchd:~$ vim ~/ans/playbook-error-handler03.yml
```

36. Create the following playbook with an error in the block. Note, you can always throw an error by making a reference to `/bin/false`.

```
---
 - name: A playbook that throws an error in the block.
   hosts: planetexpress

   tasks:
    - block:
       - shell: "echo 'My single task ran' > example.txt"
       - shell: "echo 'My second task ran' >> example.txt"
       - shell: "/bin/false" # This always returns a false value
      rescue:
       - shell: "echo 'My rescue code ran. Maybe it undoes whatever the block tried to do' >> example.txt"
      always:
       - shell: "echo 'This always runs. Maybe it is a commit, or an entry in a log' >> example.txt"
```

37. Save and exit.

38. Next run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-error-handler03.yml
```

39. Now let's connect to the one of the remote hosts with secure shell to see the change we made.

```
student@bchd:~$ ssh bender@10.10.2.3
```

40. Look at the file that was produced.

```
bender@bender:~$ cat example.txt
```

```
My single task ran
My second task ran
My rescue code ran. Maybe it undoes whatever the block tried to do
This always runs. Maybe it is a commit, or an entry in a log
```

41. This time all of our tasks ran! Even the tasks in the `rescue` and `always` blocks. That's because the `rescue` is *"how you stand back up"*, and the `always` section is *always* performed, even on a failure and rescue.

42. Remove the example file.

```
bender@bender:~$ rm example.txt
```

43. Exit from the ssh session.

```
bender@bender:~$ exit
```

44. By pulling together **block**, **rescue**, and **always**, we were able to create very responsive code blocks, that are still highly readable. Ansible is domain-specific code, that is easy to use, and highly predictable. However, it does offer enough complexity, to allow you to adapt Ansible playbooks to mimic your own internal procedures and methodologies for administering changes. Let's create a playbook that works in a **responsive** manner, and can **rollback** if it encounters an error.

```
student@bchd:~$ vim ~/ans/playbook-error-handler04.yml
```

45. Create the following solution:

```
---
## Our goal is to create a playbook that perform pre-checks, changes, and post-checks.
## Upon a failure / error, the playbook will "ROLLBACK" any changes it has made thus far.
## This playbook can serve as a template for constructing 'intelligent' playbooks
## within your enterprise

- name: A playbook demonstrating Error Handling techniques
  hosts: planetexpress
  gather_facts: yes

  vars:
          # list of services to install (seperated by commas and available in the apt repo)
          # apached = apache http server
          # vsftpd = sftp server
          apps_to_install: [apache2, vsftpd]

  tasks:
          ## our job is to install some software on ONLY Debian hosts within the network
          ## therefore our PRECHECK PHASE needs to involve a 'check' to ensure we ONLY are on
          ## Debian hosts
          - name: PRECHECK PHASE
            block:
                 - name: Ensure the platforms we logged into are Debian
                   fail:
                           msg: "Ansible has detected this host is not part of the Debian family."
                   when: ansible_os_family != "Debian"

            rescue:
                 - name: PRECHECK PHASE - FAILED
                   fail:
                           msg: "PRECHECK PHASE - FAILED. Nothing to rollback. Exiting..."


          ## this is the block containing the work we actually want to do
          - name: MAINTENANCE PHASE
            block:
                 - name: Install services(s) to our remote host(s)
                   apt:
                           state: present
                           name: "{{ apps_to_install }}"
                   become: yes

                 - name: Turn up the service(s) installed on our remote host(s)
                   service:
                           name: "{{ item }}"
                           state: started
                   loop: "{{ apps_to_install }}"
                   become: yes

            ## this rescue only runs if the BLOCK fails
            rescue:
                 - name: Remove services(s) on our remote host(s)
                   apt:
                           state: absent
                           name: "{{ apps_to_install }}"
                   become: yes

                 - name: MAINTENANCE PHASE - FAILED
                   fail:
                           msg: "MAINTENANCE PHASE - FAILED. Rollback complete. Exiting..."
```

```
- name: POSTCHECK PHASE
  block:

       - name: populate service facts
         service_facts:

       - name: Print out the service facts
         debug:
                 var: ansible_facts.services

       - name: Ensure all of the new services installed
         fail:
                 msg: "Ansible has detected not all services have installed."
         when: ansible_facts.services.get(item).state != "running"
         loop: "{{ apps_to_install }}"

  ## this rescue only runs if the BLOCK fails
  rescue:
       - name: Remove services(s) on our remote host(s)
         apt:
                 state: absent
                 name: "{{ apps_to_install }}"
         become: yes

       - name: MAINTENANCE PHASE - FAILED
         fail:
                 msg: "POSTCHECK PHASE - FAILED. Rollback complete. Exiting..."

## in a real deployment you might consider an ALWAYS section to provide the results
## of how the playbook ran
# always:
#      - name: EXAMPLE - Email your team the results of the playbook
#        mail:   # mail module is used to send SMTP (email)
#           host:
#           port:
#           password:
#           username:
#           subject:
#           body: send an email with results via the mail module
```

46. Save and exit.

47. Run the playbook. It should run successfully.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-error-handler04.yml
```

48. Try injecting a fail into the playbook with the following snippet at the end of the `PRECHECK PHASE` block (just before the rescue). Your goal is to make a failure within the first of our three sections, and observe how the playbook reacts. It **should** exit without moving onto the `MAINTENANCE PHASE`.

```
- name: Inject a fail
  shell: "/bin/false"
```

49. Remove the code from your `PRECHECK PHASE` block, and move it to the end of the `MAINTENANCE PHASE` block (just before the rescue). Once you've made this change, run your playbook a second time. It **should** uninstall the installed applications, then exit without moving onto the `POSTCHECK PHASE`.

50. Remove the code from your `MAINTENANCE PHASE` block, and move it to the end of the `POSTCHECK PHASE` block (just before the rescue). Once you've made this change, run your playbook a second time. It **should** uninstall the installed applications, then exit.

51. Great job! That's it for this lab.

36. pre_tasks, roles, tasks, post_tasks, and handlers

Lab Objective

The objective of this lab is to understand some advanced components within playbooks. Namely, the pre-tasks and post-tasks sections. When paired with roles, and tasks, it can be overwhelming to understand how to properly architect playbooks.

In this lab you'll create a playbook 'skeleton' that might be used to better understand playbook execution.

Some recommended reading to better understand how playbooks execute:
https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. Ensure `tree` is installed. We'll use it a bit later in the lab.

```
student@bchd:~$ sudo apt install tree -y
```

3. Let's mimic a new project, and learn about Ansible architecture at the same time. One of the takeaways from this lab will be it pays to be organized. Make a new directory to mimic our new project.

```
student@bchd:~$ mkdir -p ~/ans/classicproject/
```

4. The roles directory might exist in a few different locations, but let's create one local to our new project folder.

```
student@bchd:~$ mkdir -p ~/ans/classicproject/roles/
```

5. Move into the new directory.

```
student@bchd:~$ cd ~/ans/classicproject/roles/
```

6. Use the `ansible-galaxy` command line tool (this installs with Ansible) to create the directory structure and files associated with a role. Roles are named after the parent directory that stores this highly organized directory structure.

```
student@bchd:~/ans/classicproject/roles$ ansible-galaxy init bootstrap
```

7. Run a `tree` command to see the directory layout of a role.

```
student@bchd:~/ans/classicproject/roles$ tree
```

8. In a 'real' role, we would now begin to fill in the `main.yml` files with tasks, and possibly move templates and files into the appropriate directories. Here is quick rundown of each folder:

    - `tasks` - contains the main list of tasks to be executed by the role.
    - `handlers` - contains handlers, which may be used by this role or even anywhere outside this role.
    - `defaults` - default variables for the role (see Using Variables for more information).
    - `vars` - other variables for the role (see Using Variables for more information).
    - `files` - contains files which can be deployed via this role.
    - `templates` - contains templates which can be deployed via this role.
    - `meta` - defines some meta data for this role. See below for more details.

9. For now, just use `ansible-galaxy` to create the remaining role structures.

```
student@bchd:~/ans/classicproject/roles$ ansible-galaxy init securityapps
```

10. In a 'real' role, we would fill in the `main.yml` files with tasks. For now, just use `ansible-galaxy` to create the remaining role structures.

```
student@bchd:~/ans/classicproject/roles$ ansible-galaxy init officetools
```

11. Move back into your home directory.

```
student@bchd:~/ans/classicproject/roles$ cd ~
```

12. Create a playbook that uses roles. **You won't be running this playbook. It is only for reference.**

```
student@bchd:~$ vim ~/ans/classicproject/classicpb.yml
```

13. Create the following playbook:

```
---
- name: New Hire Desktop
  hosts: all
  gather_facts: yes

  roles:
    - bootstrap
    - securityapps
    - officetools
```

14. Save and exit. **You won't be running this playbook. It is only for reference.**

15. This designates the following behaviors, for each role 'x':

    - If `roles/x/tasks/main.yml` exists, tasks listed therein will be added to the play.
    - If `roles/x/handlers/main.yml` exists, handlers listed therein will be added to the play.
    - If `roles/x/vars/main.yml` exists, variables listed therein will be added to the play.
    - If `roles/x/defaults/main.yml` exists, variables listed therein will be added to the play.
    - If `roles/x/meta/main.yml` exists, any role dependencies listed therein will be added to the list of roles (1.3 and later).
    - Any copy, script, template or include tasks (in the role) can reference files in roles/x/{files,templates,tasks}/ (dir depends on task) without having to path them relatively or absolutely.

16. Create a larger playbook that could serve as a framework to understand advanced playbook techniques. **You won't be running this playbook. It is only for reference.**

```
student@bchd:~$ vim ~/ans/classicproject/frameworkpb.yml
```

```
---
- name: Advanced Playbook Architecture
  hosts: ghosthouse   ## name of group in inventory
  gather_facts: yes

  pre_tasks: #<-- these run first, followed by handler calls


  # by default, ansible looks for the rolepath as...
  # roles/ in the relative local directory
  # /etc/ansible/roles
  # controlled in ansible.cfg
  roles:
          - bootstrap   #<-- this role runs next
          - bootstrap   #<-- by default this will NOT run a second time
                        # to 'fix' make an entry in 'roles/bootstrap/meta/main.yml'
                        # allow_duplicates: true
          - webserver   #<-- this role runs after the first

  tasks:   #<-- this runs after roles, followed by handlers
   - name: Print to the screen
     debug:
           msg: "Example of a task within the tasks section"


   # default to import_role if you are unsure what to use
   # if nothing else, syntax errors in your code will be caught
   # when the playbook is run, instead of when the role is called

   - import_role:         #<-- STATIC(preprocessing)   diff between
           name: nginx    # import and include
                          # is v poorly described by the concepts
                          # governing "dynamic" vs "static" imports

   - include_role:        #<-- DYNAMIC
           name: webserver-db-{{ webserverhostname | lower }}


  post_tasks:    #<--- this runs after roles, followed by handlers


  handlers:    #<--- custom handlers called by pre_tasks, tasks, or post_tasks



# consider new plays as 'isolated' from the previous play
- name: A second play
  hosts: all
  gather_facts: yes

  tasks:
   - name: Print to the screen
     debug:
             msg: "This is a separate play, but think of it almost like running an entirely new playbook!"
```

17. Save and exit. **You won't be running this playbook. It is only for reference.**

18. When used in this manner, the order of execution for your playbook is as follows:

    - Any `pre_tasks` defined in the play.

- Any `handlers` triggered so far will be run.
- Each role listed in roles will execute in turn. Any role dependencies defined in the roles meta/main.yml will be run first, subject to tag filtering and conditionals. Handlers will **not** be called yet. Handlers called within roles will run after `tasks`.
- Any `tasks` defined in the play.
- Any `handlers` triggered so far will be run.
- Any `post_tasks` defined in the play.
- Any `handlers` triggered so far will be run.

19. So much is possible with Ansible, therefore, don't feel like you need to use EVERY aspect of Ansible to be successful! If you need a default setting, keep it simple! Readability goes hand in hand with re-usability!

20. Great job! That's it for this lab. Hopefully playbook construction is a bit more clear.

37. Ansible Keywords: register and when

Lab Objective

The objective of this lab is to explore keywords, `register`, `when`, as well as explore 'extra variables' (those variables passed at the CLI during run time).

The keyword `register` allows for the assignment of a new variable to the JSON returned to a task after it completes execution. This new variable, and the associated information, could then be recalled for use later in the playbook. Perhaps, as an example, within a conditional.

Conditionals are created with the `when` keyword. As the name implies, this keyword allows users to control 'when' a playbook executes. It should be noted, that when writing a 'when' statement, there is no need to surround variable values with double-mustached brackets.

Review Ansible keywords here: https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html

Ansible variables have precedence, where variables passed from the CLI always have the highest precedence (greatest priority). The flag to pass extra variables from the CLI is the, `-e` flag.

Review variable precedence here: https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

Procedure

1. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

3. Run the teardown script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

4. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

5. Run the setup script for this lab.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id bender@10.10.2.3
   ```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id fry@10.10.2.4
   ```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

   ```
   student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
   ```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

    ```
    student@bchd:~$ mkdir -p ~/ans/inv/dev/
    ```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

    ```
    student@bchd:~$ vim ~/ans/inv/dev/hosts
    ```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit.

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

16. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

17. Save and exit with `:wq`

18. Create a playbook that we can use to explore the `register` keyword.

```
student@bchd:~$ vim ~/ans/playbook-register01.yml
```

19. Create the following playbook.

```
---
 - name: Playbook with the register keyword
   hosts: planetexpress

   tasks:

     - name: "Issue a trivial command"
       command: ls /lib/
       register: contentz_lib

     - name: "debug - display data returned"
       debug:
         msg: "{{ contentz_lib }}"

     - name: "debug - standard error"
       debug:
         msg: "{{ contentz_lib.stderr }}"

     - name: "debug - standard out"
       debug:
         msg: "{{ contentz_lib.stdout }}"
```

20. Save and exit.

21. Run the playbook and examine the JSON displayed on the screen by the debug tasks.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-register01.yml
```

22. Our playbook was able to 'save' what was returned from the command, `ls /lib/` via `register`. We then recalled that stored variable data with `debug` and displayed it to the screen. First we looked at everything, then included dot notation to retrieve, specifically, `stderr` and then `stdout`.

23. Create a basic playbook file called, `~/ans/playbook-register02.yml`

```
student@bchd:~$ vim ~/ans/playbook-register02.yml
```

24. Create the following playbook. The website random.org has an HTTP api that allows us to harvest any kind of random number patterns we like. The link in our playbook will return a 0 or a 1. The `register` keyword will save what is returned to the variable `randvarz`. If you'd like, check out the website and API documentation here: https://www.random.org/clients/http/

```
---
- name: More Register Examples
  hosts: planetexpress
  gather_facts: yes

  tasks:
    - name: Reach out with geturl to random generation
      get_url:
        url: https://www.random.org/integers/?num=1&min=0&max=1&col=1&base=10&format=plain&rnd=new

        dest: ~/rand.txt
        force: yes

    - name: Read the rand.txt and save results in randvarz
      command: cat ~/rand.txt
      register: randvarz

    - name: "debug - display the returned value"
      debug:
        msg: "The value returned to ansible was: {{ randvarz.stdout }}"
```

25. Save and exit.

26. Cool. Time to run the playbook!

```
student@bchd:~$ ansible-playbook ~/ans/playbook-register02.yml
```

27. Run it a second time.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-register02.yml
```

28. Finally, run it a third time.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-register02.yml
```

29. Let's make a new playbook, and introduce the `when` keyword. This will allow our playbook to accept some test-logic (conditionals).

```
student@bchd:~$ vim ~/ans/playbook-register03.yml
```

30. Create the following playbook. Note the keyword, `when`. This allows us to introduce test logic, or, conditionals. Rather than just printing out a message, we can selectively print out a message, depending on the value of `randvarz`. Let's treat a `1` as a "win", and a `0` as a "loss". To be clear, this playbook doesn't really have any practical value. We're just testing how these keywords behave.

```
---
- name: Saving and recalling playbook values
  hosts: planetexpress
  gather_facts: true

  tasks:
    - name: Reach out with geturl to random generation
      get_url:
        url: https://www.random.org/integers/?num=1&min=0&max=1&col=1&base=10&format=plain&rnd=new
        dest: ~/rand.txt
        force: yes

    - name: Read the rand.txt and save results in randvarz
      command: cat ~/rand.txt
      register: randvarz

    - name: "debug - display a win"
      debug:
        msg: "congratulations! {{ ansible_hostname }} won :)"
      when: randvarz.stdout == "1"

    - name: "debug - display a failure"
      debug:
        msg: "Sorry. {{ ansible_hostname }} lost :("
      when: randvarz.stdout == "0"
```

31. Save and exit.
32. Cool. Time to run the playbook!

```
student@bchd:~$ ansible-playbook ~/ans/playbook-register03.yml
```

33. Run it a second time.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-register03.yml
```

34. Finally run it a third time.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-register03.yml
```

35. Tasks within the playbook are now *only* running `when` a particular condition is found to be **True**.
36. Great job! The `register` and `when` keywords should now be more clear.

38. Reading Variables into Playbooks

Lab Objective

The objective of this lab is to demonstrate some mechanisms for reading a series of variables into a playbook. Variable information can be provided from sources like, files, encrypted stores, URI, parsed from returned JSON, and many more. Once obtained, they often need manipulated. Manipulation techniques can include string and list indexing, calling on dictionary keys, referencing python methods, and using jinja templates. This lab teaches some of these techniques.

Procedure

1. Open a new terminal, then change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

2. We want to stay organized, so create a directory structure, `~/ans`.

   ```
   student@bchd:~$ mkdir ~/ans
   ```

3. Move into the new directory.

   ```
   student@bchd:~$ cd ~/ans
   ```

4. In this lab, we'll run against localhost, so go ahead and create a playbook.

   ```
   student@bchd:~/ans$ vim playbook-varspractice01.yml
   ```

5. Create the following.

```
- name: learning to recall var data
  hosts: localhost

  # these are play vars
  vars:
    telecom:
      - ericsson
      - alu
      - avaya
      - cisco
    heroes:
      marvel:
        - spiderman
        - ironman
        - daredevil
      dc:
        - wonderwoman
        - batman
        - superman

  tasks:
    # 1st item in the marvel list
    - name: print out spiderman
      debug:
        msg: "{{ heroes.marvel[0] }}"

    # second item in the dc list
    - name: print out batman
      debug:
        msg: "{{ heroes.dc[1] }}"

    - name: print out ALL dc heroes
      debug:
        msg: "{{ heroes.dc }}"


    - name: print out all dc heroes with a loop
      debug:
        msg: "{{ item }}"
      loop: "{{ heroes.dc }}"

    # fourth item in the telecom list
    - name: print out cisco
      debug:
        msg: "{{ telecom[3] }}"

    - name: Print out all dc and marvel heroes
      debug:
        msg: "{{ heroes }}"

    - name: print out ALL dc AND marvel heroes ONLY
      debug:
        msg: "{{ item }} is the greatest! Go {{ item }}!!!!"
      loop: "{{ heroes.marvel + heroes.dc }}"
```

6. Save and exit.

7. Run your playbook.

```
student@bchd:~/ans$ ansible-playbook playbook-varspractice01.yml
```

8. Make a directory for a separate vars file.

```
student@bchd:~/ans$ mkdir ~/ans/vars
```

9. Create a new vars file.

```
student@bchd:~/ans$ vim ~/ans/vars/titanvars.yml
```

10. Create the following vars file.

```
---
prefix_list:
  ipv4:
     - name: "ICM-TITAN"
       seq:
         -  num: "10"
            action: "act-10"
            rule: "rule-10"
         -  num: "20"
            action: "act-20"
            rule: "rule-20"
     - name: "ICM-TITAN2"
       seq:
         -  num: "30"
            action: "act-30"
            rule: "rule-30"
```

11. Let's convert the data above to JSON so we can see the data association more clearly. We can do that simply with python.

```
student@bchd:~/ans$ python3 -m pip install yml2json
```

12. Use the new python library to convert YAML to JSON. This might help you understand the relationships of the data you're looking at.

```
student@bchd:~/ans$ yml2json ~/ans/vars/titanvars.yml
```

13. Below is the JSON produced by the above Python code.

```
{
    "prefix_list": {
        "ipv4": [
            {
                "name": "ICM-TITAN",
                "seq": [
                    {
                        "action": "act-10",
                        "num": "10",
                        "rule": "rule-10"
                    },
                    {
                        "action": "act-20",
                        "num": "20",
                        "rule": "rule-20"
                    }
                ]
            },
            {
                "name": "ICM-TITAN2",
                "seq": [
                    {
                        "action": "act-30",
                        "num": "30",
                        "rule": "rule-30"
                    }
                ]
            }
        ]
    }
}
```

14. Go ahead and create a playbook.

```
student@bchd:~/ans$ vim playbook-varspractice02.yml
```

15. Create the following. To move through our data, we'll use a loop. The problem is that our data is a dictionary, and not a list, **we will expect this first playbook to fail**

```
---
- name: Using a complex vars file
  hosts: localhost
  gather_facts: False

  vars_files:
    - vars/titanvars.yml

  tasks:
  - name: debug with_dict
    debug:
      var: "{{ item }}"
    loop: "{{ prefix_list }}"  # this will FAIL because prefixlist is a dictionary
```

16. Save and exit.

17. Let's run this playbook and **watch this playbook fail**

```
student@bchd:~/ans$ ansible-playbook playbook-varspractice02.yml
```

18. Let's fix our broken playbook.

```
student@bchd:~/ans$ vim playbook-varspractice02.yml
```

19. The keyword 'loop' wants a list. To correct this, we can use a jinja filter, `dict2items`

```
---
- name: Using a complex vars file
  hosts: localhost
  gather_facts: False

  vars_files:
    - vars/titanvars.yml

  tasks:
  - name: debug with_dict
    debug:
      var: "{{ item }}"
    loop: "{{ prefix_list|dict2items }}"   ## edit this line to include dict2items
    ## the above is a jinja filter
```

20. Save and exit.

21. Jinja filters are... strange. The documentation on them isn't the best, but you should review it anyways: https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html

22. Let's run this playbook and watch the playbook run successfully, as the jinja filter is able to convert our dictionary into an iterable list.

```
student@bchd:~/ans$ ansible-playbook playbook-varspractice02.yml
```

23. Create a new playbook.

```
student@bchd:~/ans$ vim playbook-varspractice03.yml
```

24. The following playbook shows two techniques for looping across data. It really depends what you're trying "to do" to say which way is "correct".

```
---
- name: Practice with variable manipulation
  hosts: localhost
  gather_facts: False

  vars_files:
    - vars/titanvars.yml

  tasks:

    - name: debug with specific values
      debug:
        msg:
          - "prefix-list key    >>> {{ item.key }}"
          - "p value 0 name     >>> {{ item.value[0].name }}"
          - "p value 1 name     >>> {{ item.value[1].name }}"
          - "val 0 seq 0 num    >>> {{ item.value[0].seq[0].num }}"
          - "val 0 seq 1 rule   >>> {{ item.value[0].seq[1].rule }}"
          - "val 1 seq 0 action >>> {{ item.value[1].seq[0].action }}"
      loop: "{{ prefix_list|dict2items }}"

    - name: Loop across the list mapped to prefix_list.ipv4
      debug:
        var: item.name
      loop: "{{ prefix_list.ipv4 }}"
```

25. Save and exit.

26. Run the new playbook.

```
student@bchd:~/ans$ ansible-playbook playbook-varspractice03.yml
```

27. It is also possible to use Python methods within playbooks. Let's try manipulating a string with some python methods. First, obtain the python methods available to us.

```
student@bchd:~/ans$ python3
```

28. Use the `dir()` function on a string to obtain a list of methods available.

```
>>> dir("hello")
```

29. Now use the help function to get a breakdown of how the `upper()` function works.

```
>>> help(str.upper)
```

30. Press q to quit. Repeat as many times as you'd like. Check out the `split()` method.

```
>>> help(str.split)
```

31. Again, press q to quit. You might also try getting help on some of the list methods (used for working with lists).

```
>>> dir(list)
```

32. And you might also get some help on the dictionary methods (used for working with dictionaries).

```
>>> dir(dict)
```

33. Get help on how the dictionary method `.keys()` works.

```
>>> help(dict.keys)
```

34. Exit the python interpreter.

```
>>> exit()
```

35. A perfect demonstration might be those situations in which we need to read it from APIs. The following playbook uses the `uri` module to send an HTTP GET to the described API. The returned JSON is then parsed and python is used within the playbook to manipulate variable data. The data set used in this example is provided here, visit the data set now: https://statsapi.web.nhl.com/api/v1/teams

36. Create a playbook.

```
student@bchd:~/ans$ vim ~/ans/playbook-varspractice04.yml
```

37. Create the following solution. This solution not only uses Python to manipulate data, but also uses playbook tags. If you haven't yet studied Python tags, you might want to study them before studying the following.

```
---
- name: A playbook that manipulates data from uri
  hosts: localhost

  vars:
    uritouse: "https://statsapi.web.nhl.com/api/v1/teams"

  tasks:

  - name: Use URI module to send HTTP GET
    uri:
      method: GET
      url: "{{ uritouse }}"
    register: results
    tags:
      - keys
      - upper
      - stringslice

  - name: Use python to show all dictionary keys with python
    debug:
      var: results.keys()   # .keys() is a python dict method
    tags:
      - keys

  - name: Loop across the teams and display the team locations made uppercase with python
    debug:
      var: item.venue.name.upper()  # .upper() is a python string method
    loop: "{{ results.json.teams }}"
    tags:
      - upper

  # transform http://pittsburghpenguins.com/ into pittsburghpenguins.com
  - name: Loop across the teams and write out a lite of URLs without the http:// or trailing /
    shell: "echo {{ item.officialSiteUrl.lstrip('http://').rstrip('/') }} >> /tmp/teamsites.txt"
    loop: "{{ results.json.teams }}"
    tags:
      - stringslice
```

38. Save and exit. In this playbook, notice the abundance of Python methods being used to manipulate data. Run your playbook and observe the use of the python `.keys()` method.

```
student@bchd:~/ans$ ansible-playbook ~/ans/playbook-varspractice04.yml --tags "keys"
```

39. Now run your playbook and observe the use of the python `.upper()` method.

```
student@bchd:~/ans$ ansible-playbook ~/ans/playbook-varspractice04.yml --tags "upper"
```

40. Now run your playbook and observe the use of the python `.lstrip()` and `.rstrip()` methods.

```
student@bchd:~/ans$ ansible-playbook ~/ans/playbook-varspractice04.yml --tags "stringslice"
```

41. See if your playbook produced `/tmp/teamsite.txt` without any `https://` or `/`

```
student@bchd:~/ans$ cat /tmp/teamsites.txt
```

42. Great! That's it for this job.

39. ansible-doc

Lab Objective

Learning to use Ansible can be made easier by mastering the help utility known as `ansible-doc`. Where you can always reach for https://doc.ansible.com for guidance, it takes time to take your hands off the keyboard, open a browser, and "click around". Instead, it is recommended you build a dependence on using the `ansible-doc` utility, which will display the same information as the webpages, only more efficiently. Because the `ansible-doc` tool is reading the documentation directly from the plugin code actually installed on your system, it is likely to be "more accurate" than things you might find on the web.

