

# IGNW

NetOps Workshop



## Table of Contents

Table of Contents	2
Getting Started	7
Lab Topology	7
Lab IP and Credential Information	8
How to Access the Lab	8
Version Control Introduction with Git	13
Overview and Objectives	13
Introduction	13
Sign up for a Github Account	13
Add SSH keys	14
Create a Repo(sitory)	22
Your First Commit	23
Time to Push	26
Python Hello World	30
Overview and Objectives	30
Hello World and the Interactive Interpreter	30
Access the Interactive Interpreter	30
Variables and Printing	33
Learn about Types	36
Manipulating Strings	38
String Formatting	38
Working with Lists	47
Accessing Elements in a List	47
Appending Elements to a List	51
Deleting List Elements	52
Working with Nested Lists	53
Slicing	59
Working with Dicts	60
Creating a Dictionary	61
Accessing a Dictionary	62
What else can go in Dictionaries?	63
Accessing nested Dictionaries	64
Conditionals	65
Introducing the "if" Statement	65
Elif	68
Else	69
While Loops	70
Build a Countdown Timer	70
For Loops	72
Iterate over lists	72
Importing Modules	73
Import a Module	74

Capturing User Input	75
Input	75
CheckiO Labs	77
Overview and Objectives	77
Introduction	77
Three Words	86
Fizz Buzz	92
Additional Labs	92
Secret Message	93
Right to Left	93
Index Power	93
Bonus Labs	93
House Password	93
Most Wanted Letter	93
Python for Network and Systems Engineers Labs	94
Overview and Objectives	94
Introducing Pexpect	94
Logging into a Router with pexpect	94
Get the Hostname	100
Capture Relevant Interface Configuration	100
Introducing Netmiko	107
Logging into a Router with Netmiko	107
Get the Hostname	109
Capture Relevant Interface Configuration	110
Netmiko Basic Configuration	112
Configure a new loopback	112
Validate Interface is "up/up"	113
Config and Validate (CSR, ASA, and NXOSv)	114
CSR Configurations	114
Create another Loopback	114
Configure Interface	116
Create a Host Route	118
ASA Configuration	121
Configure Interfaces	122
Configure a Host Route	127
Configure and Apply ACL	129
Configuring NXOS Interface and Route	132
Validate Success!	135
Bonus 1	136
Bonus 2	136
Getting Started APIs with Postman	137
Overview and Objectives	137
APIs and Authentication	137
HTTP Basic Authentication	137

API Key / Token	137
OAuth	137
Getting Familiar with Postman	138
Authenticating to the ASA	146
Learning how to Navigate the API	155
GETting some (useful?) Data	163
Configuration Time!	168
GET	169
PUT	169
PATCH	169
POST	169
Magical cURL/Python Output	184
Python and APIs	189
Overview and Objectives	189
Introduction	189
Introducing the Requests Module	189
Authenticating to the ASA	189
Requests PUT? POST? PATCH?	201
Build all the things!	210
Kicking the tires on NETCONF & what the heck is YANG?	211
RESTCONF, back to requests	224
RESTCONF POST a new Loopback	229
Ansible for Network Engineer Labs	231
Overview and Objectives	231
Gathering Facts	231
Conditionals	241
Working with Lists	247
Working with Dictionaries	252
Ansible Setup	256
Validate Ansible Configuration	262
Host and Group Vars -- Dealing with more Variables	266
Beyond Facts, the "ios_command" Module	271
Finally, Configurations!	273
IOS_Config - Before and Match	278
Handlers - Saving your Work!	288
Setting up Jenkins	291
Overview and Objectives	291
Network Delivery Lifecycle Overview	291
Jenkins Installation	291
Initial Log In	292
Plugins	297
Creating your first Pipeline	303
Overview and Objectives	303
What are Pipelines?	303

Create a Pipeline	305
Connecting to GitHub	310
First Build	315
The Jenkinsfile	318
Build Output	324
Network Configurations in Git	332
Overview and Objectives	332
Network Delivery Lifecycle Overview	332
Managing Configurations in General	332
Getting the Configs in Git	333
Now what?	338
Getting Started with NAPALM	339
Merge Candidate and Diffs	342
Templates, Diffs, and Idempotency	344
Templatizing Configurations with Ansible	346
Overview and Objectives	346
Network Delivery Lifecycle Overview	346
Why and how to Template Configurations	346
Getting Started	347
First Template	351
Generating Templates	356
Host and Group Variables	362
Logic in Jinja2	369
Jinaj2 Include	380
ASA and NX-OS Templates	383
Deploying Configurations Programmatically	393
Overview and Objectives	393
Network Delivery Lifecycle Overview	393
Intro to NAPALM Ansible	393
Setting up NAPALM Ansible	394
Playbook time -- Getting logged in	395
NAPALM Diffs in Ansible	399
Committing Changes	401
Moving on to NX-OS	404
ASA NAPALM Woes	407
Group and Host Variables	413
Jenkins and IaC	421
Overview and Objectives	421
Network Delivery Lifecycle Overview	421
Getting ready for Jenkins	421
Enter Jenkins (again, but for real this time!)	426
Jenkins Build Environment	433
Running the Playbooks	441
Requirements, requirements, requirements	452