40. https://docs.ansible.com/ansible/latest/cli/ansible-doc.html

Procedure

1. Move into your home directory.

   ```
   student@bchd:~$ cd ~
   ```

2. First get the help menu for `ansible-doc`

   ```
   student@bchd:~$ ansible-doc -h
   ```

3. Start by getting help on the Python plugin type (`-t`) "module", specifically the `mount` module.

   ```
   student@bchd:~$ ansible-doc -t module mount
   ```

4. Scroll up and down with the arrow keys. Press `q` to quit.

5. The `-t module` or `--type module` is actually the default parameter passed to `ansible-doc`

   ```
   student@bchd:~$ ansible-doc mount
   ```

6. Scroll up and down with the arrow keys. Press `q` to quit.

7. Return just the section of the help document that contains playbook examples (`-s` is for snippet).

   ```
   student@bchd:~$ ansible-doc -s mount
   ```

8. Scroll up and down with the arrow keys. Press `q` to quit.

9. Get information about the technology type `connection`, specifically, `ssh`

   ```
   student@bchd:~$ ansible-doc -t connection ssh
   ```

10. Scroll up and down with the arrow keys. Press `q` to quit.

11. Get information about the technology type `connection`, specifically, `local`

    ```
    student@bchd:~$ ansible-doc -t connection local
    ```

12. Scroll up and down with the arrow keys. Press `q` to quit.

13. Get information about the technology type `connection`, specifically, `network_cli`

    ```
    student@bchd:~$ ansible-doc -t connection network_cli
    ```

14. Scroll up and down with the arrow keys. Press `q` to quit.

15. Get information about the technology type `lookup`, specifically, `env`

    ```
    student@bchd:~$ ansible-doc -t lookup env
    ```

16. Scroll up and down with the arrow keys. Press `q` to quit.

17. How would we know what kind of lookup plugins even exist? Get a list (`-l`) of all of them.

```
student@bchd:~$ ansible-doc -t lookup -l
```

18. Scroll up and down with the arrow keys. Press `q` to quit.

19. That's a bit much. Remove some of the results by providing a namespace, `community.general`

```
student@bchd:~$ ansible-doc -t lookup -l community.general
```

20. Scroll up and down with the arrow keys. Press `q` to quit.

21. Remove some of the results by providing a namespace, `community.kubernetes`

```
student@bchd:~$ ansible-doc -t lookup -l community.kubernetes
```

22. Scroll up and down with the arrow keys. Press `q` to quit.

23. Get help on how to use the type of plugin, lookup, specifically, community.kubernetes.k8s

```
student@bchd:~$ ansible-doc -t lookup community.kubernetes.k8s
```

24. Scroll up and down with the arrow keys. Press `q` to quit.

25. Get a list of every single Ansible module typed plugin.

```
student@bchd:~$ ansible-doc -l
```

26. Hmm, that was a bit much! Get a list of those relating to Dell!

```
student@bchd:~$ ansible-doc -l dellemc
```

27. Get all of those within the `community.general` namespace.

```
student@bchd:~$ ansible-doc -l community.general
```

28. Get information about the module `community.general.jenkins_job`.

```
student@bchd:~$ ansible-doc community.general.jenkins_job
```

29. Great! That's it for this lab. Moving forward, try to rely on `ansible-doc` for getting help.

41. Ansible Lookup Plugin

Lab Objective

In this lab, you will explore the Ansible Lookup Plugins. The name sounds a bit more dramatic than it is, the TLDR on Lookup Plugins is they are used to return data from an outside source. Source types include csvfiles, flat files, URLs, mongoDB, rabbitMQ, and many more.

Looking up the contents of a file is as simple as, `{{ lookup("file", "/tmp/file.txt") }}` where the first value, `file` describes the type of data to be accessed, followed by the path.

Review more about Lookup Plugins here: https://docs.ansible.com/ansible/latest/plugins/lookup.html

A list of the types of Plugins available are maintained here: https://docs.ansible.com/ansible/latest/plugins/lookup.html#plugin-list

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. This lab uses the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the teardown script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id bender@10.10.2.3
```

7. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id fry@10.10.2.4
```

8. To ssh into these machines without a password, we will need to copy over a key. Type `alta3` when prompted.

```
student@bchd:~$ ssh-copy-id zoidberg@10.10.2.5
```

9. Farnsworth has some issues with copying his key, so we'll connect with a password.

10. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

11. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

12. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following)

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender ansible_python_interpreter=/usr/bin/python3
fry         ansible_host=10.10.2.4 ansible_user=fry ansible_python_interpreter=/usr/bin/python3
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg ansible_python_interpreter=/usr/bin/python3
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth ansible_ssh_pass=alta3
```

13. Save and exit.

14. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

15. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

16. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

17. Save and exit with `:wq`

18. Suppose we wanted to harvest from JSON data from an API, and then parse out that data. Start by navigating to the link http://api.open-notify.org/astros.json and review the returned data (JSON). This highly available real-time data-set can be returned to Ansible by using `lookup` with the `url` plugin.

19. Create a basic playbook file called `~/ans/playbook-lookup-url.yml`.

```
student@bchd:~$ vim ~/ans/playbook-lookup-url.yml
```

20. Create the following playbook.

```
- name: Ansible and RESTful interfaces
  hosts: localhost
  gather_facts: false

  vars:
          astros: "{{ lookup('url', 'http://api.open-notify.org/astros.json') }}"
          spacecowboys: []

  tasks:
          - name: Quick iss api checkup
            debug:
                    msg: "Today on the ISS are: {{ astros.people }}"

          - name: Loop through our data and display each dict (name and craft)
            debug:
                    msg: "Loop across ISS data: {{ item }}"
            loop: "{{ astros.people }}"

          - name: Loop through our data and display ONLY first and last names
            debug:
                    msg: "Astro names are: {{ item.name }}"
            loop: "{{ astros.people }}"

          - name: Create a new list with ansible of astro names
            set_fact:
                    spacecowboys: "{{ astros.people|map(attribute='name')|list }}"

          - name: Only display the first names of the astros
            debug:
                    msg: "{{ item.split()[0] }}"
            loop: "{{ spacecowboys }}"
```

21. Save and exit.

22. Run your playbook. Study the results and ensure you understand why the playbook is producing the values that it is.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-lookup-url.yml
```

23. Very well done! That's it for this lab.

42. Ansible Callback Plugins

Lab Objective

The objective of this lab is to learn how to control the output of the playbook from invocation until the **PLAY RECAP** statistics. Suppose you wanted to simplify the output for those individuals that might only be running playbooks. Or maybe you want the results of the playbook returned as JSON. Callback plugins are the solution.

Callback plugins may be activated one at a time from within `ansible.cfg`. Below are several permutations to test. The list of possible plugins can be found within the Ansible Documentation on callbacks.

Read about "callback" plugins here:
https://docs.ansible.com/ansible/latest/plugins/callback.html

Procedure

1. Ensure you are in `/home/student/` (your home folder).

```
student@bchd:~$ cd ~
```

2. Be sure you have the `~/ans` directory. We use this directory to store our playbooks, this helps keep us organized.

```
student@bchd:~$ mkdir ~/ans
```

3. Create a playbook, `~/ans/playbook-littletest.yml`

```
student@bchd:~$ vim ~/ans/playbook-littletest.yml
```

4. Create the following simple playbook.

```
---
- name: Debug playbook to display ansible facts
  hosts: localhost
  gather_facts: yes

  tasks:
          # package attempts to identify the package handler on the local system
          # and use the correct module
          # however each repo may name package in a unique manner so you still
          # may with to limit a task with the when: ansible_distribution conditional
          # examples: yum, apt, dnf, pkg, etc.
          - name: Install an app with apt only on Ubuntu systems
            package:
                    state: present
                    name: sl
            when: ansible_distribution == "Ubuntu"
            become: yes

          # print out the value of ansible_distribution
          # (provided by gather_facts)
          - name: Print out both users distro
            debug:
                    var: ansible_distribution

          # print out the value of ansible_distribution_version
          - name: Print out both users distro ver
            debug:
                    var: ansible_distribution_version
```

5. Save and exit.
6. From within a terminal window, edit your `~/.ansible.cfg` file. Remember that this file contains the global settings we want to associate with Ansible at runtime.

```
student@bchd:~$ vim ~/.ansible.cfg
```

7. Make sure your `~/.ansible.cfg` has the following settings. It is okay to have more than this, but you must have *at least* the following.

```
[defaults]
host_key_checking=no
nocows=True

# DENSE CALLBACK
# activate callback plugin (one at a time)
#stdout_callback = dense

# JSON CALLBACK
# activate callback plugin (one at a time)
#stdout_callback = json

# NULL CALLBACK
# activate callback plugin (one at a time)
#stdout_callback = null

# LOG_PLAY CALLBACK
# activate callback plugin (one at a time)
#stdout_callback = log_plays
[callback_log_plays]
# this is a FOLDER not a file. Ansible will create a log file PER host
log_folder = /tmp/hosts/
```

8. Save and exit.
9. Run your playbook. It should execute with typical output.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-littletest.yml
```

10. Edit your `~/.ansible.cfg` file.

```
student@bchd:~$ vim ~/.ansible.cfg
```

11. Un-comment the following line. This will activate the **dense** callback. If you'd like, you can read about it here: https://docs.ansible.com/ansible/latest/plugins/callback/dense.html

```
stdout_callback = dense
```

12. Save and exit.
13. Run your playbook. It should execute with a **dense** output (only a few lines will appear on the screen).

```
student@bchd:~$ ansible-playbook ~/ans/playbook-littletest.yml
```

14. Edit your `~/.ansible.cfg` file.

```
student@bchd:~$ vim ~/.ansible.cfg
```

15. You may only have **one** `stdout_callback` defined at a time. Comment out `#stdout_callback = dense` and then un-comment `stdout_callback = json` which will activate the **json** callback. If you'd like, you can read about it here: https://docs.ansible.com/ansible/latest/plugins/callback/json.html

```
# DENSE CALLBACK
# activate callback plugin (one at a time)
# stdout_callback = dense

# JSON CALLBACK
# activate callback plugin (one at a time)
stdout_callback = json
```

16. Once you have made the change, save and exit.
17. Run your playbook. It should execute with a **json** output to the screen.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-littletest.yml
```

18. Edit your `~/.ansible.cfg` file.

```
student@bchd:~$ vim ~/.ansible.cfg
```

19. Remember, **one** `stdout_callback` at a time. Comment out `#stdout_callback = json` and then un-comment `stdout_callback = null` which will activate the **null** callback. Nothing will be printed to the screen. If you'd like, you can read about it here: https://docs.ansible.com/ansible/latest/plugins/callback/null.html

```
# JSON CALLBACK
# activate callback plugin (one at a time)
#stdout_callback = json

# NULL CALLBACK
# activate callback plugin (one at a time)
stdout_callback = null
```

20. Once you have made the change, save and exit.
21. Run your playbook. It should execute with a **null** output to the screen. This callback plug in makes nothing display the screen.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-littletest.yml
```

22. Edit your `~/.ansible.cfg` file.

```
student@bchd:~$ vim ~/.ansible.cfg
```

23. Comment out `#stdout_callback = null` and then un-comment `stdout_callback = log_plays`, `[callback_log_plays]` and `log_folder = /var/log/ansible/hosts`. Notice that [call_log_plays] it's own section of the `ansible.cfg`, so you may not have additional `[defaults]` settings below it. These changes will activate the **log_plays** callback. All of the output will be stored in a folder according to host. If you'd like, you can read about it here: https://docs.ansible.com/ansible/latest/plugins/callback/log_plays.html

```
# NULL CALLBACK
# activate callback plugin (one at a time)
#stdout_callback = null

# LOG_PLAY CALLBACK
# activate callback plugin (one at a time)
stdout_callback = log_plays
[callback_log_plays]
# this is a FOLDER not a file. Ansible will create a log file PER host
log_folder = /tmp/hosts/
```

24. Once you have made the change, save and exit.
25. Great! Run your playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-littletest.yml
```

26. Check out the results of your playbook.

```
student@bchd:~$ cd /tmp/hosts/
```

27. cat out the file. It should have the results of the playbook inside.

```
student@bchd:~/tmp/hosts$ cat localhost
```

28. Return to your home directory.

```
student@bchd:~/tmp/hosts$ cd
```

29. Before you sign off of this lab, be sure to comment out all of the **log_play** settings! This will return Ansible to the **default** output type. Read about the default Ansible callback here: https://docs.ansible.com/ansible/latest/plugins/callback/default.html

30. Great job! That's it for this lab.

43. Ansible Plugin System

Lab Objective

Plugins augment Ansible's core functionality such as parsing, loading inventory and Playbooks, running Playbooks and reading the results. As expected, plugins are written in Python. And while Ansible ships with a number of handy plugins, and you can easily write your own.

Ansible Plugins fall into the following categories:

- action
- become
- cache
- callback
- cliconf
- connection
- httpapi
- inventory
- lookup
- netconf
- shell
- strategy
- vars
- filters
- tests

In this lab, we'll explore some of the more critical plugins available to Ansible.

Procedure

1. To begin, run the `ansible --version` command to get some hints on where to begin looking for the Ansible plugins.

   ```
   ansible --version
   ```

2. Looks like we might have some luck if we move into `/usr/local/lib/python3.6/dist-packages/ansible`. Move into that directory now.

   ```
   cd /usr/local/lib/python3.6/dist-packages/ansible
   ```

3. Look at the directory structure.

   ```
   ls
   ```

4. Move into `plugins/`

   ```
   cd plugins/
   ```

5. Let's begin our understanding of plugins with **Action Plugins**. An action plugin runs *every time* you run a module. They represent a layer between the executor engine and the module, **allowing for controller-side actions to be taken before the module is executed**.

   ```
   cd action/
   ```

6. List the plugin files.

   ```
   ls
   ```

7. Let's try walking through how the `template` module would work. To start, review the usage via https://docs.ansible.com/ansible/latest/modules/template_module.html

8. Another way to check out your Ansible modules is to use `ansible-doc -l`

   ```
   ansible-doc -l
   ```

9. Scroll up / down until you find the `template` module.

Press `q` to quit.
10.
11. We can get better resolution on the `template` module with the `ansible-doc` tool.

    ```
    ansible-doc template
    ```

12. Reviewing the documentation, you'll notice about half-way down the page, it makes notice that, **note: This module has a corresponding action plugin.**

13. Press `q` to quit.

14. If Ansible finds an action plugin with the same name as the module, that action plugin is used, otherwise the `normal` action plugin is used. *NOTE: Tasks which use 'async' have a special action plugin, which is used to launch the task using the 'async_wrapper' module.* Notice that within the plugin directory, we have a file called, `template.py`. Let's review it.

    ```
    less template.py
    ```

15. After the plugin code is executed, Ansible would next execute the module with the matching name, in this case `template.py`, within the configured module directory. Check that out now.

    ```
    cd ../modules
    ```

16. Display the module `template.py`.

    ```
    cat template.py
    ```

17. Notice that it is *mostly* doc strings? That's because everything was handled by the template action plugin. The template action plugin itself creates the template file locally as a temporary file, and then uses the copy or file modules to push it out to the target system. Move back into `../plugins/action`

    ```
    cd ../action
    ```

18. If a module name does not match any of the files within `plugins/action`, than the `normal` plugin is run. Let's take a look at that now.

    ```
    less normal.py
    ```

19. The following code is the entirety of the 'normal' action plugin.

```
# code from plugins/action/normal.py
# (comments and most blank lines have been removed for brevity)

from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

from ansible.plugins.action import ActionBase
from ansible.utils.vars import merge_hash


class ActionModule(ActionBase):

    def run(self, tmp=None, task_vars=None):

        # individual modules might disagree but as the generic the action plugin, pass at this point.
        self._supports_check_mode = True
        self._supports_async = True

        result = super(ActionModule, self).run(tmp, task_vars)
        del tmp  # tmp no longer has any effect

        if not result.get('skipped'):

            if result.get('invocation', {}).get('module_args'):
                # avoid passing to modules in case of no_log
                # should not be set anymore but here for backwards compatibility
                del result['invocation']['module_args']

            # FUTURE: better to let _execute_module calculate this internally?
            wrap_async = self._task.async_val and not self._connection.has_native_async

            # do work!
            result = merge_hash(result, self._execute_module(task_vars=task_vars, wrap_async=wrap_async))

            # hack to keep --verbose from showing all the setup module result
            # moved from setup module as now we filter out all _ansible_ from result
            if self._task.action == 'setup':
                result['_ansible_verbose_override'] = True

        if not wrap_async:
            # remove a temporary path we created
            self._remove_tmp_path(self._connection._shell.tmpdir)

        return result
```

20. Exit less with `q`

21. New action plugins usually only need to subclass ActionBase and override the 'run()' method.

22. Within a plugin, modules may be executed remotely by the '_execute_module()' method, which can also accept other parameters to allow you to run more than one module remotely to create some complex actions. For example, the 'template' action uses the 'copy' and 'file' modules to do the real work of copying the templated file to the remote system. Take a peek at `plugins/action/template.py` again.

    ```
    less template.py
    ```

23. The following is the snippet you're looking for within `template.py`

```
# from plugins/action/template.py
# L175-184 on the devel branch at the time of writing

        # run the copy module
        new_module_args.update(
            dict(
                src=xfered,
                dest=dest,
                original_basename=os.path.basename(source),
                follow=True,
            ),
        )
        result.update(
            self._execute_module(
                module_name='copy',             # copy module
                module_args=new_module_args,
                task_vars=task_vars,
                tmp=tmp,
                delete_remote_tmp=False,
            )
        )
```

24. The above code is run after a temporary file is generated using the templating engine, which all occurs on the Ansible controller side. Using other modules in this manner allows us to avoid duplicating code and is very common in modules related to file operations.

25. Press `q` to exit.

26. Time to move on to **Callback Plugins**. One of the more heavily developed plugins, callbacks provide a way to react to events which occur during the execution of Playbooks. Move into `../callback`

    ```
    cd ../callback
    ```

27. Ansible can load multiple callbacks, however, we differentiate between callbacks which send output to the screen and those that don't. This allows us to ensure output to the screen is legible. Callbacks are configured via a white list, configurable in `ansible.cfg` or via an environment variable `ANSIBLE_CALLBACK_WHITELIST`.

28. Suppose you wanted to direct output to a Slack channel. The `slack` plugin could be used to achieve this.

    ```
    less slack.py
    ```

29. The following is the `slack.py` callback plugin.

```
from ansible.constants import mk_boolean
from ansible.module_utils.urls import open_url
from ansible.plugins.callback import CallbackBase

try:
    import prettytable
    HAS_PRETTYTABLE = True
except ImportError:
    HAS_PRETTYTABLE = False

class CallbackModule(CallbackBase):
    """This is an ansible callback plugin that sends status
    updates to a Slack channel during playbook execution.

    This plugin makes use of the following environment variables:
        SLACK_WEBHOOK_URL (required): Slack Webhook URL
        SLACK_CHANNEL     (optional): Slack room to post in. Default: #ansible
        SLACK_USERNAME    (optional): Username to post as. Default: ansible
        SLACK_INVOCATION  (optional): Show command line invocation
                                      details. Default: False

    Requires:
        prettytable

    """
    CALLBACK_VERSION = 2.0
    CALLBACK_TYPE = 'notification'
    CALLBACK_NAME = 'slack'
    CALLBACK_NEEDS_WHITELIST = True

...

    def send_msg(self, attachments):
        payload = {
            'channel': self.channel,
            'username': self.username,
            'attachments': attachments,
            'parse': 'none',
            'icon_url': ('http://www.ansible.com/hs-fs/hub/330046/file-449187601-png/ansible_badge.png'),
        }

        data = json.dumps(payload)
        self._display.debug(data)
        self._display.debug(self.webhook_url)
        try:
            response = open_url(self.webhook_url, data=data)
            return response.read()
        except Exception as e:
            self._display.warning('Could not submit message to Slack: %s' %
                                  str(e))

    def v2_playbook_on_play_start(self, play):
        """Display Play start messages"""

        name = play.name or 'Play name not specified (%s)' % play._uuid
        msg = '*Starting play* (_%s_)\n\n*%s*' % (self.guid, name)
        attachments = [
            {
                'fallback': msg,
                'text': msg,
```

```
                'color': 'warning',
                'mrkdwn_in': ['text', 'fallback', 'fields'],
            }
        ]
        self.send_msg(attachments=attachments)
```

30. Callback plugins have very many entry points, which are triggered at various points in the executor engine. For a full listing, see the stubbed methods defined in CallbackBase (located in the 'plugins/callback/**init**.py' file).

31. Press `q` to quit.

32. In addition to action and callback plugins, there are **connection plugins**.

```
cd ../connection
```

33. Take a look at the selections.

```
ls
```

34. Everything in this list are things ansible can "control". A few pop out, `ssh.py` (OpenSSH), `local.py`, `network_cli.py`, and `paramiko_ssh.py`. By default, Ansible uses native OpenSSH, because it supports ControlPersist (a performance feature), Kerberos, and options in ~/.ssh/ config such as Jump Host setup. If your control machine uses an older version of OpenSSH that does not support ControlPersist, Ansible will fallback to a Python implementation of OpenSSH called 'paramiko'. If you'd like, read more about connection plugins with the `ansible-doc` command.

```
ansible-doc -t connection -l
```

35. Let's try quickly using a `connection` plugin you might not have experience with. Notice the connection plugin called `docker.py`. This plugin allows Ansible to control Docker. To start, build a container with Docker.

```
sudo docker run -d --name=mycontainer -e FOO=bar alpine:latest sleep 600
```

36. Now try using `ansible` on `all` of the inventory defined by comma separated host `mycontainer,` using the connection plugin `docker` and the argument `echo $FOO` passed to the module `raw`.

```
sudo ansible all -i 'mycontainer,' -c docker -m raw -a 'echo $FOO'
```

37. Stop the container

```
sudo docker stop mycontainer
```

38. Now that the container is stopped, we can delete it.

```
sudo docker rm mycontainer
```

39. Connection plugins are used in the execution of every task, as they provide the transport layer between the Ansible controller and managed hosts. The simple API includes five methods:

    - `connect`
    - `exec_command`
    - `put_file`
    - `get_file`
    - `disconnect`

40. These make it very easy to write connection plugins to connect to hosts via a wide range of methods. The good news is, the majority of connections you'd want to make have already been written. Whether it's SSH, LXC, chroot, Docker, and so on, Ansible to take control of.

```
ls
```

41. Look at the `croot` connection plugin.

```
less chroot
```

42. The following code snippet shows some code from the 'chroot' connection plugin:

```
# from plugins/connection/chroot.py
...

class Connection(ConnectionBase):
    ''' Local chroot based connections '''

    transport = 'chroot'
    has_pipelining = True
    # su currently has an undiagnosed issue with calculating the file
    # checksums (so copy, for instance, doesn't work right)
    # Have to look into that before re-enabling this
    become_methods = frozenset(C.BECOME_METHODS).difference(('su',))

...

    def _buffered_exec_command(self, cmd, stdin=subprocess.PIPE):
        ''' run a command on the chroot.  This is only needed for implementing
        put_file() get_file() so that we don't have to read the whole file
        into memory.

        compared to exec_command() it loses some niceties like being able to
        return the process's exit code immediately.
        '''
        executable = C.DEFAULT_EXECUTABLE.split()[0] \
                     if C.DEFAULT_EXECUTABLE else '/bin/sh'
        local_cmd = [self.chroot_cmd, self.chroot, executable, '-c', cmd]

        display.vvv("EXEC %s" % (local_cmd), host=self.chroot)
        local_cmd = \
          [to_bytes(i, errors='surrogate_or_strict') for i in local_cmd]
        p = subprocess.Popen(local_cmd, shell=False, stdin=stdin,
                stdout=subprocess.PIPE, stderr=subprocess.PIPE)

        return p

    def exec_command(self, cmd, in_data=None, sudoable=False):
        ''' run a command on the chroot '''
        super(Connection, self).exec_command(
            cmd,
            in_data=in_data,
            sudoable=sudoable,
        )

        p = self._buffered_exec_command(cmd)

        stdout, stderr = p.communicate(in_data)
        return (p.returncode, stdout, stderr)

...
```

43. The 'put_file' and 'fetch_file' methods also make use of the '_buffered_exec_command()' method shown above to move files in and out of the chroot. But to Ansible, this looks just like a remote host.