Testing in the Pipeline	457
Overview and Objectives	457
Network Delivery Lifecycle Overview	457
What are we Testing?	457
Ansible syntax-check	458
Unit Testing in Action	461
Functional Testing	466
Testing for Failures (on purpose!)	474
Rollback Functionality	485
Overview and Objectives	485
Network Delivery Lifecycle Overview	485
Rollback Time	485
Capture Backup/Baseline Configurations	486
What's in a Name?	489
NX-OS Checkpoints	492
Pushing Rollback Configs Out	495
Ansible and Secrets	496
NX-OS Checkpoint Rollback	502
ASA Checkpoint? No... Config Replace? No..	504
Push to Git	510
When do we Rollback?	511
Environment Updates	514
Testing Rollbacks	517
Promoting to Production	523
Overview and Objectives	523
Network Delivery Lifecycle Overview	523
Promotion Time!	523
Inventory Part 2	524
Updating the Jenkinsfile	525
Features Features Features	528
Setting up the Archive Framework	534
Refactoring	540
Overview and Objectives	540
Network Delivery Lifecycle Overview	540
Containerize the Build Environment	540
Overhauling the Jenkinsfile	550
Initial Testing Jenkins and Docker	553
Re-establishing our Pipeline	556

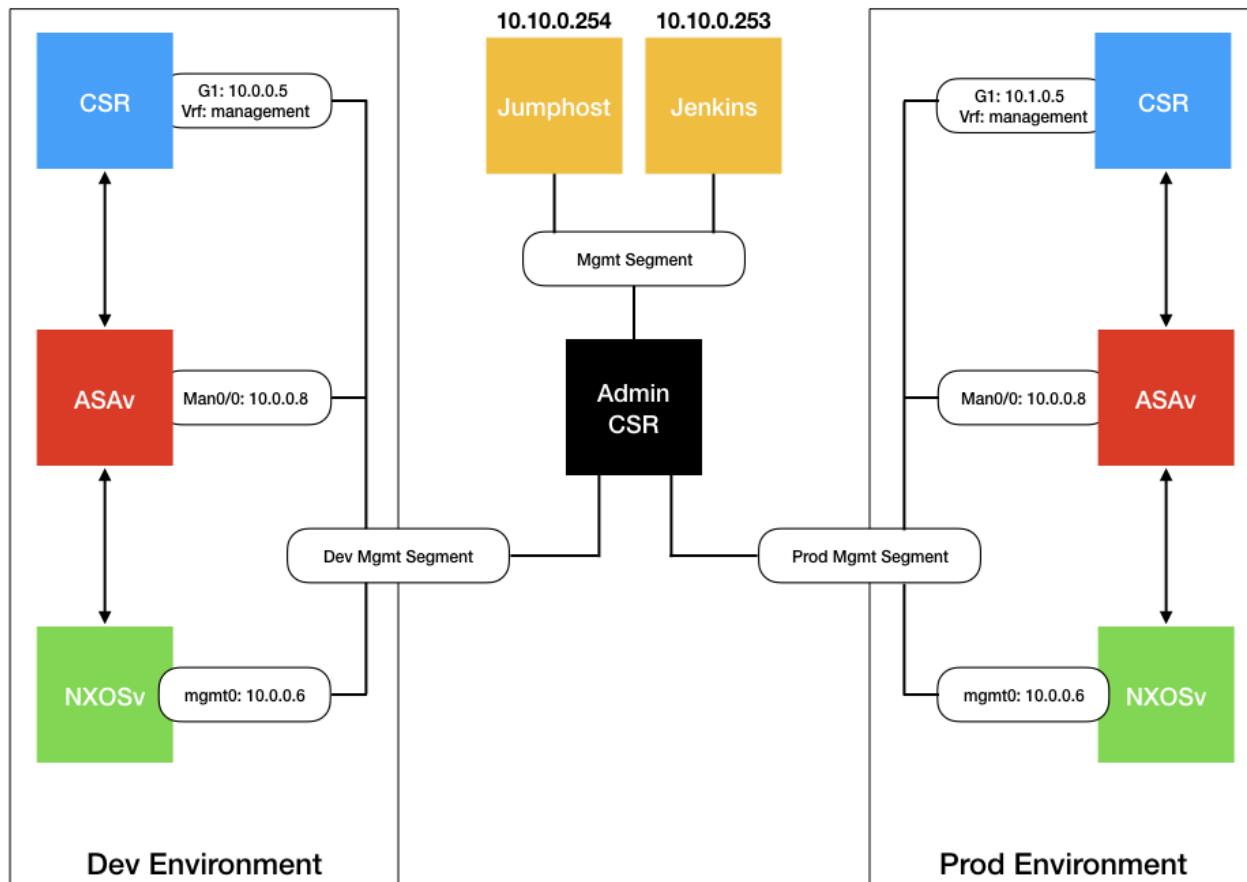
# Getting Started

## Lab Topology

This is a relatively small and simple topology built in a private cloud environment. Depending on the labs you complete you may interact with a subset of, or all of the devices. An Ubuntu Desktop instance is the primary point of entry to the lab -- students will primarily use this host for access to all other devices. This host is exposed via RDP and SSH on non-standard port numbers (assigned per student) on a single public IP address.

Within the lab itself there are two environments -- a "dev" and "prod" environment. These environments are exact mirrors other than the out of band management IP addressing. Each environment contains a Cisco Cloud Services Router (CSR), Cisco Adaptive Security Virtual Appliance (ASAv), and a Nexus 9000v switch. These devices are intended to be a small example of an edge router, edge firewall, and core switch.

There is also another Ubuntu host (server) in the lab that is running Jenkins. This host is exposed via HTTP (for Jenkins web UI) and SSH on non-standard port numbers (assigned per student) on a single public IP address.



## Lab IP and Credential Information

The following table outlines public and private IP addressing within the lab environment.

Device	IP Address	Misc./Notes
Jumphost	207.162.210.85	Public IP Address
Jumphost	10.10.0.254	Internal IP Address
Jenkins	207.162.210.85	Public IP Address
Jenkins	10.10.0.253	Internal IP Address
Dev CSR	10.0.0.5	Interface G1
Dev ASA v	10.0.0.8	Interface Man0/0
Dev NXOS	10.0.0.6	mgmt0
Prod CSR	10.1.0.5	Interface G1
Prod ASA v	10.1.0.8	Interface Man0/0
Prod NXOS	10.1.0.6	mgmt0

Per student Jumphost/Jenkins Access, where XX is student number:

Device	IP Address	Port
Jumphost - SSH	207.162.210.85	220XX
Jumphost - RDP	207.162.210.85	338XX
Jenkins - SSH	207.162.210.85	221XX
Jenkins - HTTP	207.162.210.85	80XX

The following table outlines credentials for lab access.

Device	Access Type	Username	Password(s)
Jumphost	Remote Desktop/SSH	ignw	ignw
Jenkins	SSH	ignw	ignw
CSR (dev & prod)	SSH	ignw	ignw
ASA v (dev & prod)	SSH	ignw	ignw
NXOS (dev & prod)	SSH	ignw	ignw