44. The **strategy plugins** allow users to control the order and manner of execution of tasks on hosts. Move into `/usr/local/lib/python3.6/dist-packages/ansible/plugins/strategy/`

```
cd /usr/local/lib/python3.6/dist-packages/ansible/plugins/strategy/
```

45. Look at the files here.

```
ls
```

46. Ansible currently includes four strategies:
    - **linear** - All hosts in the inventory must complete each task before any move on to running the next task. Some users objected to the linear strategy because it slows the system, so the "free" strategy was introduced.
    - **free** - This allows each host to execute the tasks in its list as quickly as possible without waiting for other hosts to complete the same task.
    - **debug** - The debug strategy turns Ansible into an interactive debugger.
    - **host_pinned** - Ansible will not start a play for a host unless the play can be finished without interruption by tasks for another host, i.e. the number of hosts with an active play does not exceed the number of forks.
47. You can take a look at any of the code for the strategy plugins if you'd like.

    ```
    less linear.py
    ```

48. Press `q` to exit when you are finished.
49. The following code shows the 'debug' strategy, which turns the 'linear' strategy into an interactive debugger:

```
from ansible.plugins.strategy.linear import StrategyModule as LinearStrategyModule

...

class NextAction(object):
    """ The next action after an interpreter's exit. """
    REDO = 1
    CONTINUE = 2
    EXIT = 3
    def __init__(self, result=EXIT):
        self.result = result

class StrategyModule(LinearStrategyModule):
    def __init__(self, tqm):
        self.curr_tqm = tqm
        super(StrategyModule, self).__init__(tqm)

    def _queue_task(self, host, task, task_vars, play_context):
        self.curr_host = host
        self.curr_task = task
        self.curr_task_vars = task_vars
        self.curr_play_context = play_context

        super(StrategyModule, self)._queue_task(
            host,
            task,
            task_vars,
            play_context,
        )

    def _process_pending_results(self,
        iterator,
        one_pass=False,
        max_passes=None):
        if not hasattr(self, "curr_host"):
            return super(
                StrategyModule,
                self
            )._process_pending_results(iterator, one_pass, max_passes)

        prev_host_state = iterator.get_host_state(self.curr_host)
        results = super(
            StrategyModule,
            self)._process_pending_results(iterator, one_pass)

        while self._need_debug(results):
            next_action = NextAction()
            dbg = Debugger(self, results, next_action)
            dbg.cmdloop()

            if next_action.result == NextAction.REDO:
                # rollback host state
                self.curr_tqm.clear_failed_hosts()
                iterator._host_states[self.curr_host.name] = prev_host_state
                if reduce(lambda total, res : res.is_failed() or \
                    total, results, False):
                    self._tqm._stats.failures[self.curr_host.name] -= 1
                elif reduce(lambda total, res : res.is_unreachable() or \
                    total, results, False):
                    self._tqm._stats.dark[self.curr_host.name] -= 1
```

```
                # redo
                super(StrategyModule, self)._queue_task(
                    self.curr_host,
                    self.curr_task,
                    self.curr_task_vars,
                    self.curr_play_context
                )
                results = super(
                    StrategyModule,
                    self
                )._process_pending_results(iterator, one_pass)
            elif next_action.result == NextAction.CONTINUE:
                break
            elif next_action.result == NextAction.EXIT:
                exit(1)

        return results

    def _need_debug(self, results):
        return reduce(lambda total, res : res.is_failed() or \
               res.is_unreachable() or total, results, False)

class Debugger(cmd.Cmd):
    prompt = '(debug) '  # debugger
    prompt_continuous = '> '  # multiple lines

    def __init__(self, strategy_module, results, next_action):
        # cmd.Cmd is old-style class
        cmd.Cmd.__init__(self)

        self.intro = "Debugger invoked"
        self.scope = {}
        self.scope['task'] = strategy_module.curr_task
        self.scope['vars'] = strategy_module.curr_task_vars
        self.scope['host'] = strategy_module.curr_host
        self.scope['result'] = results[0]._result
        self.scope['results'] = results  # for debug of this debugger
        self.next_action = next_action
...
```

50. The 'debug' strategy simply sub-classes the 'linear' strategy class and overrides two methods defined in StrategyBase- 'queue_task' (which handles starting a worker to run the task) and '_process_pending_results()' (which reads results back from workers). The bulk of the work is done in the later, which invokes the interactive Debugger class when a failed task result is encountered and allows the user to do things like retry the task or modify internal Ansible variables and state.

51. Exit with `q`

52. Using the 'debug' strategy is very useful when writing new Playbooks and roles and can drastically reduce the number of times you have to re-run things while you're getting your procedures straightened out.

53. Moving our attention to the **lookup plugins**

    ```
    cd /usr/local/lib/python3.6/dist-packages/ansible/plugins/lookup/
    ```

54. Lookup plugins are ways to tightly integrate python within your code, primarily to "lookup" values that maybe stored in difficult to reach places. Consider the following lookup plugin usage

```
- name: My playbook
  hosts: localhost
    vars:
      firstvar: {{lookup(`pipe', `/usr/bin/whoami')}}
      secondvar: {{lookup(`etcd', `somekey')}}   # this allows you to fetch a key out of a etcd store.

  tasks:
      ...
```

55. A couple of caveats to bear in mind about lookup plugins are that they always execute on the Ansible controller, not on a remote system. Furthermore, they are always expected to return a list of items, because of their potential use with loops.

56. Look at the various lookup plugins available

```
ls
```

57. The following code shows the 'pipe' lookup as shown above, which allows the user to fetch the output from a command-line program and store it in the variable:

```
# from plugins/lookup/pipe.py

from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

import subprocess

from ansible.errors import AnsibleError
from ansible.plugins.lookup import LookupBase

class LookupModule(LookupBase):
    def run(self, terms, variables, **kwargs):
        ret = []
        for term in terms:
            term = str(term)
            p = subprocess.Popen(
                term,
                cwd=self._loader.get_basedir(),
                shell=True,
                stdin=subprocess.PIPE,
                stdout=subprocess.PIPE
            )
            (stdout, stderr) = p.communicate()
            if p.returncode == 0:
                ret.append(stdout.decode("utf-8").rstrip())
            else:
                raise AnsibleError(
                    "lookup_plugin.pipe(%s) returned %d" % \
                    (term, p.returncode)
                )
        return ret
```

58. The **Filter and test plugin** classes extend the Jinja2 templating system Ansible uses for variables.

```
cd /usr/local/lib/python3.6/dist-packages/ansible/plugins/test/
```

59. Check out the files here.

```
ls
```

60. Check out the filters.

```
cd /usr/local/lib/python3.6/dist-packages/ansible/plugins/filter/
```

61. Look at the various filters.

```
ls
```

62. In short, these allow you to do some cool things with data transformation and testing of the value or type of variable being run through the Jinja2 engine. Take a peek at those now

```
{{nextgen|int}}            # transform the var nextgen into an into
{{ncc|default('1701D')}}   # if var ncc does not have a value than give it 1701D
```

63. Tests are used to validate data:

```
{{foo is defined}}
```

64. The following code is an example of a filter which allows you to query JSON data using the jmespath query language:

```
# from plugins/filter/json_query.py

from ansible.errors import AnsibleError
from ansible.plugins.lookup import LookupBase
from ansible.utils.listify import listify_lookup_plugin_terms

try:
    import jmespath
    HAS_LIB = True
except ImportError:
    HAS_LIB = False

def json_query(data, expr):
    if not HAS_LIB:
        raise AnsibleError(
            'You need to install "jmespath" prior to running '
            'json_query filter'
        )
    return jmespath.search(expr, data)

class FilterModule(object):
    ''' Query filter '''
    def filters(self):
        return {
            'json_query': json_query
        }
```

65. Part of the templating variable system, **cache plugins** are used to store gathered facts outside of local memory. This is important because, by default, Ansible uses the in-memory cache plugin which can cause problems if your process involves running several individual Playbooks and you need fact data. In such instances, each of those individual runs would need to regather those facts because they only reside in memory as long as a Playbook is running. In addition to the in-memory default, Ansible includes cache plugins to store fact data in memcached and Redis, or even just a flat JSON file.

```
cd /usr/local/lib/python3.6/dist-packages/ansible/plugins/cache/
```

66. Look at the various filters.

```
ls `
```

67. The following code shows the 'memory' cache plugin (which is the default as noted above):

```
# from plugins/cache/memory.py

from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

from ansible.plugins.cache.base import BaseCacheModule

class CacheModule(BaseCacheModule):
    def __init__(self, *args, **kwargs):
        self._cache = {}
    def get(self, key):
        return self._cache.get(key)
    def set(self, key, value):
        self._cache[key] = value
    def keys(self):
        return self._cache.keys()
    def contains(self, key):
        return key in self._cache
    def delete(self, key):
        del self._cache[key]
    def flush(self):
        self._cache = {}
    def copy(self):
        return self._cache.copy()
```

68. For those who are familiar with Python internals, you may notice that cache plugins pretty much implement a dictionary interface.

69. **Shell plugins** are used to properly format commands for remote execution (quoting, escaping, logic, etc.). They were originally written to simplify the handling of ssh vs. winrm execution, but more plugins have been added for other shells (csh, fish, dash, to name a few).

```
cd /usr/local/lib/python3.6/dist-packages/ansible/plugins/shell/
```

70. Look at the various shell plugins.

```
ls
```

71. Each connection plugin has a default shell plugin, for instance the winrm connection defaults to PowerShell. The following code shows the 'csh' shell plugin:

```
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

from ansible.plugins.shell import ShellBase

class ShellModule(ShellBase):

    # Common shell filenames that this plugin handles
    COMPATIBLE_SHELLS = frozenset(('csh', 'tcsh'))
    # Family of shells this has.  Must match the filename without extension
    SHELL_FAMILY = 'csh'

    # How to end lines in a python script one-liner
    _SHELL_EMBEDDED_PY_EOL = '\\\n'
    _SHELL_REDIRECT_ALLNULL = '>& /dev/null'
    _SHELL_AND = '&&'
    _SHELL_OR = '||'
    _SHELL_SUB_LEFT = '"`'
    _SHELL_SUB_RIGHT = '`"'
    _SHELL_GROUP_LEFT = '('
    _SHELL_GROUP_RIGHT = ')'

    def env_prefix(self, **kwargs):
        return 'env %s' % super(ShellModule, self).env_prefix(**kwargs)
```

72. Mainly, shell plugins override class-level variables. A few methods are also available for overriding, such as the 'env_prefix()' method above (which controls how environment variables are formatted for the given shell).

73. If Ansible isn't doing what you need it to do, write a plugin to make it do so! In most cases, it's very easy to add new plugins to extend the power of Ansible.

74. When writing a plugin, always use the provided base classes. The Ansible plugin loader (the main class responsible for finding and loading files with a specific plugin class) will ignore your plugin class if it doesn't have the proper base class as a parent object. The great thing about base classes is that they provide a ton of pre-written methods so you don't have to reinvent the wheel (or cargo-cult a bunch of code).

75. Many other plugin systems exist, and typically each Ansible release will include new plugins. Spend some time clicking through the other directories if you'd like.

    ```
    cd /usr/local/lib/python3.6/dist-packages/ansible/plugins/
    ```

76. That's it for this lab!

44. YAML, JSON, Dynamic, and Cloud Inventories

Lab Objective

The objective of this lab is to build inventory from dynamic sources. Scripts may be used to perform lookups within databases, parse files, query APIs, as well as pull from other sources, to ultimately construct an inventory that Ansible may use to run against.

Within cloud environments, users can build and delete virtual machines too rapidly to record to changes manually within an inventory file. Therefore, one popular application of dynamic inventorying, is when working with clouds such as Amazon (AWS) as well as OpenStack.

Here are some terms to add to your vocabulary:
Static Inventory - Users manually describe hosts within inventories
Dynamic Inventory - Scripts (with Python being the popular go to) are used to pull data from various sources
Hybrid Inventory - A combination of static and dynamic inventory techniques

Acceptable formats for inventories are:
- ini
- JSON
- YAML
- Comma separated host lists

Examples of 'comma separated hosts' lists include:

```
# define 2 hosts in command line
# ansible -i '10.10.2.6, 10.10.2.4' -m ping all

# DNS resolvable names
# ansible -i 'host1.example.com, host2' -m user -a 'name=me state=absent' all

# just use localhost
# ansible-playbook -i 'localhost,' play.yml -c local
```

Introduction to Dynamic Inventory:
https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html#other-inventory-scripts

Review documentation on inventory sources here:
https://docs.ansible.com/ansible/latest/dev_guide/developing_inventory.html#inventory-sources

Review the documentation on dynamic inventorying here:
https://docs.ansible.com/ansible/latest/dev_guide/developing_inventory.html

Procedure

1. We will begin this lab by exploring building our own **dynamic inventory** with Python.
2. Change directory to the `/home/student/` directory.

   ```
   student@bchd:~$ cd ~
   ```

3. Download the teardown script (you may already have this):

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
   ```

4. Run the teardown script.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

5. Download the setup script for this lab.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
   ```

6. Run the setup script for this lab.

   ```
   student@bchd:~$ bash pexpress-setup.sh
   ```

7. Ping the each of the below lab hosts.

```
student@bchd:~$ ping -c 1 10.10.2.3
student@bchd:~$ ping -c 1 10.10.2.4
student@bchd:~$ ping -c 1 10.10.2.5
student@bchd:~$ ping -c 1 10.10.2.6
```

8. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

9. Edit your inventory file.

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

10. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following).

```
[planetexpress]
bender      ansible_host=10.10.2.3 ansible_user=bender
fry         ansible_host=10.10.2.4 ansible_user=fry
zoidberg    ansible_host=10.10.2.5 ansible_user=zoidberg
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth dnsip=9.9.9.9 ansible_python_interpreter=/usr/bin/python
[planetexpress:vars]
dnsip=8.8.8.8
ansible_python_interpreter=/usr/bin/python3
ansible_ssh_pass=alta3
```

11. Save and exit.

12. Now let's describe that same inventory, only use the YAML format. Remember, YAML is JSON and JSON is YAML. So transcribing our INI format to YAML is a huge step in producing an inventory with JSON as well. Make a new directory we can store our YAML inventories in.

```
student@bchd:~$ mkdir -p ~/ans/inv/yaml/
```

13. Create a `~/ans/inv/yaml/hosts.yaml` file.

```
student@bchd:~$ vim ~/ans/inv/yaml/hosts.yaml
```

14. Create the following YAML file:

```
---
all:
        hosts:
                # if you had ungrouped hosts
                # describe them here
                # 10.10.2.2:
                # 10.10.1.1:
        children:
                planetexpress:
                        hosts:
                                bender:
                                        ansible_host: 10.10.2.3
                                        ansible_user: bender
                                fry:
                                        ansible_host: 10.10.2.4
                                        ansible_user: fry
                                zoidberg:
                                        ansible_host: 10.10.2.5
                                        ansible_user: zoidberg
                                farnsworth:
                                        ansible_host: 10.10.2.6
                                        ansible_user: farnsworth
                                        dnsip: 9.9.9.9
                                        ansible_python_interpreter: /usr/bin/python
                        vars:
                                dnsip: 8.8.8.8
                                ansible_python_interpreter: /usr/bin/python3
                                ansible_ssh_pass: alta3
```

15. Save and exit.

16. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

17. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

18. Save and exit.

19. Run an Ansible Ad-Hoc command to proof-of-concept our YAML inventory file.

```
student@bchd:~$ ansible all -i ~/ans/inv/yaml/hosts.yaml -m ping
```

20. You should see `ping pong` push and pull responses from each of the four hosts described within the YAML inventory.

21. Scripts may also be used to produce inventories, however, scripts will typically produce JSON. It's easier than it sounds. Create a new directory we can use to store our script.

```
student@bchd:~$ mkdir -p ~/ans/inv/hybridinv/
```

22. Create a python script within the new directory.

```
student@bchd:~$ vim ~/ans/inv/hybridinv/inventory.py
```

23. Create the following script:

```
#!/usr/bin/python3
"""Russell Zachary Feeser || rzfeeser@alta3.com"""

## for accepting arguments from user at CLI
import argparse

## required for working with JSON (pystd. library)
import json

def main():
    # this is what we will ultimately return
    inventory = {}

    # Called with `--list`
    if args.list:
        # inventory is result of calling example_inventory()
        inventory = example_inventory()
    # Called with `--host [hostname]`
    elif args.host:
        # Not implemented, since we return _meta info `--list`
        # Let's put API request logic here...
        inventory = empty_inventory()
    # If no groups or vars are present, return an empty inventory.
    else:
        inventory = empty_inventory()

    # print the result of inventory
    print(json.dumps(inventory))  # from the json library use the
                                  # DUMPString function, dumps()

# Example inventory for testing.
def example_inventory():
    return {
        'group': {
            'hosts': ['bender', 'fry'],
            'vars': {
                'example_var1': 'proxyeast',
                'example_var2': 'proxywest',
                'ansible_ssh_pass': 'alta3',
                'ansible_python_interpreter': '/usr/bin/python3'
            }
        },
        '_meta': {
            'hostvars': {
                'bender': {
                    'ansible_user': 'bender',
                    'ansible_host': '10.10.2.3'
                },
                'fry': {
                    'ansible_user': 'fry',
                    'ansible_host': '10.10.2.4'
                }
            }
        }
    }

# Empty inventory for testing.
def empty_inventory():
    return {'_meta': {'hostvars': {}}}
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('--list', action = 'store_true')
    parser.add_argument('--host', action = 'store')
    args = parser.parse_args()
    main()
```

24. Save and exit.

25. Make the script, `~/ans/inv/hybridinv/inventory.py` executable.

```
student@bchd:~$ chmod u+x ~/ans/inv/hybridinv/inventory.py
```

26. In the above script, we hard coded some json within `example_inventory()`. In production, we would want our script to make an API call, or parse some file. If you're new to Python, start studying the latest Python3.x release. If you want to jump-start your learning of making API requests with Python3, study the `requests` library. To access the hard-coded JSON inventory within our script, just run the script with the `--list` parameter.

```
student@bchd:~$ python3 ~/ans/inv/hybridinv/inventory.py --list
```

27. Now try using your script to generate an inventory for an Ansible Ad-Hoc command. It won't matter that our ansible inventory is described as a differently in `~/.ansible.cfg` as CLI options always override other settings. Notice that the `--list` parameter is no longer needed. That is because Ansible automatically calls all script with this parameter.

```
student@bchd:~$ ansible all -m ping -i ~/ans/inv/hybridinv/inventory.py
```

28. Try it again, this time, target just `fry`. In running this second command, understand that inventory.py is providing the potential candidates, much like the static `hosts` file provides. However, we still must select the individual host(s) from that candidate list.

```
student@bchd:~$ ansible fry -m ping -i ~/ans/inv/hybridinv/inventory.py
```

29. Let's build a hybrid inventory by adding a static file to our directory `~/ans/inv/hybridinv/`

```
student@bchd:~$ vim ~/ans/inv/hybridinv/hostszoidfarns.yaml
```

30. Create the following YAML file.

```
---
all:
        hosts:
                zoidberg:
                        ansible_host: 10.10.2.5
                        ansible_user: zoidberg
                        ansible_python_interpreter: /usr/bin/python
                        ansible_ssh_pass: alta3
                farnsworth:
                        ansible_host: 10.10.2.6
                        ansible_user: farnsworth
                        ansible_python_interpreter: /usr/bin/python
                        ansible_ssh_pass: alta3
```

31. Save and exit.

32. Run the Ansible Ad-Hoc command again, only this time, specify the *directory* `~/ans/inv/hybridinv/` the expected behavior is that ansible will read all of the files, and execute the script(s), found within the directory. Any encountered scripts will be run with the `--list` parameter.

```
student@bchd:~$ ansible all -m ping -i ~/ans/inv/hybridinv/
```

33. Some files extensions can be ignored. This behavior is configurable within `~/.ansible.cfg`.

34. Great job! Run the tear down script to stay cleaned up.

```
student@bchd:~$ bash max-teardown.sh
```

45. Ansible and AWS

Lab Objective

AWS is the most popular cloud service, so it follows that automating services provided by AWS is "in demand". If you want to perform this lab, you'll need to have an AWS account, which includes getting out a credit card. Typically, AWS will offer all kinds of limited free access for first time users, but consider yourself warned! **If you aren't certain you want to try something that involves a credit card, than don't perform this lab! Just reading through this lab is fine.**

Lab Procedure

1. To start, you'll need an AWS account. If you don't, go to http://aws.amazon.com/
2. Sign Up & create a new account (they'll give you the option for 1 year trial or similar). You'll need to fork over a credit card. Typically, AWS will offer all kinds of limited free access for first time users. **Again, if you aren't certain you want to try something that involves a credit card, than don't perform this lab!**
3. After logging into your AWS account, **click** the account menu in the upper-right (has your name on it)
4. **Click** the sub-menu: Security Credentials
5. In the center of the screen click on **Access keys (access key ID and secret access key)
6. Once the window is expanded by clicking on it, you can click on the blue button that says **Create New Access Key**
7. Downloading the key is not necessary, but you will need their values. Copy them to the clipboard for now.
8. On your Ansible controller, we can place those keys within a file that you won't accidently commit to a git repo. Your home directory is fine.

   ```
   vim ~/cred.zon
   ```

9. Fill out the credential file as follows:

   ```
   ---
   ec2_access_key: "--REMOVED--"     # place your Access Key ID here
   ec2_secret_key: "--REMOVED--"     # place your Secret Key here
   ```

10. Save and exit with `:wq`
11. **IMPORTANT** Without wasting any time, we want to encrypt the `~/cred.zon` file with `ansible-vault`

    ```
    ansible-vault encrypt cred.zon --ask-vault-pass
    ```

12. Make up a password that you won't forget. You'll need it in a moment.
13. It is worth noting that if you do use a `vars_files` credential file to store your AWS credentials, you'll need to keep reference your keys in every module. An alternative would be setting the keys as environmental variables. For more information on this see the guide on Amazon Web Services https://docs.ansible.com/ansible/latest/scenario_guides/guide_aws.html
14. Let's try to create an Amazon S3 bucket. The steps for manually creating a bucket are here https://docs.aws.amazon.com/AmazonS3/latest/user-guide/create-bucket.html If you have never made an S3 bucket, you might want to review those steps now. In short, S3 buckets are for object storage.
15. Ansible has a module we can use to do this work `s3_bucket` https://docs.ansible.com/ansible/latest/modules/s3_bucket_module.html
16. Looks like in order to use it, we need the `boto3` client installed. Install that now.

    ```
    python3 -m pip install boto3
    ```

17. Great! All that is left is to write a playbook.

    ```
    vim ~/ans/aws-s3-playbook.yml
    ```

18. Create the following play.

```
---
- name: Create an s3 bucket on AWS with Ansible
  hosts: localhost
  connection: local

  vars:
      hereorthere: "present"

  vars_files:
          - ~/cred.zon
  tasks:
          - name:
            s3_bucket:
              name: ducktoaster77    # naming rules https://docs.aws.amazon.com/AmazonS3/latest/dev/
BucketRestrictions.html#bucketnamingrules
              state: "{{ hereorthere }}"
              aws_access_key: "{{ ec2_access_key }}"
              aws_secret_key: "{{ ec2_secret_key }}"
```

19. Save and exit with `:wq`
20. Try running the playbook.

```
ansible-playbook ~/ans/aws-s3-playbook.yml --ask-vault-pass
```

21. When prompted, type the password you invented a few steps ago to encrypt your `~/cred.zon` file.

> A common failure for creating S3 buckets, is that the name must be unique. If `ducktoaster77` has already been taken, you might need to create `koolaidchistmas62`

22. Now that your bucket has been created, you should be able to view it on your AWS account via https://console.aws.amazon.com/s3/home
23. Remove your bucket with the following command:

```
ansible-playbook aws-playbook.yml --ask-vault-pass -e "hereorthere=absent"
```

24. When prompted, type the password you invented a few steps ago to encrypt your `~/cred.zon` file.
25. Confirm in your AWS account that the S3 bucket has been removed.

46. Playbook Vars Prompts

Lab Objective

The objective of this lab is to learn how to prompt for input via Ansible. When prompting for input we use the **vars_prompt** keyword within the play section of the playbook. This section is totally optional and should be used when you need to prompt the user for dynamic input, as pausing the playbook and waiting for entry is **not** the most automation friendly technique.

Procedure

1. Change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. This lab uses the farnsworth machine from the **planetexpress** team. If they are already built, skip ahead now. Otherwise, run the teardown script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the teardown script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. No need to copy any keys over, as we are only using the farnsworth host. We'll plan on using a password to connect to this host.

7. We want to stay organized, so create a directory structure, `~/ans/inv/dev/`.

```
student@bchd:~$ mkdir -p ~/ans/inv/dev/
```

8. Edit your inventory file. We'll describe this group within `~/ans/inv/dev/hosts`

```
student@bchd:~$ vim ~/ans/inv/dev/hosts
```

9. Ensure that you have the following entry within your inventory (you can have other group entries, just ensure you have 'at least' the following). Notice, that the host `farnsworth` does not have the variable `ansible_ssh_pass` defined.

```
[professors]
farnsworth  ansible_host=10.10.2.6 ansible_user=farnsworth
```

10. Save and exit.

11. Looks like farnsworth wants to use a password to connect. To do that, we should install `sshpass`. Do that now.

```
student@bchd:~$ sudo apt install sshpass
```

12. Ensure your copy of `ansible.cfg` is accurate.

```
student@bchd:~$ vim ~/.ansible.cfg
```

13. Your file should reflect the following values:

```
[defaults]
# default location of inventory
# this can be a file or a directory
inventory = /home/student/ans/inv/dev/
# prevents playbook from hanging on new connections
host_key_checking = no
```

14. Save and exit with `:wq`

15. Use vim to create a file called, `prompt-playbook01.yml`.

```
student@bchd:~$ vim ~/ans/prompt-playbook01.yml
```

16. The following playbook will prompt the user to define the variable **yourpassword**. Please note that we don't need to be asking for a password. This could be a server to connect to, or perhaps a value to end up within a config file. Add the following into your playbook:

```
---
- name: How to prompt for variables
  hosts: localhost
  gather_facts: no

  vars_prompt:
    - name: "yourpassword"  # Variable name to define
      prompt: "What is your password?" # Question to ask
      ## do not reveal the password (optional)
      private: yes
      ## run this encryption method (optional)
      encrypt: "sha512_crypt"
      ## type the password twice (optional)
      confirm: yes
      ## this is the size of the RAND you'd like to have generated (optional)
      salt_size: 7
      # salt: slappysquirrel88 # provide your own RAND (optional)
      ## default to this password (optional)
      default: "qwerty"

  tasks:
    - name: Print out the password
      debug:
        msg: "{{ yourpassword }}"
```

17. Save and exit with `:wq`

18. Run the playbook and examine the output.

```
student@bchd:~$ ansible-playbook ~/ans/prompt-playbook01.yml
```

19. Notice that the password is totally encrypted. This is how it will appear in logs as well. If we allow Ansible to define a new RAND each time (do not use a static RAND), then it is highly doubtful that this encrypted value will be 'cracked.'

20. Comment out (put a # in front of) some of the optional parameters within **vars_prompt** and try running it again. Commenting out **default: "qwerty"** is a line that would be fine to comment out. After you have commented out a line or two, run the code again to see the effects.

21. Continue to perform the above step until you're sure you understand what each of the vars_prompt parameters do.

22. Create a new playbook.

```
student@bchd:~$ vim ~/ans/prompt-playbook02.yml
```

23. Create the following playbook:

```
---
- name: How to prompt for variables
  hosts: professors     # connect to the professors group
  gather_facts: no

  vars_prompt:
    - name: "ansible_ssh_pass"  # password to connect to the hosts in the professors group
      prompt: "What is your ssh password?" # Question to ask
      ## do not reveal the password (optional)
      private: yes

  tasks:
    - name: connect to the professors and install figlet
      yum:
              name: figlet
              state: present
      become: yes
```

24. Save and exit with `:wq`
25. Run your new playbook.

```
student@bchd:~$ ansible-playbook ~/ans/prompt-playbook02.yml
```

26. When prompted, type `alta3`
27. The playbook should install `figlet` on the remote system. If you'd like to check, SSH to `farnsworth@10.10.2.6` and type `figlet` followed by `did it work?`. Use **CTRL** + **c** to exit the app.
28. Great job! That's it for this lab.

47. Ansible Vault

Lab Objective

The objective of this lab is to learn to encode text files with ansible-vault. Vault installs alongside Ansible, so you already have it on your system. Its job is to both encrypt text files (password files, variable files, playbooks), as well as decrypt. With ansible vault, you can permanently decrypt files, or just decrypt them for run-time. The default cipher is AES (which is shared-secret based).

Procedure

1. Open a new terminal, then change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. Great! Now create a directory to store variables in.

```
student@bchd:~$ mkdir -p ~/ans/vars
```

3. Move into the `~/ans` directory.

```
student@bchd:~$ cd ~/ans
```

4. Open a new text file that we can encrypt with ansible-vault.

```
student@bchd:~/ans$ vim ~/ans/vars/mypasswords.yml
```

5. Create the following:

```
---
datacenter: pennsylvania
hostname: tardis
password: DoctorWho?TheGoodDoctor
secondary: tardis2
dlspeed: Download_60M
upspeed: Upload_20M
password2: QuestForGlory1989EGA
password3: LauraBowDaggerOfAmonRa1992
```

6. Save and exit. Be sure the file is saved as, **mypasswords.yml**

7. Now let's encrypt that file with ansible-vault.

```
student@bchd:~/ans$ ansible-vault encrypt vars/mypasswords.yml
```

8. When prompted, type the password **qwerty**

9. When prompted, re-type the password **qwerty**

10. Great. Now print out the encrypted file.

```
student@bchd:~/ans$ cat vars/mypasswords.yml
```

11. If it worked, you should 'just' see an encrypted file. This file could be considered quite a bit safer than clear text.

12. To edit an encrypted file in place, use the ansible-vault edit command. This command will decrypt the file to a temporary file and allow you to edit the file, saving it back when done and removing the temporary file:

```
student@bchd:~/ans$ ansible-vault edit vars/mypasswords.yml
```

13. When prompted, type the password **qwerty**

14. You don't need to make any changes, just appreciate that the file is decrypted for editing purposes only. Exit the file.

15. To ensure that is the case, cat the file again. It should still be encrypted.

```
student@bchd:~/ans$ cat vars/mypasswords.yml
```

16. Now let's decrypt that file with ansible-vault.

```
student@bchd:~/ans$ ansible-vault decrypt vars/mypasswords.yml
```

17. You should be able to view and understand the file, as it has now been decrypted.

```
student@bchd:~/ans$ cat vars/mypasswords.yml
```

18. Open a new text file to create a playbook that we might encrypt with ansible-vault.

```
student@bchd:~/ans$ vim playbook-vault01.yml
```

19. Create the following:

```
---
- hosts: localhost

  tasks:
    - name: Print to the screen
      debug:
        msg: "This is your secret playbook. You and you alone must retain control of it. Your mission, Jim, should you
choose to accept it. Is to first encrypt, then run this playbook (while it is encrypted). This message will self-destruct
in 5 seconds."
```

20. Save and exit.

21. Try running your **unencrypted** playbook.

```
student@bchd:~/ans$ ansible-playbook playbook-vault01.yml
```

22. Now, **encrypt** the playbook file with ansible-vault.

```
student@bchd:~/ans$ ansible-vault encrypt playbook-vault01.yml
```

23. When prompted, type the password **qwerty**

24. When prompted, re-type the password **qwerty**

25. Print out the encrypted file.

```
student@bchd:~/ans$ cat playbook-vault01.yml
```

26. Once again, if it worked, you should 'just' see an encrypted file. This file could be considered quite a bit safer than clear text.

27. Next run the encrypted playbook.

```
student@bchd:~/ans$ ansible-playbook --vault-id @prompt playbook-vault01.yml
```

28. When prompted, enter **qwerty** and the playbook should run.

29. Suppose we have a variable file that we're interested in keeping secret and safe. This could be credentials or sensitive IP addresses. Let's start by creating an encrypted vars file.

```
student@bchd:~/ans$ vim ~/ans/vars/oscreds.yml
```

30. Create the following, which mimics the values required to authenticate to OpenStack.

```
---
imscloud:
  auth:
    auth_url: http://192.168.122.10:35357/
    project_name: demo
    username: demo
    password: 0penstack
  region_name: RegionOne
```

31. Save and exit.

32. Encrypt this new credential file with ansible vault.

```
student@bchd:~/ans$ ansible-vault encrypt vars/oscreds.yml
```

33. When prompted, enter **qwerty**

34. Now create a playbook.

```
student@bchd:~/ans$ vim playbook-vault02.yml
```

35. Create the following:

```
---
- name: Extra vars vault
  hosts: localhost

  # reference the encrypted variable file
  vars_files:
    - vars/oscreds.yml

  tasks:
    - name: print out encrypted vars
      debug:
        msg: "{{ imscloud }}"
```

36. Save and exit.

37. Now run the playbook, and pass a command to decrypt the vars_file.

```
student@bchd:~/ans$ ansible-playbook playbook-vault02.yml -e @vars/oscreds.yml --ask-vault-pass
```

38. The playbook should run, and print out the credentials found within the password file. Obviously, you wouldn't want to print out your credentials! However, this proves that your variables have been successfully decrypted for the purposes of running your playbook. Double check that the file itself is still encrypted.

```
student@bchd:~/ans$ cat ~/ans/vars/oscreds.yml
```

39. Suppose we wanted to create a playbook with encrypted vars inside of it. Run the following command to encrypt the string `pAssw0rD` and map it to the key `mypass`. *Note, if you were to use a $ in your password, be sure to ESCAPE it with a backslash, or it will be interpreted by the shell (passing $$ at the shell will echo a PID). Therefore, the proper way to pass a password containing $ symbols is* `pA\$\$w0rD`

```
student@bchd:~/ans$ ansible-vault encrypt_string "pAssw0rD" --name "mypass"
```

40. Copy everything from `mypass` to the end of the encryption code.

41. Create a playbook.

```
student@bchd:~/ans$ vim ~/ans/playbook-vault03.yml
```

42. Create the following playbook. You'll need to edit the following, and replace the value under `vars:` with the value you copied out of the CLI:

```
---
- name: encrypt a string
  hosts: localhost

  # edit the following section with your string
  vars:
    mypass: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          30373930383263363356...
          6236262656665383373...
          3761633065666139373...
          6232623765613133640...

  tasks:
    - name: print an encrypted string
      debug:
         msg: "{{ mypass  }}"
```

43. Save and exit.

44. Run your playbook with the following command. When prompted, enter **qwerty**

```
student@bchd:~/ans$ ansible-playbook playbook-vault03.yml --ask-vault-pass
```

45. The playbook should display `pAssw0rD`

46. Great job!

48. Roles and Molecule

Lab Objective

Welcome to the new hotness in role creation and testing. Roles are highly organized playbooks that work as a kind of building block for creating more advanced playbooks. Molecule is a brand new tool that ships with Ansible v2.7 (and beyond). The purpose of this tool is testing roles (as in, if they are properly constructed) which is accomplished via a series of Python scripts. In this lab we will examine Ansible roles as well as using the Molecule tool.

Start by reviewing the documentation: https://molecule.readthedocs.io/en/latest/index.html

For testing purposes, review usage of assert: https://docs.ansible.com/ansible/latest/modules/assert_module.html

Procedure

1. Run `groups`. If it says something about `docker`, you can skip the next few steps.

   ```
   student@bchd:~$ groups
   ```

2. For this lab to run we'll need to give Docker permission to run as sudo. Issue the following command:

   ```
   student@bchd:~$ sudo usermod -aG docker student
   ```

3. Create a new shell with all the current settings and users carried over.

   ```
   student@bchd:~$ su - $USER
   ```

4. Enter **alta3**

5. This command should now show a Docker group.

   ```
   student@bchd:~$ groups
   ```

6. You should be able to run this Docker command successfully (i.e. *without* sudo). **NOTE: If this fails, run the command again (and again) until it works.**

   ```
   student@bchd:~$ docker run hello-world
   ```

7. Run an apt update.

   ```
   student@bchd:~$ sudo apt-get update
   ```

8. Install `libssl-dev`

   ```
   student@bchd:~$ sudo apt install libssl-dev
   ```

9. Install setuptools.

   ```
   student@bchd:~$ python3 -m pip install --upgrade --user setuptools
   ```

10. We need to install Molecule, which is available via pip from the Pypi repo. We also throw the instruction to install the docker driver.

    ```
    student@bchd:~$ python3 -m pip install --user "molecule[docker,lint]"
    ```

11. Great! See what version you're running. You just installed it, so it should be the latest-ish version of the tool.

    ```
    student@bchd:~$ molecule --version
    ```

12. Before we start integrating Molecule into an existing role, let's take a look at using Molecule itself to init a new role within itself:

    ```
    student@bchd:~$ molecule init role alta3.example --driver-name docker
    ```

13. If the command worked, you'll see that a new role was initialized within `/home/student/alta3.example successfully`. This command uses `ansible-galaxy` behind the scenes to generate a new Ansible role, then it injects a Molecule directory in the role and sets it up to run

builds and test runs in a Docker environment. Inside the Molecule directory is a default directory indicating the default test scenario. It contains the following:

```
student@bchd:~$ cd ~/alta3.example && ls
```

14. This looks like the structure of a basic role only a Molecule directory is also present. Let's look inside.

```
student@bchd:~$ cd ~/alta3.example/molecule/default/ && ls
```

15. Below is a breakdown each of the files you are seeing.

    - **converge.yml:** Contains instructions for how to handle the `converge` command.
    - **verify.yml:** Contains the tests to run against the virtual environment for testing your role.
    - **molecule.yml:** Tells Molecule everything it needs to know about your testing: what OS to use, how to lint your role, how to test your role, etc. We'll cover a little more on this later.

16. We can customize just about any of these files, but they will work in their current state.

17. Move down two directories.

```
student@bchd:~/alta3.example/molecule/default$ cd ../../
```

18. Verify that Molecule has no instances up and running.

```
student@bchd:~/alta3.example$ molecule list
```

19. Tell Molecule to create an instance to test with.

```
student@bchd:~/alta3.example$ molecule create
```

20. Verify that Molecule has an instance up and running.

```
student@bchd:~/alta3.example$ molecule list
```

21. Our role is currently empty. Let's try adding a simiple task.

```
student@bchd:~/alta3.example$ vim tasks/main.yml
```

22. Add the following into `main.yml`

```
- name: Molecule Hello World!
  debug:
    msg: Hello, World!
```

23. Save and exit.

24. We can now tell Molecule to test our role against our instance.

```
student@bchd:~/alta3.example$ molecule converge
```

25. If we want to see the effects it had on the instance we can inspect changes with:

```
student@bchd:~/alta3.example$ molecule login
```

26. To be clear, you're in a virtual environment (like a pretend server). You can run changes against it and then inspect them... pretty cool! All right, escape the instance:

```
root@instance:/# exit
```

27. Now destroy the instance.

```
student@bchd:~/alta3.example$ molecule destroy
```

28. The instance should no longer exist.

```
student@bchd:~/alta3.example$ molecule list
```

29. Edit the tasks again, let's make ansible install something.

```
student@bchd:~/alta3.example$ vim tasks/main.yml
```

30. Add the following into `main.yml`

```
# tasks file for alta3.example
- name: Molecule Hello World!
  debug:
    msg: Hello, World!

# install cowsay
- name: install cowsay
  package:
    name: cowsay
```

31. Save and exit.

32. Let's build a test as well.

```
student@bchd:~/alta3.example$ vim molecule/default/verify.yml
```

33. The molecule testing engine used to be pytest, but has been replaced with Ansible! Keeping everything in the Ansible language makes it a bit easier to grok. When you build tests, it's likely you'll make heavy use of the `assert` module. This will "assert that..." some condition tests true. Replace `verify.yml` with the following.

```
---
# This is an example playbook to execute Ansible tests.
#
- name: Verify
  hosts: all
  tasks:

  # Review how to use the assert module
  # https://docs.ansible.com/ansible/latest/modules/assert_module.html
  - name: Example assertion
        assert:
          that: true  # test "that" true is true (its a silly test, it will always "work")

  - name: Cowsay says hello
        command: cowsay "hello"   # attempt to use the program cowsay
        register: result      # this contains a run code that reveals if the command was successful

  # Review how to use the assert module
  # https://docs.ansible.com/ansible/latest/modules/assert_module.html
  - name: Perform an assertion test. This will fail if result.rc is not equal to 0
        assert:
          that: result.rc == 0
```

34. Save and exit.

35. Create a test environment.

```
student@bchd:~/alta3.example$ molecule create
```

36. Run your role against the environment with the `converge` command

```
student@bchd:~/alta3.example$ molecule converge
```

37. Try running our verification tests with the `verify` command.

```
student@bchd:~/alta3.example$ molecule verify
```

38. Run an idempotence test. This will ensure nothing changes when the role is run a second time against the environment

```
student@bchd:~/alta3.example$ molecule idempotence
```

39. Rip down the old environment.

```
student@bchd:~/alta3.example$ molecule destroy
```

40. Try running a `molecule test`. This cycles through an entire full lifecycle of possible tests, which includes linting (best practice), checking for available dependencies, checking syntax, running the role against an instance (create, prepare and converge), testing for idempotence, then cleaning up the test instance.

    ```
    student@bchd:~/alta3.example$ molecule test
    ```

41. Cool! You might want to try building out your own role and running a test against it. Some practical suggestions include bootstrapping a switch or perhaps spinning up a VM with a keypair the user passes as a variable.

42. That's it for this lab! Nothing really to backup here, so don't worry about running any Git commands.

49. Ansible Playbook Output Logging

Lab Objective

A common question is where to find Ansible output data. If you pass **-v** with your playbooks, as in, ***ansible-playbook myplaybook.yml -i hosts -v***, stderr and stdout will be shown. These values may be written out to a static file, the location of this file needs to be configured within the `ansible.cfg` file. The `ansible.cfg` will be searched for in the following order:

First ANSIBLE_CONFIG (location is taken from environmental variable, if set) **Second** ansible.cfg (in the current directory) **Third** ~/.ansible.cfg (in the home directory) **Fourth** /etc/ansible/ansible.cfg *Ansible will process the above list and use the first file found, all others are ignored*

Procedure

1. Ensure you have a place to work in. According to the **4** places that may host `ansible.cfg`, this 'local' directory would be the **third** in the search process.

```
student@bchd:~$ mkdir -p ~/ans/
```

2. Edit `~/.ansible.cfg`. It is important to precede the config file with a dot. This makes the file hidden. Ansible can't find the file in the home directory, unless it is hidden.

```
student@bchd:~$ vim ~/.ansible.cfg
```

3. Replace the contents of `~/.ansible.cfg` with the following. Notice that we have a new parameter, `log_path = /tmp/ansible.log`

```
[defaults]
log_path = /tmp/ansible.log
ask_sudo_pass = False
ask_pass = False
inventory = /home/student/hosts
host_key_checking = False
record_host_keys = /dev/null
```

4. Save and exit.

5. Create a simple playbook.

```
student@bchd:~$ vim ~/ans/playbook-logme.yml
```

6. Create the following playbook.

```
---
- name: Intro to looping
  hosts: localhost
  gather_facts: no

  tasks:
    - name: First debug
      debug:
        msg: "The first message in the playbook."
      no_log: true   # this setting will PREVENT the task from being logged

    - name: Second debug
      debug:
        msg: "The second message in the playbook."
```

7. Save and exit. **Notice that we placed a `no_log: true` setting on our first debug task. This task will no longer be logged. This is an important keyword for tasks that perform sensitive operations, like passing credentials.**

8. Now everything Ansible outputs will be placed in `/tmp/ansible.log`. Ensure that is the case by checking how ansible is configured. The settings in YELLOW are variables that have been altered from their defaults. Scroll down and you should see an entry regarding logging.

```
student@bchd:~$ ansible-config dump
```

9. To exit press `q`

10. Try running any playbook you'd like. Even if it doesn't work, we should capture something in */tmp/ansible.log*

```
student@bchd:~$ ansible-playbook ~/ans/playbook-logme.yml
```

11. Let's look at our log file and see what was produced. Remember, we have `no_log: true` on our first task. Therefore, we don't expect the first task to be logged.

```
student@bchd:~$ cat /tmp/ansible.log
```

12. Make the following directory if it does not already exist.

```
student@bchd:~$ sudo mkdir /etc/ansible/
```

13. Try moving your ansible.cfg file to a different location. Even if you move the config file, it should still direct ansible to place logs in the **/tmp/ansible.log**. Do not make the file hidden when you move it. That condition is only valid if it is inside the user home directory.

```
student@bchd:~$ sudo mv ~/.ansible.cfg /etc/ansible/ansible.cfg
```

14. Check the current config. Ansible should still be logging to **/tmp/ansible.log**, but the instruction to do so should no be coming from **/etc/ansible.cfg**

```
student@bchd:~$ ansible-config dump
```

15. Run a playbook again.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-logme.yml
```

16. Look again at what was produced. You should (still) see the results of your latest playbook run within the log.

```
student@bchd:~$ cat /tmp/ansible.log
```

17. **IMPORTANT** to prevent problems later, remove the current config file.

```
student@bchd:~$ sudo rm /etc/ansible/ansible.cfg
```

18. Re-create `~/.ansible.cfg` without logging activated!

```
student@bchd:~$ vim ~/.ansible.cfg
```

19. Logging should always be **off** unless you need it on (the security people will thank you for this). No logging is the default setting.

```
[defaults]
ask_sudo_pass = False
ask_pass = False
inventory = /home/student/hosts
host_key_checking = False
```

20. Save and exit.

21. Ensure logging is OFF.

```
student@bchd:~$ ansible-config dump
```

22. When you are sure logging is off (it should be GREEN **NOT** YELLOW), press `q` to exit.

23. Great job! This lab is complete.

50. Ansible Module - script

Lab Objective

The objective of this lab is to learn to use Ansible to call several different flavors of script, including Python and Perl, but any flavor of script might be drawn on. This really isn't best practice, in fact, it would be a **much** better idea to *just* write an ansible module that did this. Calling scripts is typically a get-it-done approach, whereas an Ansible module should be highly-reviewed, stabilized, and ensured to be idempotent before it became a module and was able to be used within a playbook.

Read about the script module here:
https://docs.ansible.com/ansible/latest/modules/script_module.html

Procedure

1. To begin, we should create a place to store the scripts we want to have executed on our remote hosts.

```
student@bchd:~$ mkdir -p ~/ans/scripts/
```

2. Now create a Python script, `~/ans/scripts/nasaLookup.py`

```
student@bchd:~$ vim ~/ans/scripts/nasaLookup.py
```

3. Create the following Python script.

```
#!/usr/bin/python3

import requests

astros = requests.get("http://api.open-notify.org/astros.json")
print(astros.json()["people"])
```

4. Save and exit.

5. Before invoking the script, you might want to visit http://api.open-notify.org/astros.json in a browser and see the JSON returned by an HTTP GET request.

6. Before running the script, use pip to ensure `requests` is installed. It is a fairly commonly python library used to make API calls with the HTTP protocol, so it is likely already installed. The tool pip installs code freely available from https://pypi.org

```
student@bchd:~$ python3 -m pip install requests
```

7. Try running the Python script.

```
student@bchd:~$ python3 ~/ans/scripts/nasaLookup.py
```

8. Create a playbook to utilize our Python script.

```
student@bchd:~$ vim ~/ans/scripts/playbook-script01.yml
```

9. Create the following playbook that invokes the python script.

```
---
- name: Playbook that runs a Python script
  hosts: localhost

  tasks:
  - name: Copy script to target hosts and execute it
    script: ~/ans/scripts/nasaLookup.py
    args:
      executable: /usr/bin/python3
    register: results    # save the JSON sent back by our script

  - name: Print out the 'results'
    debug:
      var: results
      # notice there is a stdout and stdout_lines
```

10. Save and exit.

11. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/scripts/playbook-script01.yml
```

12. We could use the `uri` module to make Ansible send an HTTP GET message, and mimic what a script does, but that's not the point. The goal right now is to understand that we can make Ansible run scripts. Let's try making ansible run a Perl script.

```
student@bchd:~$ vim ~/ans/scripts/starWars.pl
```

13. Create the following Perl script.

```
#!/usr/bin/perl

# Modules used
use strict;
use warnings;

# Print functions
print("In an Ansible Galaxy far, far away\n");
print("It is a period of civil war.\nRebel spaceships, striking\n");
print("from a hidden base, have won their first victory against\n");
print("the evil Galactic Empire.\n")
```

14. Save and exit.

15. Try running the Perl script.

```
student@bchd:~$ perl ~/ans/scripts/starWars.pl
```

16. Create a playbook to utilize our Perl script.

```
student@bchd:~$ vim ~/ans/scripts/playbook-script02.yml
```

17. Create the following playbook that invokes the perl script.

```
---
- name: Playbook that runs a Perl script
  hosts: localhost

  tasks:
  - name: Copy script to target hosts and execute it
    script: ~/ans/scripts/starWars.pl
    args:
      executable: /usr/bin/perl
    register: results    # save the JSON sent back by our script

  - name: Print out the 'results'
    debug:
      var: results
      # notice there is a stdout and stdout_lines
```

18. Save and exit.

19. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/scripts/playbook-script02.yml
```

20. **CHALLENGE (OPTIONAL)-** Write a playbook that performs a lookup on http://api.open-notify.org/astros.json without invoking the script module. If you get stuck, one possible solution is below. **Hint:** One possible solution uses the `uri` module to send an HTTP GET to the API, and register the results. Check out the URI module here: https://docs.ansible.com/ansible/latest/modules/uri_module.html

21. **CHALLENGE SOLUTION** - Create the challenge solution, `playbook-script01-rewrite.yml`

```
student@bchd:~$ vim ~/ans/scripts/playbook-script01-rewrite.yml
```

22. **CHALLENGE SOLUTION** - Create the following challenge script solution.