**Note:** Access all network devices *from* the Jumphost.

## How to Access the Lab

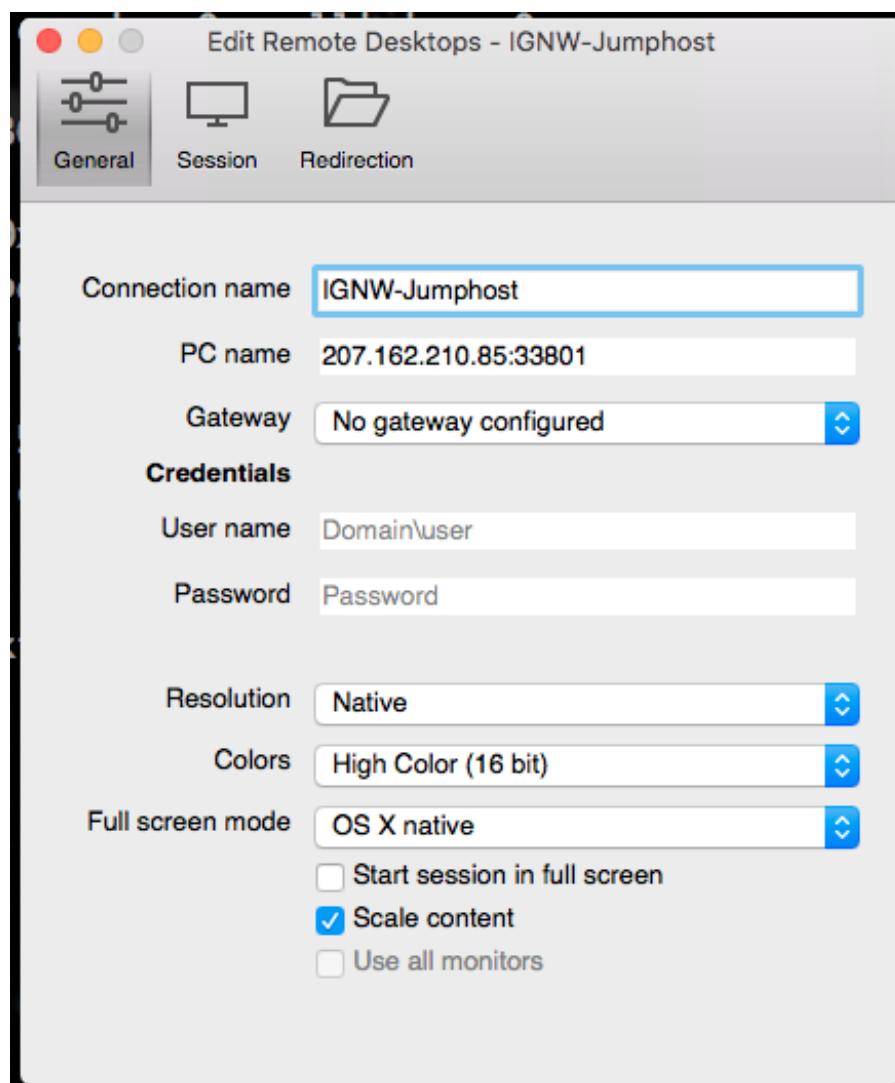
All access to the lab is via RDP to the Jump Host. The Jump Host is an Ubuntu Desktop node running XRD<sup>P</sup> to allow for standard RDP access.

Windows users very likely already have the native Windows RDP client installed -- you can verify this by pressing the Windows key and typing "mstc" -- the search should bring up the Windows Remote Desktop client. If you do not have the client for some reason you can download it from the Microsoft store by clicking [here](#).

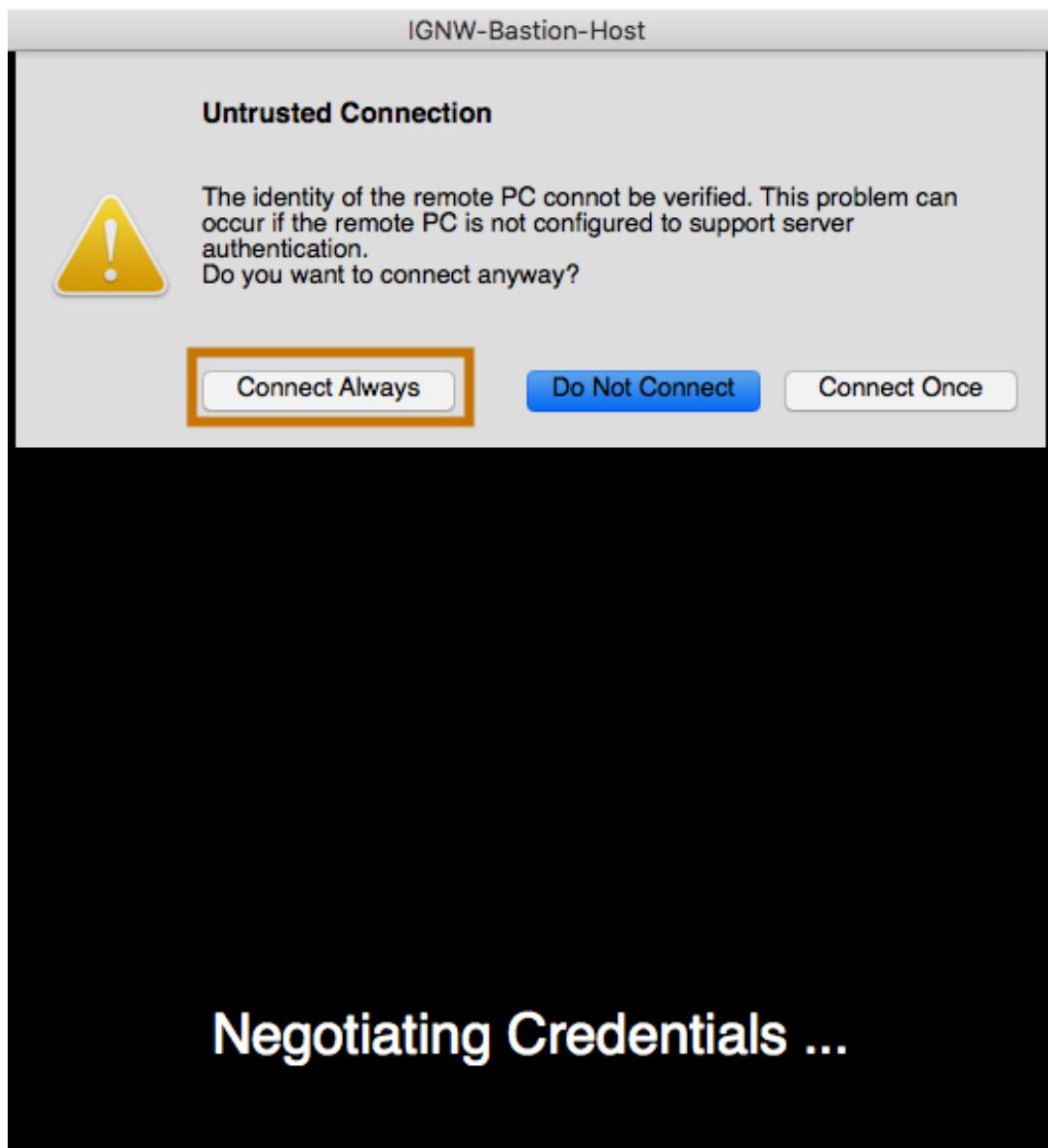
MacOS users should install the Microsoft RDP client found in the AppStore. If you do not have it already, you can download it by clicking [here](#).