```
---
- name: Playbook that sends HTTP GET to target URI
  hosts: localhost

  vars:
    urltolookup: "http://api.open-notify.org/astros.json"

  tasks:
  - name: Lookup api with uri module
    uri:
      url: "{{ urltolookup }}"
      method: GET
    register: results    # save the JSON sent back by our script

  - name: Print out the 'results'
    debug:
      var: results
```

23. Save and exit.

24. Try running your challenge solution.

```
student@bchd:~$ ansible-playbook ~/ans/scripts/playbook-script01-rewrite.yml
```

25. Great job! That is it for this lab.

51. Ansible, Python Methods and Jinja Filters

Lab Objective

The objective of this lab is to combine our Ansible and Python skillset. Python contains a templating language called "Jinja." Jinja is now on its second version, so we call it Jinja2.

- **Jinaj2 Homepage -** http://jinja.pocoo.org/docs/2.10/

At the time of this writing the official documentation on Jinja2 filters is a bit lacking in examples, but here is what's available:

- **Ansible Documentation on Jinja Filters -** https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html
- **Jinja2 Built-in Filter Documentation -** http://jinja.pocoo.org/docs/2.10/templates/#list-of-builtin-filters

Think of a filter as an abstracted call to some Python code. The thing is, we can call Python code (methods) directly from Ansible. As you'll see in this example, sometimes it makes sense to use a filter, sometimes a Python method, and sometimes both!

Procedure

1. Move into (or create) a new directory to work in, `/home/student/ans/`

   ```
   student@beachhead:~$ mkdir ~/ans/
   ```

2. Move into the `~/ans` directory.

   ```
   student@beachhead:~$ cd ~/ans/
   ```

3. Now create a script called `pyjinja/pythonandjinja01.yml`

   ```
   student@beachhead:~/ans$ vim ~/ans/pythonandjinja01.yml
   ```

4. Add the following into your new playbook:

```
---
- name: A Play to play with Python
  hosts: localhost

  vars:
    test_list:
      - '192.168.2.1'
      - '10.10.0.2'
      - 'host.example.org'
      - '192.168.70.0/24'
      - 'fe80::100/10'
      - 'True'
      - 'Witcher 3: The Wild Hunt'
      - 'Earl Grey Tea, hot'
    mytest_string: "NCC1701 USS Enterprise"
    myaddresses: ['192.168.2.101', '10.0.3.50', '171.172.32.1']

  tasks:
  # Use the Jinja ipaddr filter to reveal JUST IPs from a mixed list.
  - name: Just want the IPs
    debug:
      msg: "{{ test_list | ipaddr }}"

  # Use the Jinja random_mac filter to finish a MAC address.
  - name: Spawn rando mac
    debug:
      msg: "{{'52:54:00' | random_mac }}"

  # When using the var statement, Ansible would prefer you did NOT use mustache brackets
  - name: Some don't need mustache brackets
    debug:
      var: "'52:54:11'|random_mac"
  # FYI - The no mustache brackets rule also applies to when statements
  # when: never use {{    }} in when statements

  # Here is an example of using a Python method to parse our data
  # This method returns a bool (true/false)
  - name: Test if our string ends in prise
    debug:
      msg: "{{ mytest_string.endswith('prise') }}"

  # Building a conditional around our task-
  # this will only print if our string ends in prise
  - name: This only prints when this tests true using python
    debug:
      msg: "Ahh, but is it the Enterprise subclass A, B, C, or D?"
    when: mytest_string.endswith('prise')

  # Sometimes a Jinja filter seems like more work to learn than just using a Python method
  - name: Jinja2 way of capitalizing a string
    debug:
      msg: "{{ mytest_string | capitalize }}"

  # Same example, only we use a Python method, not a Jinja2 filter
  # notice how Python methods use dot notation and Jinja uses pipes
  - name: Python way of capitalizing a string
    debug:
      msg: "{{ mytest_string.capitalize() }}"

  # Sometimes we need to use Python AND Jinja.
```

```
    # ipaddr returns all addresses that match the scope.
    # The square brackets selects the only item in the list.
    # We split the IP address into a list containing 4 items (each item is an octet).
    # We then select the 3rd item (4th in the list) and display it to the screen
    - name: Pop the number 50 on the screen
      debug:
        msg: "{{ (myaddresses|ipaddr('10.0.0.0/8'))[0].split('.')[3] }}"
```

5. The filter ipaddr requires the package `netaddr` to run. Install that now with pip.

```
student@beachhead:~/ans$ python3 -m pip install netaddr
```

6. Read the comments on the code, then save and exit.

7. Time to run `pythonandjinja01.yml`

```
student@beachhead:~/ans$ ansible-playbook pythonandjinja01.yml
```

8. Examine the results and continue to run the code as necessary.

9. **CODE CUSTOMIZATION 01 (OPTIONAL)** - Select a filter from one of the two locations (possibly both) and see if you can use it in the playbook; write a new task if need be.

    - https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html
    - http://jinja.pocoo.org/docs/2.10/templates/#list-of-builtin-filters

10. **CODE CUSTOMIZATION 02 (OPTIONAL)** - See if you can find a Python method to use within the playbook. You can write a new task if you'd like.

11. If you're tracking your code in GitHub, issue the following commands:

    - `cd ~/ans/`
    - `git status`
    - `git add ~/ans/*`
    - `git commit -m "python methods and jinja2"`
    - `git push https://github.com/your-account-name/ansible-dev master`
    - Type in username & password

52. Ansible for Brocade

Lab Objective

*This lab is a **case study**, and is designed to reflect acquired knowledge. The lab environments do not include targets for this lab, or access to the equipment may lay outside the current environment. Reading through the lab can still offer valuable information on using Ansible for Brocade.*

The objective of this lab to learn how to automate the Brocade

There is a current Brocade collection, however, at the time of writing, this author has been **unable** to make it execute. https://galaxy.ansible.com/brocade

At this time, it is recommended you still clone the GitHub repository and map the modules and utilities via ansible.cfg: https://github.com/brocade/ansible

In addition to the modules and utilities, you will also need to install paramiko and xml2dict via pip. https://pypi.org/project/XML2Dict/ https://pypi.org/project/paramiko/

Procedure

1. Read through the above comments. When you're ready, start in your home folder.

```
student@bchd:~$ cd
```

2. We need to ensure paramiko is installed. Use pip to do that now.

```
student@bchd:~$ python3 -m pip install paramiko
```

3. We need to ensure xml2dict is installed. Use pip to do that now.

```
student@bchd:~$ python3 -m pip install xml2dict
```

4. At the time of writing, the collection with this product appears to throw an error regarding mappings to the included utilities. The work around is to manually clone the repository containing the code packaged with the collection.

```
student@bchd:~$ git clone https://github.com/brocade/ansible ~/ansible-brocade/
```

5. Create the `~/ans/` and `~/ans/vars/` folders in the correct location for your system.

```
student@bchd:~$ mkdir -p ~/ans/vars/
```

6. Create an ansible.cfg file with (at least) the following settings. This will point ansible to the location of the new modules we just downloaded. *Note: This is additional library and utility locations, and will not nullify the other default search locations for modules and utilities that may be installed with Ansible. Additional library or utility locations may be added with the colon used as a separator.*

```
[defaults]
# location of our new modules
library = ~/ansible-brocade/library
# location of utilities required to use new modules
module_utils    = ~/ansible-brocade/utils
```

7. Save and exit with `:wq`

8. Create a place save an inventory.

```
student@bchd:~$ mkdir -p ~/ans/inv/prod/
```

9. Create an inventory file:

```
student@bchd:~$ vim ~/ans/inv/prod/hosts
```

10. The following is an example of an inventory with 4 brocade devices.

```
[brocadeswitches]
ds6620b001 fos_ip_addr=10.126.70.16
ds6620b002 fos_ip_addr=10.126.70.26
ds6620b003 fos_ip_addr=10.126.70.36
ds6620b004 fos_ip_addr=10.126.70.46



[brocadeswitches:vars]
ansible_connection=local
fos_user_name=admin
fos_password=Passw0rd
fos_https=False
```

11. Save and exit with `:wq`

12. Create a playbook to communicate with the Brocade switches.

```
student@bchd:~$ vim ~/ans/playbook-brocade01.yml
```

13. Create the following solution that will gather facts on brocade devices:

```
---
- name: Gather facts on Brocade inventory (DS-6620B)
  hosts: brocadeswitches
  gather_facts: false
  connection: local

  vars:
          credential:
                  fos_ip_addr: "{{ fos_ip_addr }}"
                  fos_user_name: "{{ fos_user_name }}"
                  fos_password: "{{ fos_password }}"
                  https: "{{ fos_https }}"

  tasks:

        - name: Gather facts about the Brocade DS-6620B Device
          brocade_facts:
                  credential: "{{ credential }}"
                  vfid: -1
                  timeout: 300
                  gather_subset:
                          - brocade_chassis_chassis
          register: results

        - name: Show the collected facts
          debug:
                  var: results
```

14. Save and exit with `:wq`

15. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-brocade01.yml -i ~/ans/inv/prod/
```

16. Great! Now let's try making a playbook that provokes a change to the switches. We'll try setting a banner, and the `user_friendly_name`. Given we're going to make a change, we'll also confirm that change.

```
student@bchd:~$ vim ~/ans/playbook-brocade02.yml
```

17. Create the following solution.

```
---
- name: Gather facts on Brocade inventory (DS-6620B)
  hosts: brocadeswitches
  gather_facts: false
  connection: local

  vars:
        credential:
                fos_ip_addr: "{{ fos_ip_addr }}"
                fos_user_name: "{{ fos_user_name }}"
                fos_password: "{{ fos_password }}"
                https: "{{ fos_https }}"
        ufn: DELLEMC
        banner: This platform is managed by Ansible

  tasks:

      - name: Gather facts about the Brocade DS-6620B Device
        brocade_facts:
                credential: "{{ credential }}"
                vfid: -1
                timeout: 300
                gather_subset:
                        - brocade_fibrechannel_switch
        register: results

      - name: Show the collected facts
        debug:
                var: results
                verbosity: 1

      - name: switch configuration
        brocade_fibrechannel_switch:
                credential: "{{ credential }}"
                vfid: -1
                switch:
                        banner: "{{ banner }}"
                        user_friendly_name: "{{ ufn }}"
        register: result

      - name: gather facts about the Brocade DS-6620B Device
        brocade_facts:
                credential: "{{ credential }}"
                vfid: -1
                timeout: 300
                gather_subset:
                        - brocade_fibrechannel_switch

      - name: Determine if the user_friendly_name was updated
        debug:
                var: ansible_facts.brocade_fibrechannel_switch[0].user_friendly_name


      - name: Determine if the banner was updated
        debug:
                var: ansible_facts.brocade_fibrechannel_switch[0].banner
```

18. Save and exit with `:wq`

19. Run the newest playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-brocade02.yml -i ~/ans/inv/prod/
```

20. That should be enough to get you started automating Brocade devices.

53. Introduction to Jenkins

Lab Objective

The objective of this lab is to learn how to use Jenkins. Jenkins allows the creation of "jobs", which can be any flavor of script, run any number of times, at any time. It's like Ansible Tower, if Ansible Tower let you do **more**!

The documentation on Jenkins is rather rich, and supported by an active user base. Deployed on Linux *or* Windows platforms, Jenkins offers licensed support, but it is also easy to get up and running without support. These are just *some* of the reasons many Enterprise customers are turning away from Ansible Tower in favor of Jenkins!

Jenkins can be found here:
https://jenkins.io/

Procedure

1. Open a new terminal, then change directory to the `/home/student/` directory.

```
student@bchd:~$ cd ~
```

2. Download the tear-down script (you may already have this):

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh
```

3. Run the tear-down script.

```
student@bchd:~$ bash max-teardown.sh
```

4. Download the setup script for this lab.

```
student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/pexpress-setup.sh
```

5. Run the setup script for this lab.

```
student@bchd:~$ bash pexpress-setup.sh
```

6. Ping the fry host created by the `pexpress-setup.sh` script. We won't run Jenkins on the fry host, we're going to run Jenkins on our controller. However, we do need a host for Jenkins to target with a playbook.

```
student@bchd:~$ ping -c 1 10.10.2.4
```

7. **IMPORTANT** - To connect to the host `fry`, we're going to use a password. Ensure that `sshpass` is installed, or Ansible cannot work with password credentials.

```
student@bchd:~$ sudo apt install sshpass -y
```

8. Read about Jenkins by checking out the official Jenkins website here: https://jenkins.io/

9. Installing Jenkins isn't tricky, but there's a few steps involved, like ensuring Java is installed. Review the steps for installing on a Debian system here: http://pkg.jenkins-ci.org/debian-stable/

> If you're running on RedHat, you'll want to check out http://pkg.jenkins-ci.org/redhat-stable/

10. Alta3 Research prepared a playbook to install Jenkins. Create a new playbook.

```
student@bchd:~$ vim ~/ans/playbook-jenkins.yml
```

11. Create the following playbook solution. Be sure to read through all the comments. You might want to update the playbook to the latest available version. This information is available via the install page http://pkg.jenkins-ci.org/debian-stable/

```
# cat linux_jenkins.yml
---
- name: Install Jenkins
  hosts: localhost
  gather_facts: no
  become: yes

  # Manual Jenkins install instructions can be found @
  # Debian:
  # http://pkg.jenkins-ci.org/debian-stable/
  # Redhat:
  # http://pkg.jenkins-ci.org/redhat-stable/

  vars:
          jenkins_state: present  # present or absent
          jenkins_version: "2.204.5"  # 2.204.5 released on 2020/03/07
          jenkins_pkg_url: "http://pkg.jenkins-ci.org/debian-stable/binary"
          jenkins_http_port: 8087  # port to run jenkins on
          jenkins_init_file: /etc/default/jenkins # contains global settings for jenkins

  tasks:
        # Jenkins requires Java
        # The best way to install Java on linux is via open-java
        # in apt repo this is called default-jdk
        - name: Ensure dependencies are installed
          apt:
            name:
              - default-jdk
              - curl
              - apt-transport-https
              - gnupg
            state: "{{ jenkins_state }}"

        # add Jenkins repo key per
        # http://pkg.jenkins-ci.org/debian-stable/
        - name: Add Jenkins apt repository key
          apt_key:
            url: https://pkg.jenkins.io/debian-stable/jenkins.io.key
            state: "{{ jenkins_state }}"

        - name: Add Jenkins apt repository
          apt_repository:
            repo: deb https://pkg.jenkins.io/debian-stable binary/
            state: "{{ jenkins_state }}"
            update_cache: true
        ## is this when necessary?  when: jenkins_repo_url | default(false)

        # the versions of jenkins are available via
        # http://pkg.jenkins-ci.org/debian-stable/
        - name: Download specific Jenkins version
          get_url:
            # http://pkg.jenkins-ci.org/debian-stable/binary/jenkins_2.204.5_all.deb
            url: "{{ jenkins_pkg_url }}/jenkins_{{ jenkins_version }}_all.deb"
            dest: "/tmp/jenkins_{{ jenkins_version }}_all.deb"
          when: jenkins_version is defined

        - name: Install block
          block:
            - name: Install our specific version of Jenkins
              apt:
```

```
        deb: "/tmp/jenkins_{{ jenkins_version }}_all.deb"
        state: "{{ jenkins_state }}"
      when: jenkins_version is defined

  rescue:
    # edit the port that Jenkins runs on by editing
    # the Jenkins configuration file
    - name: Set the port on which Jenkins runs
      lineinfile:
        backrefs: true
        dest: "{{ jenkins_init_file }}"
        regexp: '^HTTP_PORT='
        line: 'HTTP_PORT={{ jenkins_http_port }}'

- name: Install block
  block:
    - name: Install our specific version of Jenkins
      apt:
        deb: "/tmp/jenkins_{{ jenkins_version }}_all.deb"
        state: "{{ jenkins_state }}"
      when: jenkins_version is defined

# edit the port that Jenkins runs on by editing
# the Jenkins configuration file
- name: Set the port on which Jenkins runs
  lineinfile:
    backrefs: true
    dest: "{{ jenkins_init_file }}"
    regexp: '^HTTP_PORT='
    line: 'HTTP_PORT={{ jenkins_http_port }}'

- name: restart jenkins
  service:
    name: jenkins
    state: restarted

- name: let jenkins come up for the first time
  pause:
          seconds: 30

# grab initial Admin password
- name: Get init password Jenkins
  shell: cat /var/lib/jenkins/secrets/initialAdminPassword
  changed_when: false
  register: result

# display the initial Admin password on the screen
- name: Print init password Jenkins
  debug:
    var: result.stdout
```

12. Save and exit.

13. Run the playbook.

```
student@bchd:~$ ansible-playbook ~/ans/playbook-jenkins.yml
```

14. If the playbook completes, the admin password will be displayed. It is a long UUID. Copy down this value.

15. Switch to your remote desktop GUI by changing your url from `/tmux/` to `/?path=view/`

16. At the bottom of the screen, on your remote desktop GUI, click the Firefox icon. It may take a moment to open.

17. After launching, navigate to: http://127.0.0.1:8087

    Be sure to set the port to reflect the value you set for `jenkins_http_port` in the playbook.

18. You should be at the sign in page for Jenkins. It will be prompting for the admin UUID. This value was displayed by your playbook. Enter it here.

    The initial admin password is stored in the file `/var/lib/jenkins/secrets/initialAdminPassword`

19. **Click Install Suggested Plugin**

    Plugins are the way the community adds features to Jenkins. You're welcome to go explore plugins beyond the basics, but that is outside the scope of this introduction to Jenkins.

20. Wait for the *Getting Started* screen to finish.
21. Fill out the *Create First Admin User* as follows:
    - **Username**: admin
    - **Password**: qwerty123
    - **Confirm password**: qwerty123
    - **Full name**: ansible student
    - **E-mail address**: ansiblestudent@example.com
22. Click **Save and Continue**
23. On the *Instance Configuration* screen, click **Save and Finish**.
24. Click on **Start using Jenkins**
25. Click on **create new jobs**
26. **Enter an item name**: lotr
27. Click on **Freestyle project**
28. Click on **OK** at the bottom of the screen.
29. Fill the *General tab* (which will change into other tabs as you scroll down the screen) out as follows:
    - **Description**: Pull playbook from GitHub and run the job
    - **GitHub project box**: Check
        - **Project url**: https://github.com/rzfeeser/simple-jenkins-playbook
    - **Source Code Management**:
        - **Git circle**: Check
            - **Repository URL**: https://github.com/rzfeeser/simple-jenkins-playbook
            - **Credentials**: None
30. Click on **Save** at the bottom of the screen.
31. After saving, you should be returned to the *Project lotr* screen. You can reach this screen by clicking the magnifying glass along the left. **Click the magnifying glass** now. You should remain on the same screen.
32. Clicking **Changes** will not display any changes. As you work through Jenkins, this page will populate with deltas from builds.
33. **Click on Workspace** it should say you haven't performed a build. Let's change that.
34. **Click on Build Now**.
35. Once again **click on workspace**
36. The screen *Workspace on lotr on master* is a reflection of the GitHub repo. If you haven't checked out the repository, you should now. It is available at: https://github.com/rzfeeser/simple-jenkins-playbook
37. **Click on playbook-tolkien.yml**

38. **Click the OK button** to open with vim.
39. The playbook should open. Examine the simple playbook. Looks like it creates a directory `~/onering/`, and then creates a file inside of this directory called, `torulethemall.txt`. Seems appropriate for the `lotr` project!
40. **Close the vim window**
41. Click on **Wipe out Current Workspace**

    This click will purge Jenkins of any recent sync with the Software Control Management (SCM) platform associated with the project.

42. **Click on Workspace**. The screen will display *Error: no workspace*. That's because we wiped out in the last step. Let's resync Jenkins with our Software Control Management (SCM) platform now.
43. **Click Build Now** to resync with your SCM.
44. You'll notice a new job number appear in your *Build History* when you perform this operation. Think of this as a log of what's happening in Jenkins. **Click on the highest number within the Build History**
45. This is the logged operation for the sync that just occurred. Looks like `ansible student` just pulled from master.
46. **Click Console Output** along the left of the screen.
47. Looks like the results of the operation that was performed by Jenkins. The take away is that Jenkins just created a local repository, pulled in our project repository, and then checked out the latest commit. That was a lot of work that just got performed in a few mouse clicks.
48. Click on **Back to Project**
49. Click on **Back to Dashboard**
50. The dashboard provides a 'heads up' view of jobs Jenkins is currently managing. The table is understood as follows:
    - **S** - status of last job run
    - **W** - Weather report showing aggregate of last few builds. Sunny is good! Stormy is bad.
    - **Last Success** - The time and name of the last job to successfully run.
    - **Last Failure** - The time and name of the last job that crashed and burned.
    - **Last Duration** - The time it took to run the last job.
51. So far, all that our project does is, via button clicks, pull the latest code from GitHub. Part of the issue, is that we should install some plugin support for Ansible.
52. Click on **Manage Jenkins**
53. Click on **Manage Plugins**
54. Click on the **Available** tab*
55. In the **Filter**: box type **Ansible**
56. Check the box on **Ansible**
57. Click **Install without restart**
58. The install should complete quickly, and the page will auto refresh.
59. Click **Back to Dashboard**
60. Click the **lotr link**
61. Click on **Configure**
62. Scroll down to **Build Triggers**
63. **Check the box Poll SCM**
64. In the *Schedule* box it is possible to schedule when you want the job to run. Scheduling is done via MINUTE, HOUR, DAY OF THE MONTH, MONTH, and DAY OF THE WEEK. It is like crontab, with some minor adjustments. Description of the syntax can be found under the Scheduling box.

65. Place the following in the **Schedule** box, which tells Jenkins to scan the SCM for updates every 5 minutes:

```
H/5 * * * *
```

> Another popular option is to create a GitHub hook to trigger Jenkins to begin scanning for changes to the SCM. Instructions on this option can be found by checking the box "GitHub hook trigger for GITScm polling"

66. Continue to scroll down until you see **Build**

67. Remember that Ansible plugin we installed? Time to test it out. Choose **Invoke Ansible Playbook** from the drop down.

68. Fill out the **Invoke Ansible Playbook** as follows:

    - **Playbook path**: playbook-tolkien.yml
    - **Inventory**: Inline content
        - **Check** Dynamic Inventory
        - **Content** box:

69. In the `content` box, create the following Dynamic Inventory script:

```
#!/usr/bin/python3
"""Russell Zachary Feeser || rzfeeser@alta3.com"""

## for accepting arguments from user at CLI
import argparse

## required for working with JSON (pystd. library)
import json

def main():
    # this is what we will ultimately return
    inventory = {}

    # Called with `--list`
    if args.list:
        # inventory is result of calling example_inventory()
        inventory = example_inventory()
    # Called with `--host [hostname]`
    elif args.host:
        # Not implemented, since we return _meta info `--list`
        # Let's put API request logic here...
        inventory = empty_inventory()
    # If no groups or vars are present, return an empty inventory.
    else:
        inventory = empty_inventory()

    # print the result of inventory
    print(json.dumps(inventory))  # from the json library use the
                                  # DUMPString function, dumps()

# Example inventory for testing.
def example_inventory():
    return {
        'group': {
            'hosts': ['bender', 'fry'],
            'vars': {
                'example_var1': 'proxyeast',
                'example_var2': 'proxywest'
            }
        },
        '_meta': {
            'hostvars': {
                'bender': {
                    'ansible_user': 'bender',
                    'ansible_host': '10.10.2.3',
                    'ansible_ssh_pass': 'alta3',
                    'fileuser': 'bender'
                },
                'fry': {
                    'ansible_user': 'fry',
                    'ansible_host': '10.10.2.4',
                    'ansible_ssh_pass': 'alta3',
                    'fileuser': 'fry'
                }
            }
        }
    }

# Empty inventory for testing.
def empty_inventory():
```

```
        return {'_meta': {'hostvars': {}}}

    if __name__ == "__main__":
        parser = argparse.ArgumentParser()
        parser.add_argument('--list', action = 'store_true')
        parser.add_argument('--host', action = 'store')
        args = parser.parse_args()
        main()
```

70. The remaining fields can remain blank:
    - **Host subset**:
    - **Credentials**:
    - **Vault Credentials**:
    - **Do NOT check become**
    - **Do NOT check sudo**
71. Click the **Advanced** button (bottom right)
72. Check the box **Disable the host SSH key check**
73. Now click **Save**
74. To recap, we told Jenkins the following:
    - Pull changes from `github.com/rzfeeser/simple-jenkins-playbook` every 5 minutes
    - On a successful pull, Jenkins should run the Ansible playbook `playbook-tolkien.yml`
    - The candidate list is provided by a Dynamic Inventory script, however the hosts to run against are selected in the playbook (fry)
75. To force Jenkins to run *now* click on the **Build Now** link. You'll see a new job appear in the *Build History*
76. Click on the newest job number in the *Build History*
77. Click on **Console Output**
78. You should see the *Console Output* screen where you see a `git fetch` performed, followed by an Ansible Playbook `PLAY RECAP`. Note that it looks like Jenkins made a change to Fry.
79. Note the current job number. Every 5 minutes, Jenkins will scan GitHub for changes, then run the playbook. You'll see a new job number appear, the results of a GitHub fetch, and then finally the results of running the playbook.
80. Check the `fry` box to ensure the changes were actually performed. Change the `/?path=view/` back to `/tmux/`
81. SSH to the fry machine.