Linux users should install a standard remote desktop client of their choosing. For many the Remmina Remote Desktop client is a good choice!

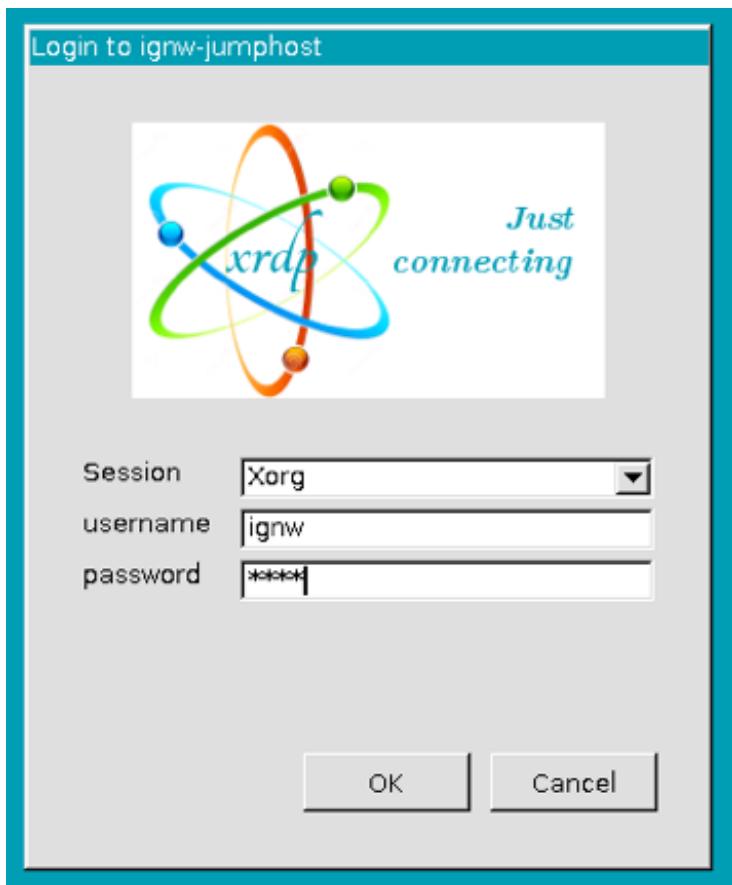
To connect to the lab environment, simply connect to the instructor provided IP address for the Bastion Host on the standard RDP port (3389), and authenticate with the provided credentials.



If prompted about an "Untrusted Connection" click the "Connect Always" button.



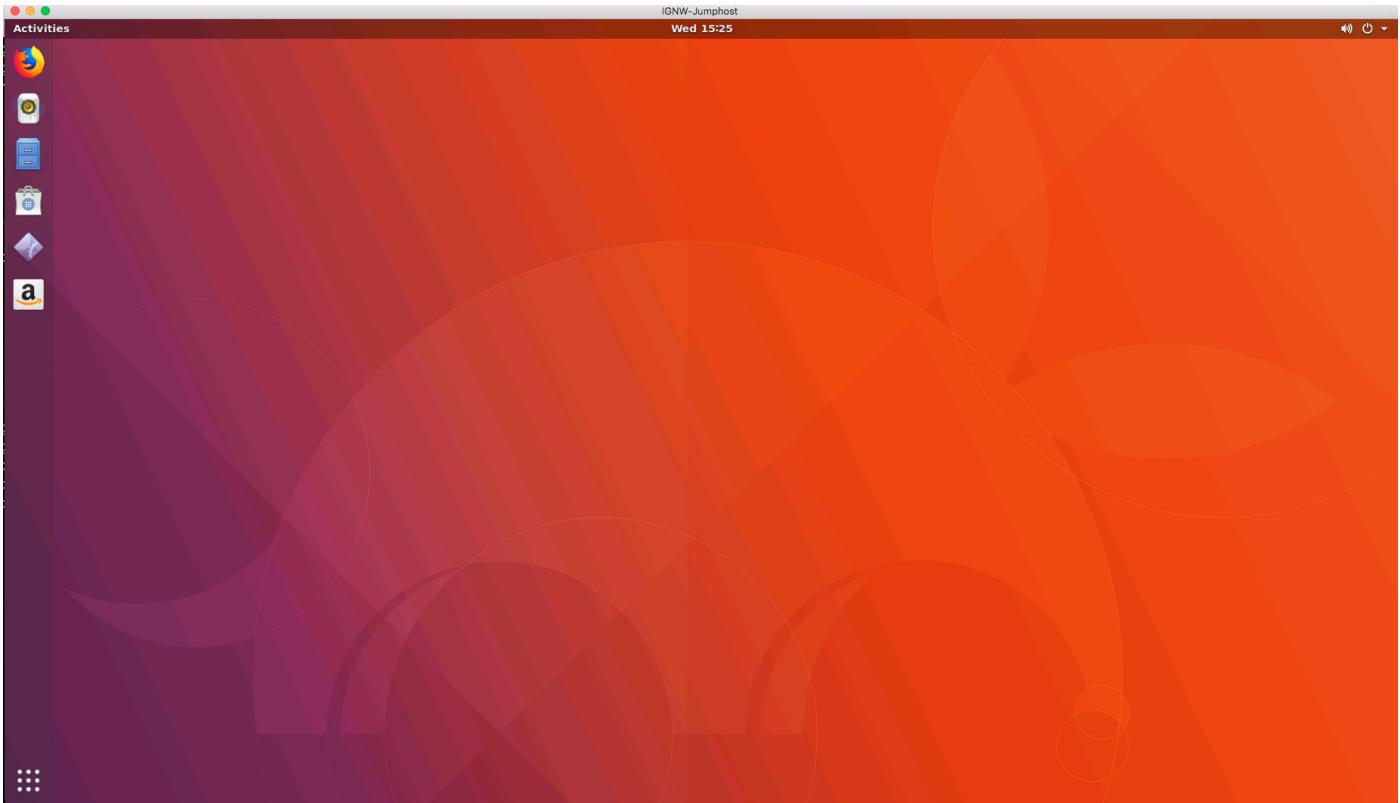
Authenticate at the XRDP prompt:



Unlock the Ubuntu host:



And you're all set!



From this point on you can access the terminal to connect to the other devices via SSH. You will use several other applications throughout this course, instructions on how to access the other applications will be provided when needed.

If you wish to SSH to the Ubuntu host instead, you can do that from the SSH client of your choosing by specifying the appropriate port:

```
1. ignw@ignw-jumphost: ~ (ssh)
x ...work_as_code (bash) 961 x ...work_as_code (bash) 962 x ...w-jumphost: ~ (ssh) 963
Last login: Wed May 23 08:18:07 on ttys000
Touchie:~ carl$ ssh ignw@207.162.210.85 -p 22001
The authenticity of host '[207.162.210.85]:22001' ([207.162.210.85]:22001)' can't be established.
ECDSA key fingerprint is SHA256:BTp20GFE54EeN57IJI4tldcSH35FuA41L0BqKFot978.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[207.162.210.85]:22001' (ECDSA) to the list of known hosts.
ignw@207.162.210.85's password:
Welcome to Ubuntu 17.10 (GNU/Linux 4.13.0-41-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

12 packages can be updated.
11 updates are security updates.

New release '18.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed May 23 14:30:38 2018 from 10.1.2.109
ignw@ignw-jumphost:~$
```

# Version Control Introduction with Git

## Overview and Objectives

In this lab you will learn the basics of working with Git.