    ```
    student@bchd:~$ ssh fry@10.10.2.4
    ```

82. Check to see `~/onering/torulethemall.txt` exists.

    ```
    fry@fry:~$ ls ~/onering/
    ```

83. Awesome! So, we only scratched the surface of Jenkins, but that should be more than enough to get you up and running! Good luck continuously deploying with Jenkins and Ansible!

54. Case Study: Ansible Tower

Lab Objective

*This lab is a **case study**, and is designed to reflect acquired knowledge. Ansible Tower is very demanding of resources, and as such, needs to be deployed on a system larger than you are currently on. Reading through the lab can still offer valuable information on building and using Ansible Tower / AWX tower.*

The objective of this lab is to learn to install AWX Tower to learn more about its capabilities. We will be using AWX instead of Ansible Tower, as this is the bleeding edge of the suite. Think of AWX as the Fedora of RedHat. The first part of the lab, we will set up and install AWX. The second part we will demonstrate using AWX.

AWX has been "restricted" to only running on RHEL or CentOS, this lab shows a 'work around' to allow this run on top of Debian / Ubuntu.

Procedure

Part 1: Install AWX

1. Start in your home directory.

   ```
   student@bchd:~$ cd ~
   ```

2. Prepare for the installation by updating your machine with the latest repo information.

   ```
   student@bchd:~$ sudo apt-get update
   ```

3. Update to the latest packages.

   ```
   student@bchd:~$ sudo apt-get upgrade
   ```
   (This may take a few minutes to complete)

4. Download the lab teardownscripts. This will help to free up resources, ensuring AWX may be installed and run on your beachhead machine.

   ```
   student@bchd:~$ wget https://static.alta3.com/projects/ansible/deploy/max-teardown.sh -O max-teardown.sh
   ```

5. Run the following script to teardown any running VMs.

   ```
   student@bchd:~$ bash max-teardown.sh
   ```

6. Run the following command to, **Install the latest release of Ansible**

   ```
   student@bchd:~$ echo "deb http://ppa.launchpad.net/ansible/ansible/ubuntu bionic main" | sudo tee /etc/apt/sources.list.d/
   ansible.list
   ```

7. Add the following key to your apt repo.

   ```
   student@bchd:~$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 93C4A3FD7BB9C367
   ```

8. Update your apt utility.

   ```
   student@bchd:~$ sudo apt update
   ```

9. Install `nodejs` and `npm` on beachhead.

   ```
   student@bchd:~$ sudo apt install -y nodejs npm
   ```

10. Use the utility to install npm package. Use the global flag to make it available to all users.

    ```
    student@bchd:~$ sudo npm install npm --global
    ```

11. Install password generator `pwgen`.

    ```
    student@bchd:~$ sudo apt -y install pwgen #python-pip git
    ```

12. This is also probably installed....

```
student@bchd:~$ sudo pip install requests==2.14.2
```

13. Switch to root. This probably is not necessary...

```
student@bchd:~$ sudo su -
```

14. As root, clone the git repo.

```
root@ans-335-bchd:~# git clone --depth 50 https://github.com/ansible/awx.git
```

15. Move into the installed directory.

```
student@bchd:~$ cd awx/installer/
```

16. Generate a password. Record this. You'll need to place it in your inventory.

```
student@bchd:~$ pwgen -N 1 -s 30
```

17. Edit your inventory file.

```
student@bchd:~$ vim inventory
```

18. Record the value produced above. Use it to modify your inventory file. Notable changes include updating the location of `ansible_python_interpreter`. You also need to update the `secret_key=` with the value calculated by `pwgen`.

```
localhost ansible_connection=local ansible_python_interpreter="/usr/bin/env python3"
[all:vars]
dockerhub_base=ansible
awx_task_hostname=awx
awx_web_hostname=awxweb
postgres_data_dir=/tmp/pgdocker
host_port=80
host_port_ssl=443
docker_compose_dir=/tmp/awxcompose
pg_username=awx
pg_password=awxpass
pg_database=awx
pg_port=5432
rabbitmq_password=awxpass
rabbitmq_erlang_cookie=cookiemonster
admin_user=admin
admin_password=alta3pass
create_preload_data=True
secret_key=2fCkx2K5GnIjBz4OterhOC3ey0WPdj
```

19. Save and exit.

20. **WARNING - IF YOU RUN THIS PLAYBOOK. YOUR BEACHHEAD MACHINE WILL BECOME UNSTABLE** Your machine needs additional resources to support AWX. Rather than run this playbook yourself, ask the instructor if they can demo Ansible Tower or AWX.

```
student@bchd:~$ ansible-playbook install.yml -i inventory
```

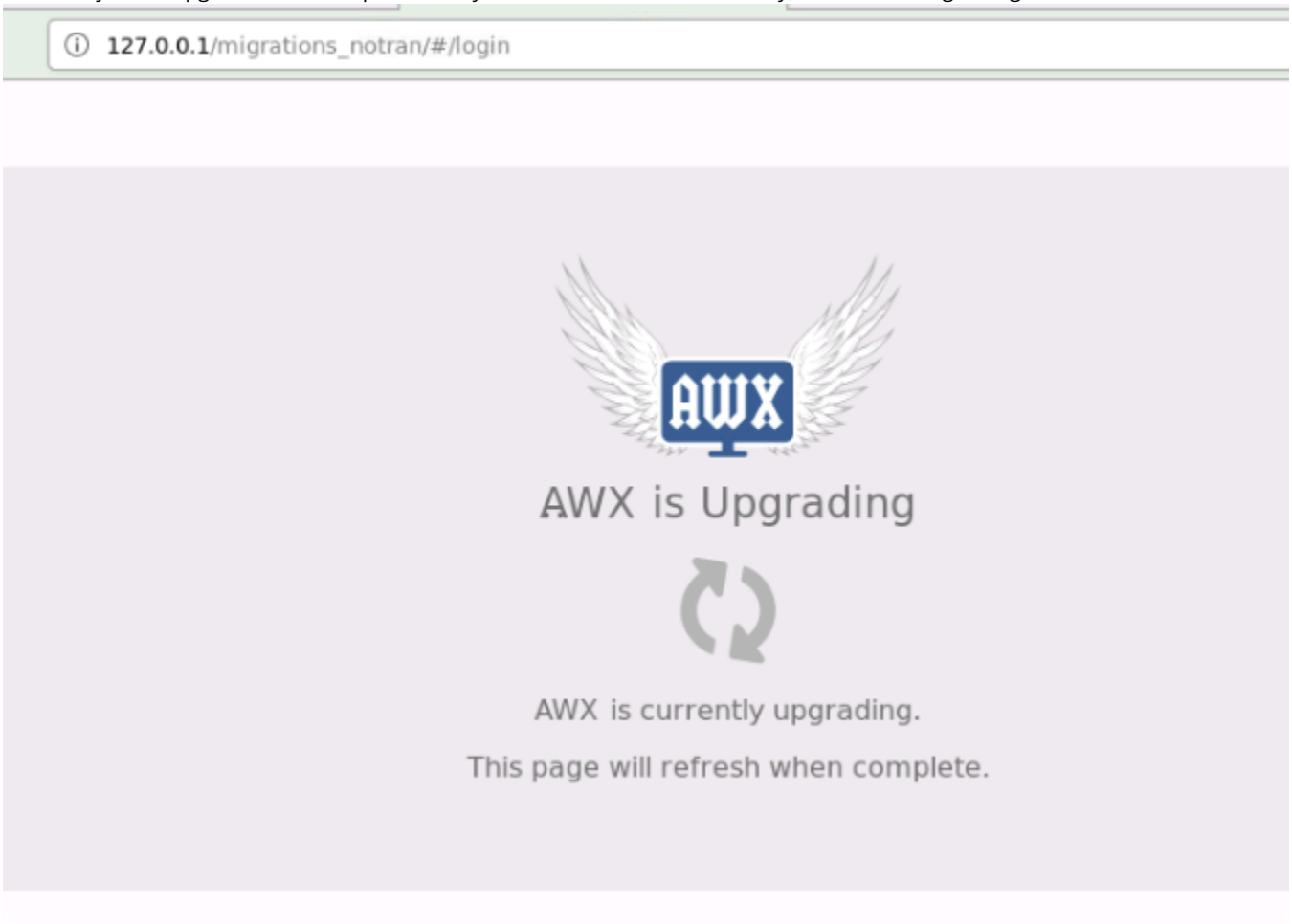
21. This will run for awhile.

22. When the playbook finishes, Ansible AWX will be successfully deployed.

23. Open up a browser tab. AWX is waiting for you. Type in the URL below.

```
http://127.0.0.1
```

24. There may be an upgrade that takes place once you enter the URL. Do not worry, there is nothing wrong.



25. Once the upgrade is complete, enter your credentials to login.

```
login: admin/password
```

26. Back on beachhead, you can see the actual docker containers used for running AWX. Look at the results below.

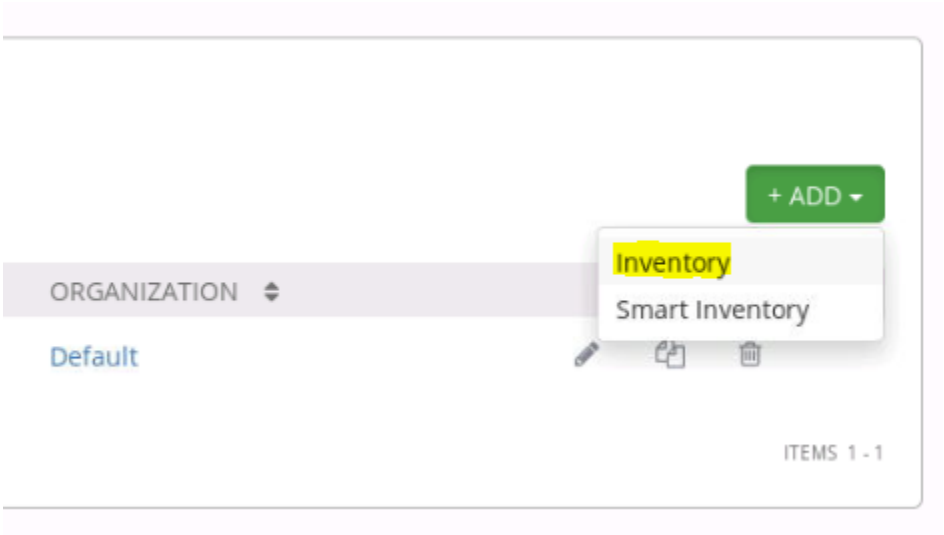
```
student@bchd:~/awx/installer$ sudo docker ps
```

```
student@beachhead:~/awx/installer$ sudo docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS                                  NAMES
8dbe32644009        ansible/awx_task:latest   "/tini -- /bin/sh -c…"   9 minutes ago       Up 9 minutes        8052/tcp                               awx_task
549ba0e0d00a        ansible/awx_web:latest    "/tini -- /bin/sh -c…"   9 minutes ago       Up 9 minutes        0.0.0.0:80->8052/tcp                   awx_web
6fb1a545de6b        memcached:alpine          "docker-entrypoint.s…"   10 minutes ago      Up 10 minutes       11211/tcp                              memcached
239413930e16        rabbitmq:3                "docker-entrypoint.s…"   10 minutes ago      Up 10 minutes       4369/tcp, 5671-5672/tcp, 25672/tcp     rabbitmq
5435724ce1ee        postgres:9.6              "docker-entrypoint.s…"   10 minutes ago      Up 10 minutes       5432/tcp                               postgres
```

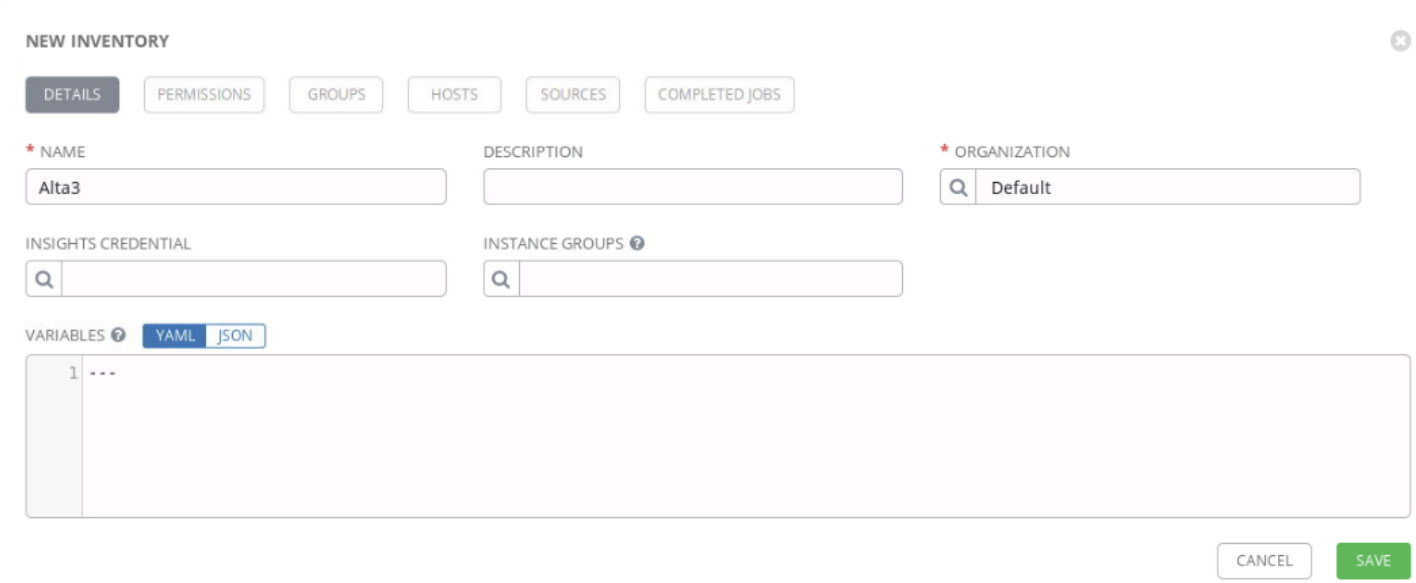
Part 2: Using AWX

27. Let's create an inventory. Click on `INVENTORIES` on the top menu.

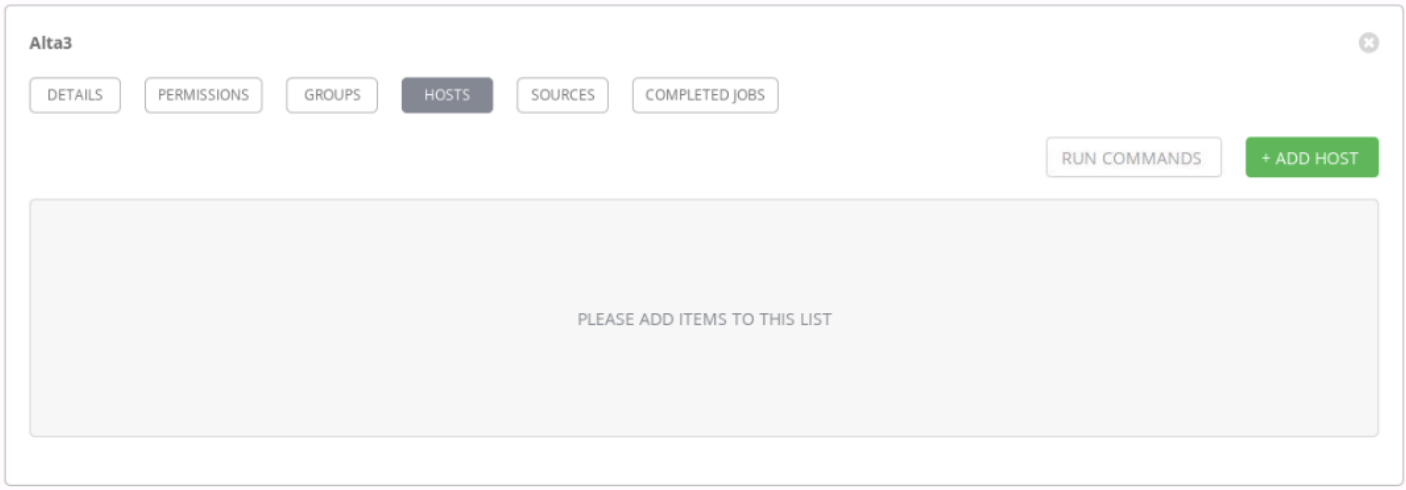
28. Click on the green `+ADD` button to add a project, then click on `Inventory` in the dropdown.



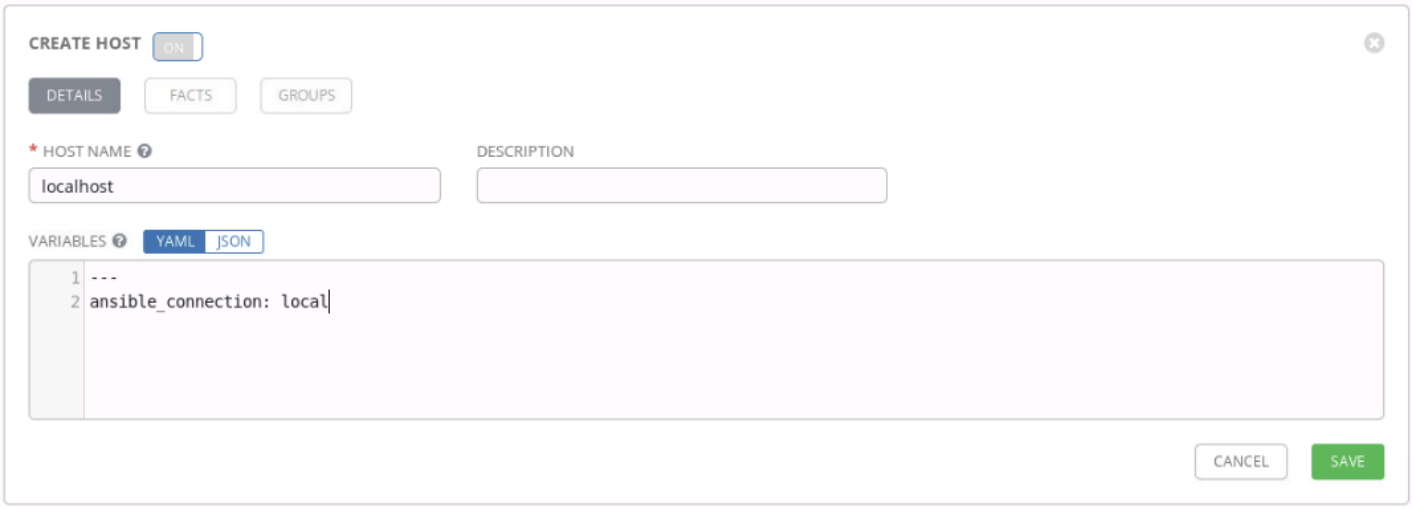
29. Give a name to your inventory and then click the green save button on the right-hand side.



30. Under the Alta3 title, click on HOSTS and then click on the green `+ADD HOST` button.

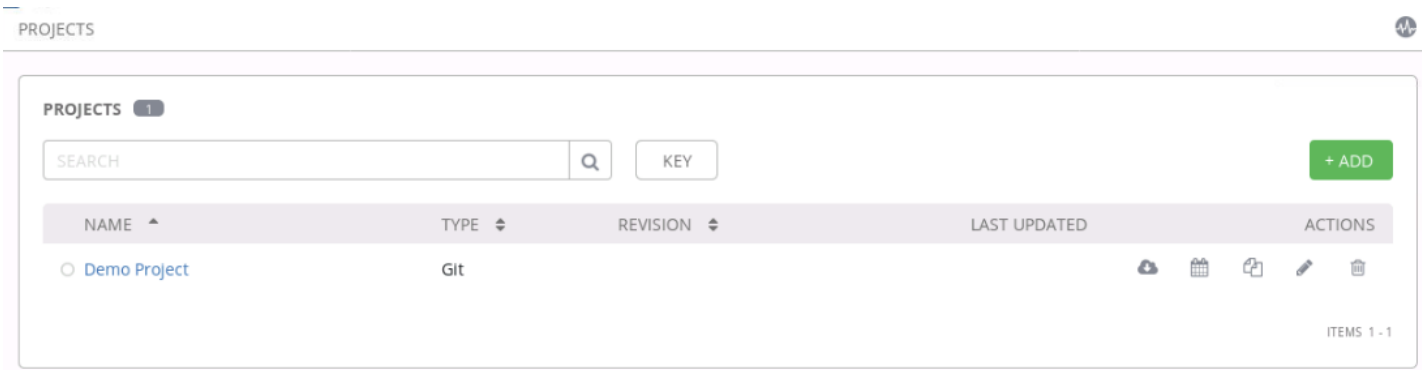


31. Name your host, add variables, and then click the green save button on the right.

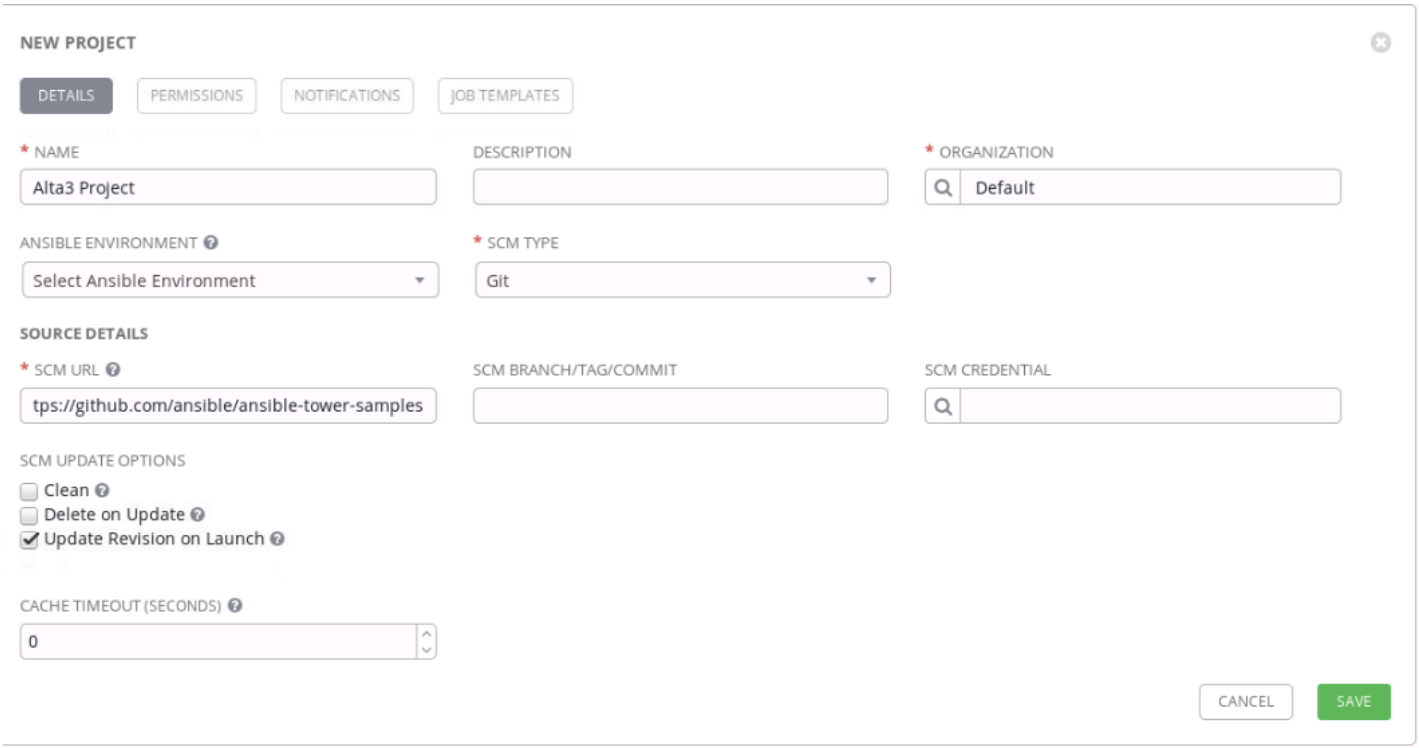


32. Now we will create a project. This is where AWX or Tower will pull the playbook from. For our example, we will be pulling from a GitHub repo. Click on the AWX icon on the top left-hand corner to get back to the main menu, and then click `PROJECTS` on the top menu.

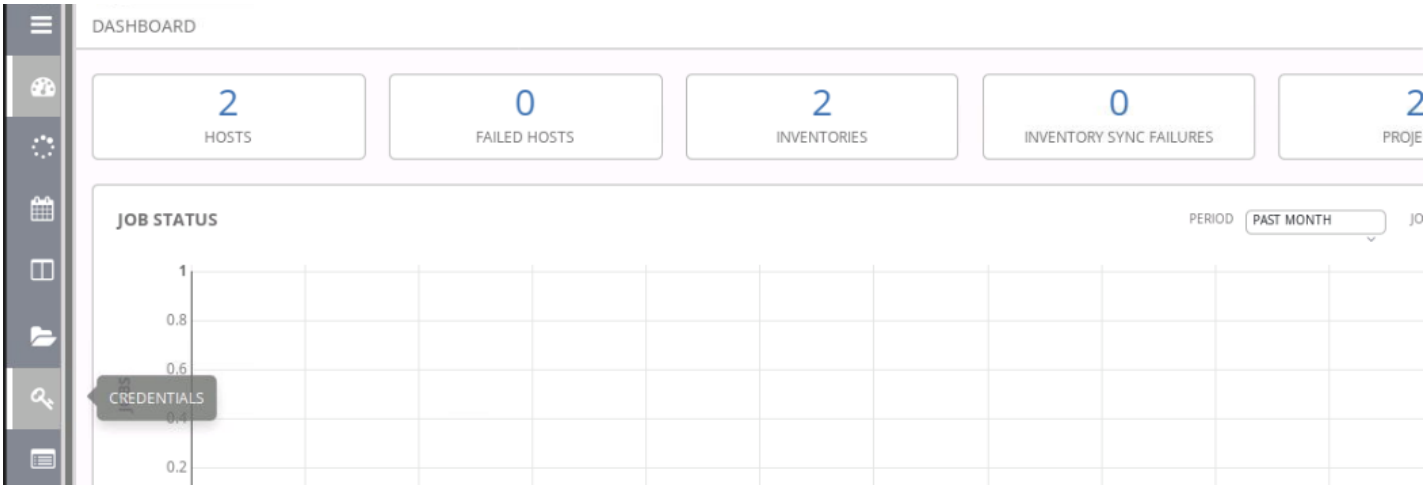
33. In the Projects page, click the green `+ADD` button to add a new project.



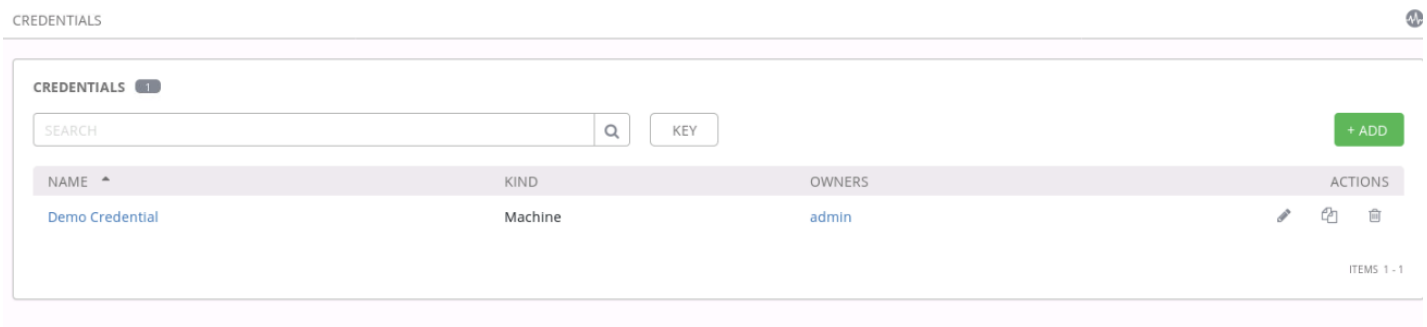
34. Fill in the Name, SCM TYPE, add SOURCE DETAILS and click the green save button on the right. The entire url is: `https://github.com/ansible/ansible-tower-samples`.



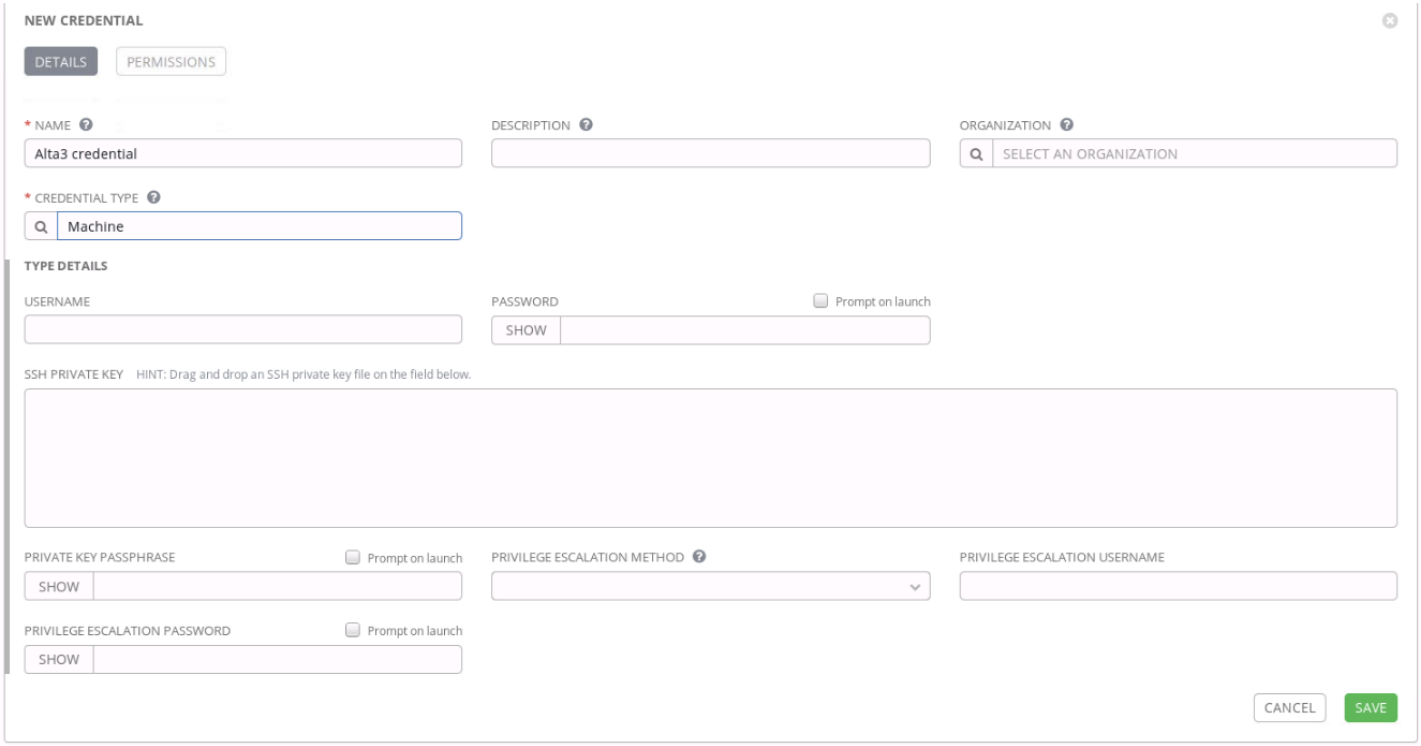
35. Now let's add a credential. In order to run a playbook, you must add a credential to tell AWX/Tower how to login to the target system. We will create a blank credential for this instance, but you can also easily add a username/password or SSH key as well. Click the AWX icon again to get back to the main menu, and then click the key icon down the left-hand pane. Note - in Tower, it would be the gear icon at the top right.



36. Once on the Credentials page, click the `+ADD` button to add a credential.

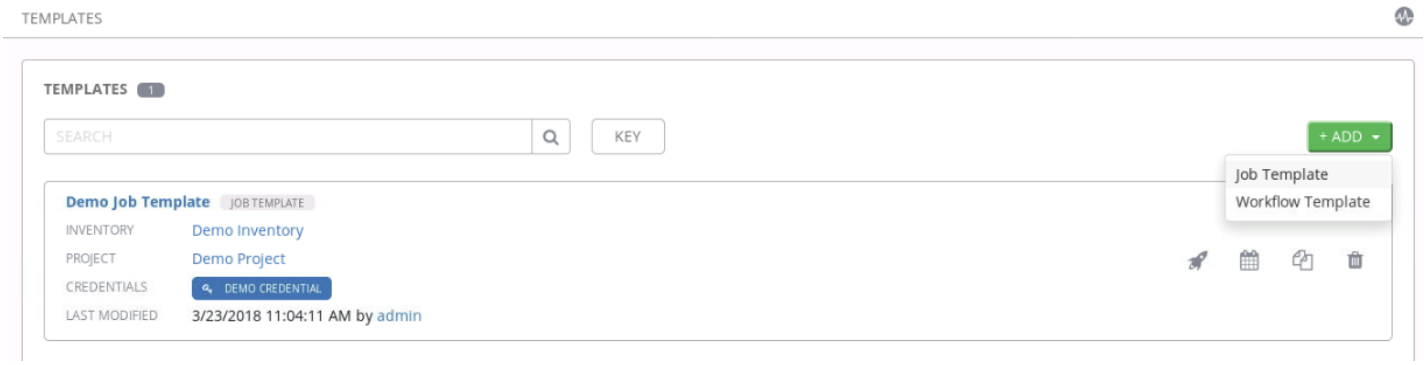


37. Select the type of credential, name the credential, and click the green save button.

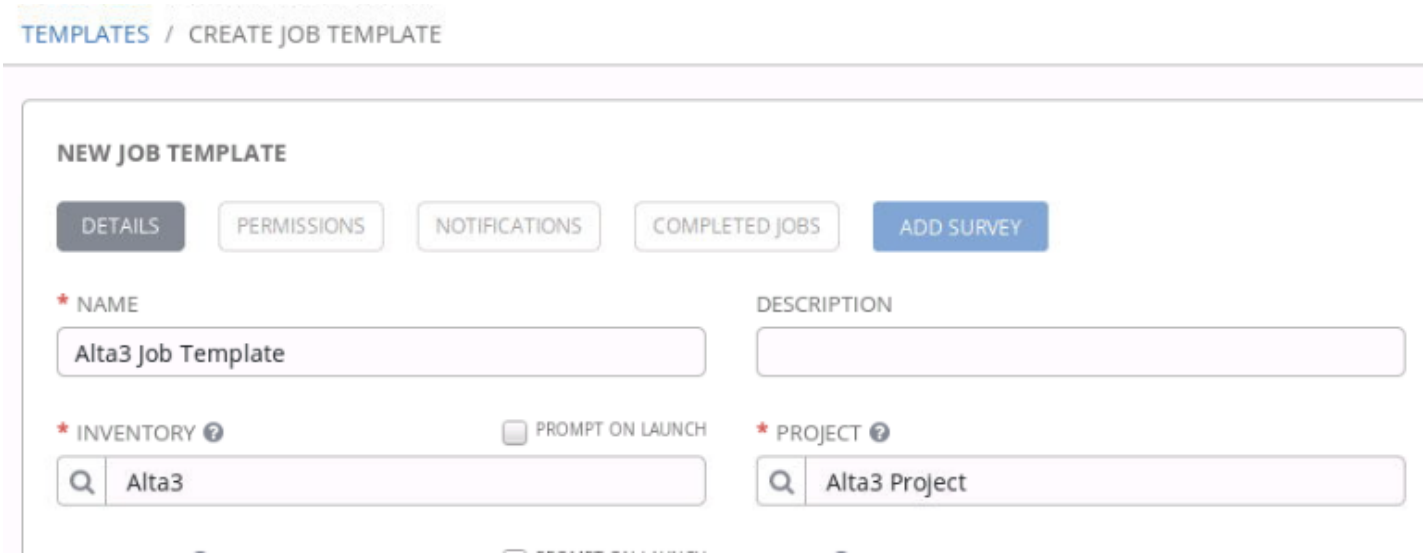


38. How about we create a Job Template. This will tell AWX/Tower how to run the playbooks. Let's go back to the main page and click on TEMPLATES icon down the left-hand pane. Note - in Tower, this would be on the top menu and called JOB TEMPLATES.

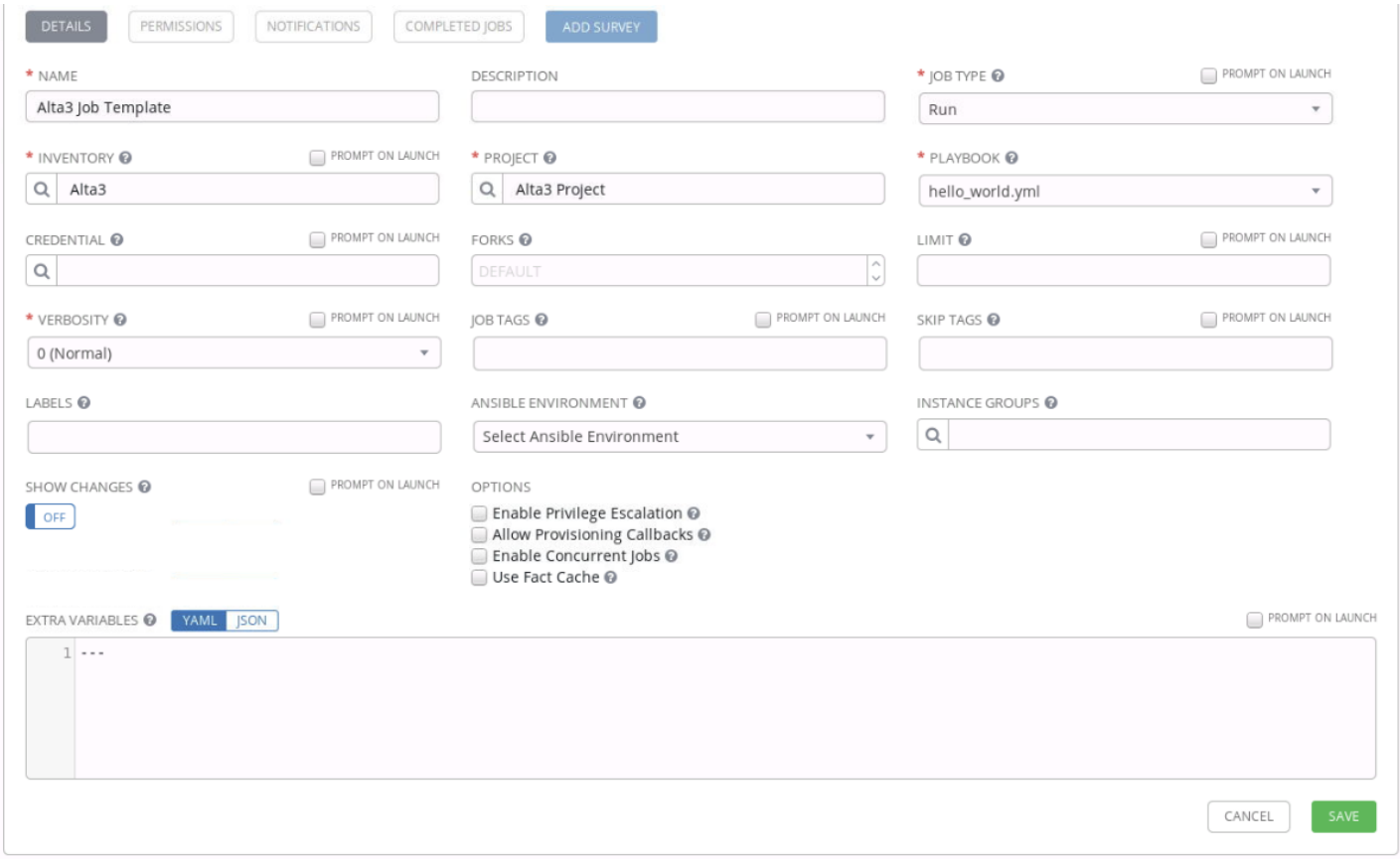
39. Once on the Templates page, click the green `+ADD` button and then click `Job Template` to add a job Template.



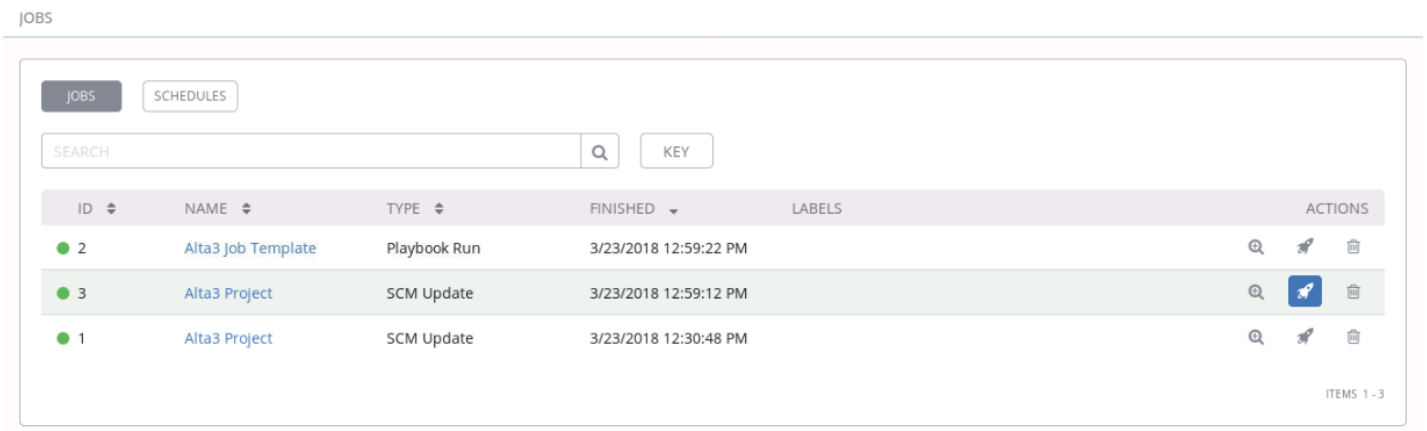
40. Name the Job Template, and fill in Inventory, and project.



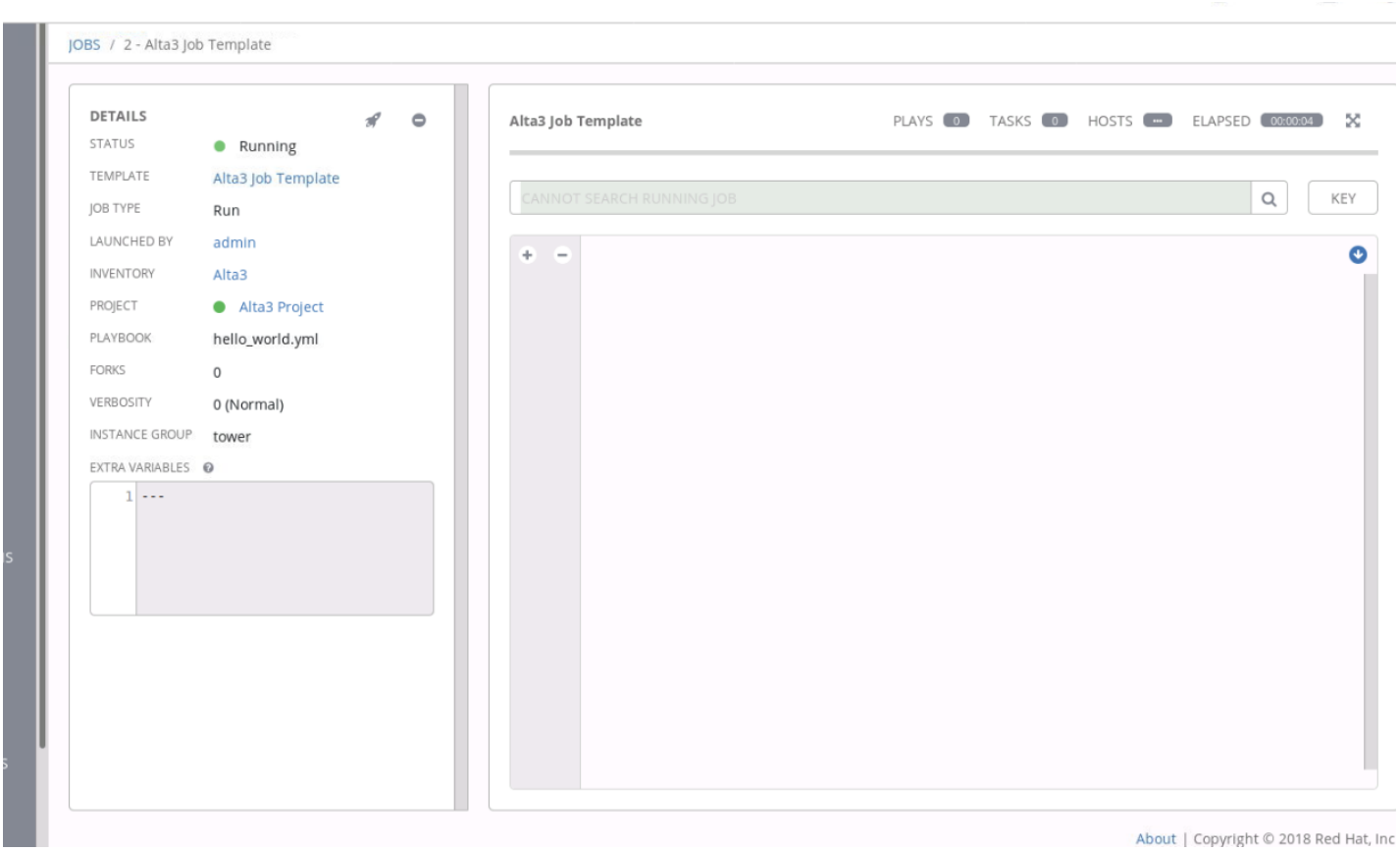
41. Then select the job type, choose a playbook, and click the green save button.



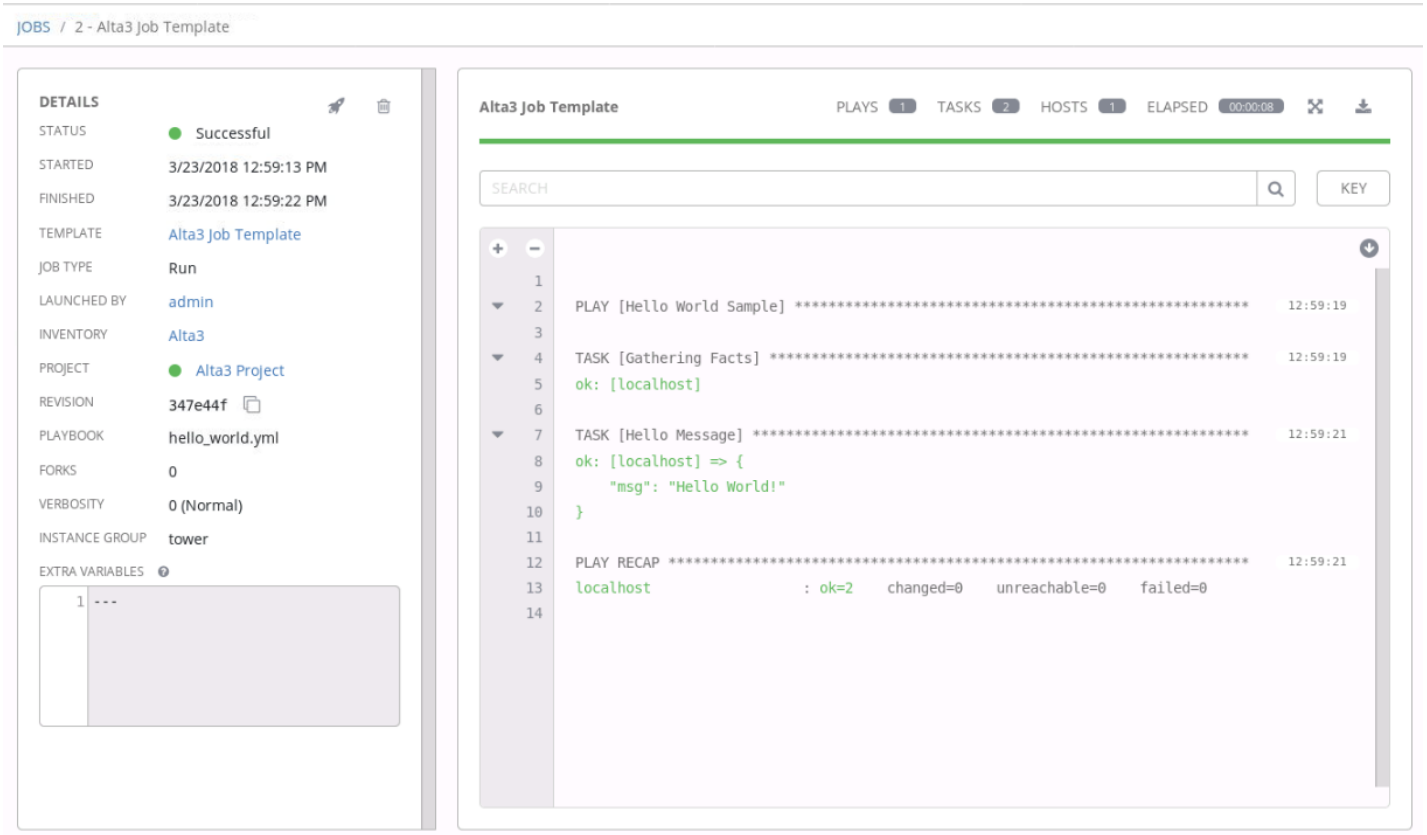
42. That wasn't too bad, was it? Let's run the job!! From the current page, click on `TEMPLATES` at the top left of your screen. This will return you to the main TEMPLATES page. This page is where you create, and run assembled templates. Find the template you created, and then click on the little rocket ship icon to run the job.