Objectives:

- Setup your GitHub account if you don't already have one
- Setup and test SSH keys
- Create and initialize a repository
- Push changes up to GitHub

## Introduction

Version control is pretty much exactly what it sounds like and more. At the simplest, version control is about being able to track changes in a project, and have the flexibility to rollback to a previous iteration if necessary.

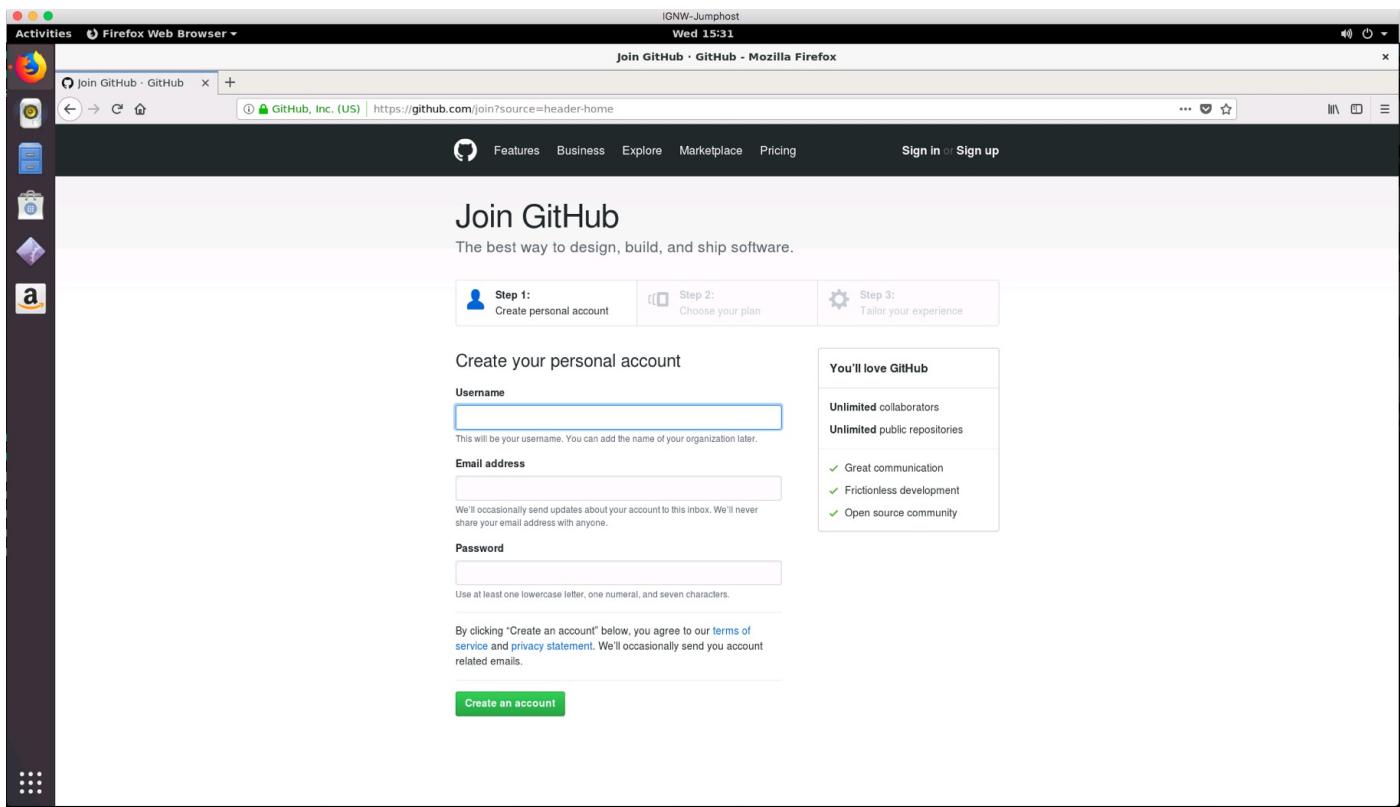
For an individual user, a developer, or a user who just wants to keep track of changes in some file be it a switch configuration or an Ansible playbook to manage servers, maintaining versions and tracking revisions may be all that is needed from a version control system. For a team collaborating on something (again, it can be anything, switch configurations, Ansible Playbooks, documentation, or some homework assignment!), version control plays an even more critical role!