43. Look at the progress of the job. Is it pending, running or has it completed?



44. Check out the completed job! Whoo, you did it! Congratulations.



55. Ansible Best Practice

The objective of this lab is to outline some basics regarding Ansible and best practice. It is expected that the enterprise will define rules beyond *just* these, however, the following are generally applicable on every Ansible project.

Limit Variable Definitions

Agree on a limited number of places to define variable precedence. Allowing users to define the same variable 21 different times is compelling, but doesn't scale well. Consider limiting where users might define variables. For example, banning the use of `vars_prompt` (which stops an automated playbook and waits for the answer to a question), might be a good start.

- https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#understanding-variable-precedence

Keep Building Roles

Creating libraries of roles allows your Ansible devs to easily reuse code. Roles should be named to promote clarity. Something like `prechecks-campusWest-cisco-ios` is certainly better than `prechecks`. Developing roles should include a peer review process before they are added to a collection, or otherwise made available for use.

- https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html
- https://galaxy.ansible.com/docs/contributing/creating_role.html

Use Collections

Use collections to augment Ansible. Collections are simply highly organized folders. They can easily be added to an existing Ansible deployment, or added to a deployment at the time of creation. Collections may include roles, playbooks, modules, documentation, scripts, and just about anything else that might be useful in running Ansible.

- https://docs.ansible.com/ansible/latest/user_guide/collections_using.html
- https://www.ansible.com/blog/hands-on-with-ansible-collections
- https://docs.ansible.com/ansible/devel/dev_guide/developing_collections.html

Spacing

Generous use of spacing makes playbooks easy to scan. Agree on "how many spaces" your project or enterprise will use (2, 3, or 4?). Consistency across all YAML documents increases overall Ansible grok.

- https://yaml.org/spec/1.2/spec.html

Always Mention 'state:'

Define the `state:` parameter, even if it is is optional. Different modules have different default settings for `state:`, and some modules support several `state:` settings. Explicitly setting `sate: present` or `state: absent` makes playbooks and roles clearer.

- https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#desired-state-and-idempotency
- https://docs.ansible.com/ansible/latest/user_guide/modules_intro.html

Use 'name:' and description comments

Always 'name:' your plays, blocks, and tasks, and use descriptive comments. Names and comments should be descriptive. For example, `# copy module` is a poorly written comment, whereas, `# replacing phoenix-app cfg file` is well formed. Your goal is to provide additional detail (and therefore understanding) to your code, not duplicate information that is already apparent.

- https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html

Remain Idempotent

Generally, playbooks should remain idempotent (not indicate changes if run twice in a row). An operation is idempotent if the result of performing it once is exactly the same as the result of performing it repeatedly without any intervening actions. Failure to adhere to this rule may result in excessive handler calls (loss of services via restarts), a lack of confidence in the ansible results (addressing configuration drift), and a decrease in the overall predictable nature of Ansible. As some modules are notoriously difficult to keep idempotent, it is *OK* to "outlaw" the use of certain modules.

- https://docs.ansible.com/ansible/latest/user_guide/plugin_filtering_config.html
- https://hvops.com/articles/ansible-vs-shell-scripts/
- https://medium.com/@relativkreativ/how-to-use-ansibles-lineinfile-module-in-a-bulletproof-way-e2c75e0aa6bb

Write Readable Playbooks

Whenever you can, do things simply. Use advanced features only when necessary, and select the feature that best matches your use case. For example, you will probably not need vars, vars_files, vars_prompt and --extra-vars all at once, while also using an external inventory file. If something feels complicated, it probably is. Take the time to look for a simpler solution.

- https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html

Use an SCM

Ansible code needs to be managed via a Software Control Management platform such as GitHub, GitLab, or BitBucket. Ansible developers should be required to use techniques like branches, and pull requests. All pull requests should be peer reviewed before being approved. Adhering to rules like the ones we are outlining make for efficient code reviews and reinforce best practice coding standards.

- https://bitbucket.org/blog/new-cloud-code-review-experience

Ansible Devs Study Python

You will write better Ansible code if you know Python. Python is the language that drives Ansible. The better your Ansible devs know Python, the more clever tricks and customizations they'll be able to create.

56. Linux Fundamentals

Lab Objective

The objective of this lab is to introduce some Linux Fundamentals. Not every person in this course has experience using a terminal to interact with a computer. This section is for those who may not have previous experience using a Linux terminal. If you are already proficient in Linux, feel free to skip this lab.

Procedure

1. First of all, take a look at your command prompt to try to understand what it can teach us. It should look like:

   ```
   student@bchd:~$
   ```

   - **student** is the name of the user
   - **bchd** is the hostname of the machine
   - **~** (to the right of the colon) shows us the present working directory. Specifically, **~** refers to this user's home directory of `/home/student`.
   - **$** shows us that this is a typical user (vs. **#** would indicate that it is the **root** user)

2. Now let's run some fundamental commands to get to know our environment a little bit better. Remember, that we are starting in our `/home/student` directory for this.

3. **pwd** - [present working directory] shows you what directory you are in.

   ```
   student@bchd:~$ pwd

   /home/ubuntu
   ```

4. **ls** - list out the contents of the current directory.

   ```
   student@bchd:~$ ls

   static
   ```

5. **cd** - [change directory] - allows you to move to a different directory. Here we can move to the static directory.

   ```
   student@bchd:~$ cd static
   ```

6. **mkdir** - [make directory] - allows you to make a new directory. Let's make one called **training**.

   ```
   student@bchd:~/static$ mkdir training
   ```

   Also, let's make sure we can see that we made this **training** directory.

   ```
   student@bchd:~/static$ ls

   training
   ```

   And let's move into the **training** directory.

   ```
   student@bchd:~/static$ cd training
   ```

7. **touch** - makes a blank file.

   ```
   student@bchd:~/static/training$ touch example_01.txt
   ```

   Verify that the file named **example_01.txt** is there now.

   ```
   student@bchd:~/static/training$ ls

   example_01.txt
   ```

8. **echo** - returns text to the standard output.

```
student@bchd:~/static/training$ echo Alta3 Research Training rocks!

Alta3 Research Training rocks!
```

The > character allows us to redirect standard output to a different place, normally a file.

```
student@bchd:~/static/training$ echo Alta3 Research has AWESOME labs! > myfile.txt
```

The >> characters allow us to redirect standard output and append it to the end of a file.

```
student@bchd:~/static/training$ echo Alta3 Research has AMAZING labs! >> myfile.txt
```

9. **cat** - [concatenate] prints file(s) to standard output.

```
student@bchd:~/static/training$ cat myfile.txt

Alta3 Research has AWESOME labs!
Alta3 Research has AMAZING labs!
```

10. **mv** - [move] allows us to move a file or directory (often used to rename files).

```
student@bchd:~/static/training$ mv myfile.txt ego_fuel.txt
```

Verify that the file has moved.

```
student@bchd:~/static/training$ ls

ego_fuel.txt example_01.txt
```

Make sure that the contents of the file have not changed.

```
student@bchd:~/static/training$ cat ego_fuel.txt

Alta3 Research has AWESOME labs!
Alta3 Research has AMAZING labs!
```

11. **history** - shows all of the commands performed in this shell session.

```
student@bchd:~/static/training$ history

    1  pwd
    2  ls
    3  cd static
    4  mkdir training
    5  ls
    6  cd training
    7  touch example_01.txt
    8  ls
    9  echo Alta3 Research Training rocks!
   10  echo Alta3 Research has AWESOME labs! > myfile.txt
   11  echo Alta3 Research has AMAZING labs! >> myfile.txt
   12  cat myfile.txt
   13  mv myfile.txt ego_fuel.txt
   14  ls
   15  cat ego_fuel.txt
   16  history
```

12. **man** - [manual] show the manual pages (aka documentation) for a given command. This is helpful for understanding any commands you may not feel comfortable with.

```
student@bchd:~/static/training$ man ls
```

To quit out of this view, press the keyboard key: `q`

13. Excellent work! Now, let's move back to the home directory before you start working on the next lab.

```
student@bchd:~/static/training$ cd ~
```

14. Your prompt should now be back to: `student@bchd:~$`
15. If you need more, Ubuntu (a Linux distribution) has a free Linux primer https://ubuntu.com/tutorials/command-line-for-beginners#1-overview

57. Ansible and Dell Glossary

Action - An action is a part of a task that specifies which of the modules to run and which arguments to pass to that module. Each task can have only one action, but it may also have other parameters.

Ad Hoc - Refers to running Ansible to perform some quick command, using /usr/bin/ansible, rather than the orchestration language, which is /usr/bin/ansible-playbook. An example of an ad hoc command might be rebooting 50 machines in your infrastructure. Anything you can do ad hoc can be accomplished by writing a playbook and playbooks can also glue lots of other operations together.

Async - Refers to a task that is configured to run in the background rather than waiting for completion. If you have a long process that would run longer than the SSH timeout, it would make sense to launch that task in async mode. Async modes can poll for completion every so many seconds or can be configured to "fire and forget", in which case Ansible will not even check on the task again; it will just kick it off and proceed to future steps. Async modes work with both /usr/bin/ansible and /usr/bin/ansible-playbook.

Callback Plugin - Refers to some user-written code that can intercept results from Ansible and do something with them. Some supplied examples in the GitHub project perform custom logging, send email, or even play sound effects.

Check Mode - Refers to running Ansible with the --check option, which does not make any changes on the remote systems, but only outputs the changes that might occur if the command ran without this flag. This is analogous to so-called "dry run" modes in other systems, though the user should be warned that this does not take into account unexpected command failures or cascade effects (which is true of similar modes in other systems). Use this to get an idea of what might happen, but do not substitute it for a good staging environment.

Connection Plugin - By default, Ansible talks to remote machines through pluggable libraries. Ansible uses native OpenSSH (SSH (Native)) or a Python implementation called paramiko. OpenSSH is preferred if you are using a recent version, and also enables some features like Kerberos and jump hosts. This is covered in the getting started section. There are also other connection types like accelerate mode, which must be bootstrapped over one of the SSH-based connection types but is very fast, and local mode, which acts on the local system. Users can also write their own connection plugins.

Conditionals - A conditional is an expression that evaluates to true or false that decides whether a given task is executed on a given machine or not. Ansible's conditionals are powered by the 'when' statement, which are discussed in the Working with playbooks.

Declarative - An approach to achieving a task that uses a description of the final state rather than a description of the sequence of steps necessary to achieve that state. For a real world example, a declarative specification of a task would be: "put me in California". Depending on your current location, the sequence of steps to get you to California may vary, and if you are already in California, nothing at all needs to be done. Ansible's Resources are declarative; it figures out the steps needed to achieve the final state. It also lets you know whether or not any steps needed to be taken to get to the final state.

Diff Mode - A --diff flag can be passed to Ansible to show what changed on modules that support it. You can combine it with --check to get a good 'dry run'. File diffs are normally in unified diff format.

Executor - A core software component of Ansible that is the power behind /usr/bin/ansible directly – and corresponds to the invocation of each task in a playbook. The Executor is something Ansible developers may talk about, but it's not really user land vocabulary.

Facts - Facts are simply things that are discovered about remote nodes. While they can be used in playbooks and templates just like variables, facts are things that are inferred, rather than set. Facts are automatically discovered by Ansible when running plays by executing the internal setup module on the remote nodes. You never have to call the setup module explicitly, it just runs, but it can be disabled to save time if it is not needed or you can tell ansible to collect only a subset of the full facts via the gather_subset: option. For the convenience of users who are switching from other configuration management systems, the fact module will also pull in facts from the ohai and facter tools if they are installed. These are fact libraries from Chef and Puppet, respectively. (These may also be disabled via gather_subset:)

Filter Plugin - A filter plugin is something that most users will never need to understand. These allow for the creation of new Jinja2 filters, which are more or less only of use to people who know what Jinja2 filters are. If you need them, you can learn how to write them in the API docs section.

Forks - Ansible talks to remote nodes in parallel and the level of parallelism can be set either by passing --forks or editing the default in a configuration file. The default is a very conservative five (5) forks, though if you have a lot of RAM, you can easily set this to a value like 50 for increased parallelism.

Gather Facts (Boolean) - Facts are mentioned above. Sometimes when running a multi-play playbook, it is desirable to have some plays that don't bother with fact computation if they aren't going to need to utilize any of these values. Setting gather_facts: False on a playbook allows this implicit fact gathering to be skipped.

Globbing - Globbing is a way to select lots of hosts based on wildcards, rather than the name of the host specifically, or the name of the group they are in. For instance, it is possible to select ww* to match all hosts starting with www. This concept is pulled directly from Func, one of Michael

DeHaan's (an Ansible Founder) earlier projects. In addition to basic globbing, various set operations are also possible, such as 'hosts in this group and not in another group', and so on.

Group - A group consists of several hosts assigned to a pool that can be conveniently targeted together, as well as given variables that they share in common.

Group Vars - The group_vars/ files are files that live in a directory alongside an inventory file, with an optional filename named after each group. This is a convenient place to put variables that are provided to a given group, especially complex data structures, so that these variables do not have to be embedded in the inventory file or playbook.

Handlers - Handlers are just like regular tasks in an Ansible playbook (see Tasks) but are only run if the Task contains a notify directive and also indicates that it changed something. For example, if a config file is changed, then the task referencing the config file templating operation may notify a service restart handler. This means services can be bounced only if they need to be restarted. Handlers can be used for things other than service restarts, but service restarts are the most common usage.

Host - A host is simply a remote machine that Ansible manages. They can have individual variables assigned to them, and can also be organized in groups. All hosts have a name they can be reached at (which is either an IP address or a domain name) and, optionally, a port number, if they are not to be accessed on the default SSH port.

Host Specifier - Each Play in Ansible maps a series of tasks (which define the role, purpose, or orders of a system) to a set of systems.

Host Vars - Just like Group Vars, a directory alongside the inventory file named host_vars/ can contain a file named after each hostname in the inventory file, in YAML format. This provides a convenient place to assign variables to the host without having to embed them in the inventory file. The Host Vars file can also be used to define complex data structures that can't be represented in the inventory file.

Idempotency - An operation is idempotent if the result of performing it once is exactly the same as the result of performing it repeatedly without any intervening actions.

Includes - The idea that playbook files (which are nothing more than lists of plays) can include other lists of plays, and task lists can externalize lists of tasks in other files, and similarly with handlers. Includes can be parameterized, which means that the loaded file can pass variables. For instance, an included play for setting up a WordPress blog may take a parameter called user and that play could be included more than once to create a blog for both alice and bob.

Inventory - A file (by default, Ansible uses a simple INI format) that describes Hosts and Groups in Ansible. Inventory can also be provided via an Inventory Script (sometimes called an "External Inventory Script").

Inventory Script - A very simple program (or a complicated one) that looks up hosts, group membership for hosts, and variable information from an external resource – whether that be a SQL database, a CMDB solution, or something like LDAP. This concept was adapted from Puppet (where it is called an "External Nodes Classifier") and works more or less exactly the same way.

Isilon - A scale out network-attached storage platform offered by Dell EMC for high-volume storage, backup and archiving of unstructured data. It provides a cluster-based storage array based on industry standard hardware, and is scalable to 50 petabytes in a single filesystem using its FreeBSD-derived OneFS file system.

Jinja2 - Jinja2 is the preferred templating language of Ansible's template module. It is a very simple Python template language that is generally readable and easy to write.

JSON - Ansible uses JSON for return data from remote modules. This allows modules to be written in any language, not just Python.

Lazy Evaluation - In general, Ansible evaluates any variables in playbook content at the last possible second, which means that if you define a data structure that data structure itself can define variable values within it, and everything "just works" as you would expect. This also means variable strings can include other variables inside of those strings.

Library - A collection of modules made available to /usr/bin/ansible or an Ansible playbook.

Limit Groups - By passing --limit somegroup to ansible or ansible-playbook, the commands can be limited to a subset of hosts. For instance, this can be used to run a playbook that normally targets an entire set of servers to one particular server.

Local Action - A local_action directive in a playbook targeting remote machines means that the given step will actually occur on the local machine, but that the variable {{ ansible_hostname }} can be passed in to reference the remote hostname being referred to in that step. This can be used to trigger, for example, an rsync operation.

Local Connection - By using connection: local in a playbook, or passing -c local to /usr/bin/ansible, this indicates that we are managing the local host and not a remote machine.

Lookup Plugin - A lookup plugin is a way to get data into Ansible from the outside world. Lookup plugins are an extension of Jinja2 and can be accessed in templates, for example, {{ lookup('file','/path/to/file') }}. These are how such things as with_items, are implemented. There are also lookup plugins like file which loads data from a file and ones for querying environment variables, DNS text records, or key value stores.

Loops - Generally, Ansible is not a programming language. It prefers to be more declarative, though various constructs like loop allow a particular task to be repeated for multiple items in a list. Certain modules, like yum and apt, actually take lists directly, and can install all packages given in those lists within a single transaction, dramatically speeding up total time to configuration, so they can be used without loops.

Modules - Modules are the units of work that Ansible ships out to remote machines. Modules are kicked off by either /usr/bin/ansible or /usr/bin/ansible-playbook (where multiple tasks use lots of different modules in conjunction). Modules can be implemented in any language, including Perl, Bash, or Ruby – but can leverage some useful communal library code if written in Python. Modules just have to return JSON. Once modules are executed on remote machines, they are removed, so no long running daemons are used. Ansible refers to the collection of available modules as a library.

Multi-Tier - The concept that IT systems are not managed one system at a time, but by interactions between multiple systems and groups of systems in well defined orders. For instance, a web server may need to be updated before a database server and pieces on the web server may need to be updated after THAT database server and various load balancers and monitoring servers may need to be contacted. Ansible models entire IT topologies and workflows rather than looking at configuration from a “one system at a time” perspective.

Notify - The act of a task registering a change event and informing a handler task that another action needs to be run at the end of the play. If a handler is notified by multiple tasks, it will still be run only once. Handlers are run in the order they are listed, not in the order that they are notified.

Orchestration - Many software automation systems use this word to mean different things. Ansible uses it as a conductor would conduct an orchestra. A datacenter or cloud architecture is full of many systems, playing many parts – web servers, database servers, maybe load balancers, monitoring systems, continuous integration systems, and so on. In performing any process, it is necessary to touch systems in particular orders, often to simulate rolling updates or to deploy software correctly. Some system may perform some steps, then others, then previous systems already processed may need to perform more steps. Along the way, emails may need to be sent or web services contacted. Ansible orchestration is all about modeling that kind of process.

paramiko - By default, Ansible manages machines over SSH. The library that Ansible uses by default to do this is a Python-powered library called paramiko. The paramiko library is generally fast and easy to manage, though users who want to use Kerberos or Jump Hosts may wish to switch to a native SSH binary such as OpenSSH by specifying the connection type in their playbooks, or using the -c ssh flag.

Playbooks - Playbooks are the language by which Ansible orchestrates, configures, administers, or deploys systems. They are called playbooks partially because it’s a sports analogy, and it’s supposed to be fun using them. They aren’t workbooks :)

Plays - A playbook is a list of plays. A play is minimally a mapping between a set of hosts selected by a host specifier (usually chosen by groups but sometimes by hostname globs) and the tasks which run on those hosts to define the role that those systems will perform. There can be one or many plays in a playbook.

Pull Mode - By default, Ansible runs in push mode, which allows it very fine-grained control over when it talks to each system. Pull mode is provided for when you would rather have nodes check in every N minutes on a particular schedule. It uses a program called ansible-pull and can also be set up (or reconfigured) using a push-mode playbook. Most Ansible users use push mode, but pull mode is included for variety and the sake of having choices. ansible-pull works by checking configuration orders out of git on a crontab and then managing the machine locally, using the local connection plugin.

Push Mode - Push mode is the default mode of Ansible. In fact, it’s not really a mode at all – it’s just how Ansible works when you aren’t thinking about it. Push mode allows Ansible to be fine-grained and conduct nodes through complex orchestration processes without waiting for them to check in.

Register Variable - The result of running any task in Ansible can be stored in a variable for use in a template or a conditional statement. The keyword used to define the variable is called register, taking its name from the idea of registers in assembly programming (though Ansible will never feel like assembly programming). There are an infinite number of variable names you can use for registration.

Resource Model - Ansible modules work in terms of resources. For instance, the file module will select a particular file and ensure that the attributes of that resource match a particular model. As an example, we might wish to change the owner of /etc/motd to root if it is not already set to root, or set its mode to 0644 if it is not already set to 0644. The resource models are idempotent meaning change commands are not run unless needed, and Ansible will bring the system back to a desired state regardless of the actual state – rather than you having to tell it how to get to the state.

Roles - Roles are units of organization in Ansible. Assigning a role to a group of hosts (or a set of groups, or host patterns, and so on) implies that they should implement a specific behavior. A role may include applying certain variable values, certain tasks, and certain handlers – or just one or more of these things. Because of the file structure associated with a role, roles become redistributable units that allow you to share behavior among playbooks – or even with other users.

Rolling Update - The act of addressing a number of nodes in a group N at a time to avoid updating them all at once and bringing the system offline. For instance, in a web topology of 500 nodes handling very large volume, it may be reasonable to update 10 or 20 machines at a time, moving on to the next 10 or 20 when done. The serial: keyword in an Ansible playbooks control the size of the rolling update pool. The default is to address the batch size all at once, so this is something that you must opt-in to. OS configuration (such as making sure config files are correct) does not typically have to use the rolling update model, but can do so if desired.

Sudo - Ansible does not require root logins, and since it's daemonless, definitely does not require root level daemons (which can be a security concern in sensitive environments). Ansible can log in and perform many operations wrapped in a sudo command, and can work with both password-less and password-based sudo. Some operations that don't normally work with sudo (like scp file transfer) can be achieved with Ansible's copy, template, and fetch modules while running in sudo mode.

SSH (Native) - **Native OpenSSH as an Ansible transport is specified with** -c ssh (or a config file, or a directive in the playbook) and can be useful if wanting to login via Kerberized SSH or using SSH jump hosts, and so on. In 1.2.1, ssh will be used by default if the OpenSSH binary on the control machine is sufficiently new. Previously, Ansible selected paramiko as a default. Using a client that supports ControlMaster and ControlPersist is recommended for maximum performance – if you don't have that and don't need Kerberos, jump hosts, or other features, paramiko is a good choice. Ansible will warn you if it doesn't detect ControlMaster/ControlPersist capability.

Tags - Ansible allows tagging resources in a playbook with arbitrary keywords, and then running only the parts of the playbook that correspond to those keywords. For instance, it is possible to have an entire OS configuration, and have certain steps labeled ntp, and then run just the ntp steps to reconfigure the time server information on a remote host.

Task - Playbooks exist to run tasks. Tasks combine an action (a module and its arguments) with a name and optionally some other keywords (like looping directives). Handlers are also tasks, but they are a special kind of task that do not run unless they are notified by name when a task reports an underlying change on a remote system.

Tasks - A list of Task.

Templates - Ansible can easily transfer files to remote systems but often it is desirable to substitute variables in other files. Variables may come from the inventory file, Host Vars, Group Vars, or Facts. Templates use the Jinja2 template engine and can also include logical constructs like loops and if statements.

Transport - Ansible uses :term:Connection Plugins to define types of available transports. These are simply how Ansible will reach out to managed systems. Transports included are paramiko, ssh (using OpenSSH), and local.

When - An optional conditional statement attached to a task that is used to determine if the task should run or not. If the expression following the when: keyword evaluates to false, the task will be ignored.

Vars (Variables) - As opposed to Facts, variables are names of values (they can be simple scalar values – integers, booleans, strings) or complex ones (dictionaries/hashes, lists) that can be used in templates and playbooks. They are declared things, not things that are inferred from the remote system's current state or nature (which is what Facts are).

YAML - Ansible does not want to force people to write programming language code to automate infrastructure, so Ansible uses YAML to define playbook configuration languages and also variable files. YAML is nice because it has a minimum of syntax and is very clean and easy for people to skim. It is a good data format for configuration files and humans, but also machine readable. Ansible's usage of YAML stemmed from Michael DeHaan's first use of it inside of Cobbler around 2006. YAML is fairly popular in the dynamic language community and the format has libraries available for serialization in many languages (Python, Perl, Ruby, and so on).