Version control systems allow for people to collaborate on a project where users may be editing the same file at the same exact time without any issues! How? When these users go to check in their work to the version control system the changes each user made will be outlined allowing for work to be *merged* together without overwriting what one another has done.

You've probably heard of Github as well; Git and Github are **not** the same thing, though it would be easy to be confused! Github is a hosting service for Git repositories, and Git is a free, open source, version control system. In this task we'll use Git and Github together to see what all the fuss is about.

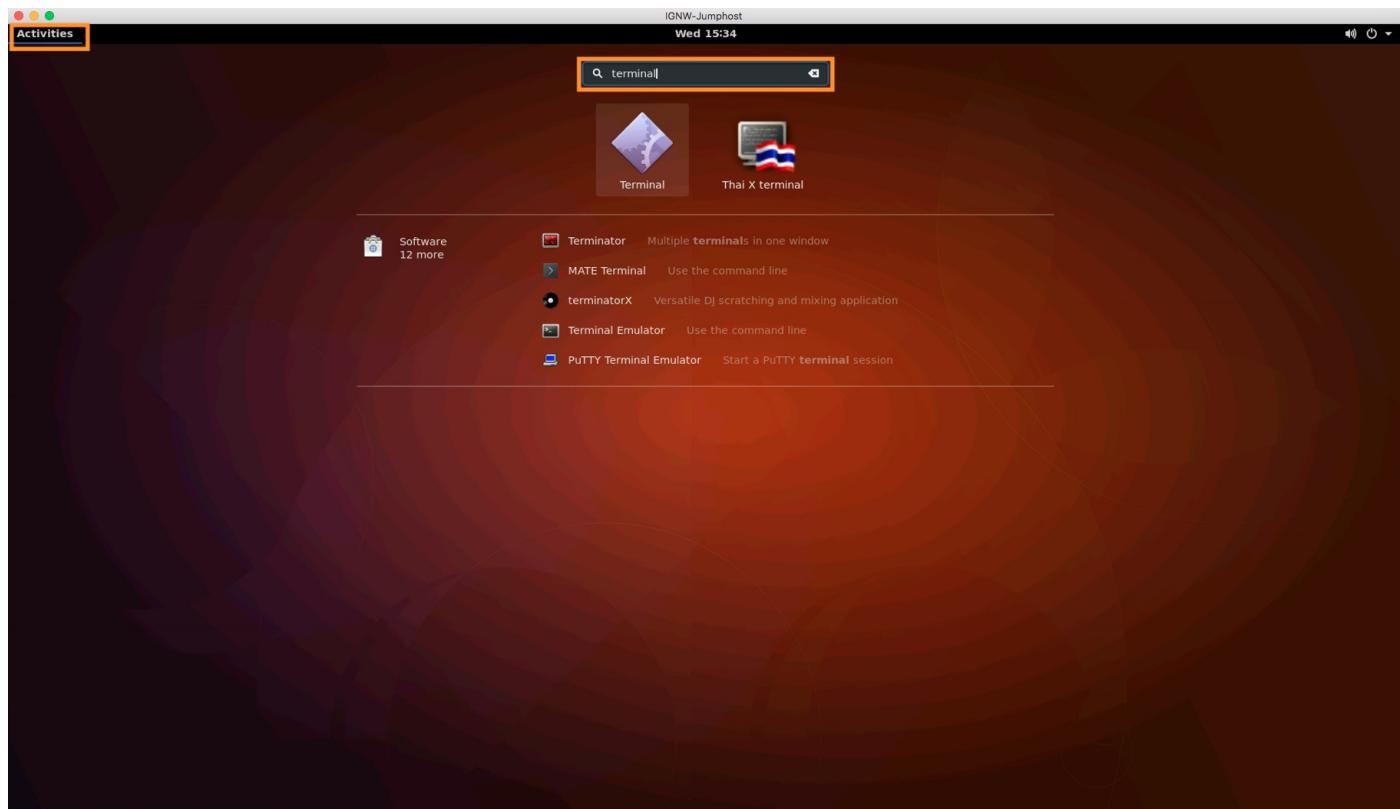
## Sign up for a Github Account

If you don't already have a Github account you will need to head over to [Github](#) and get signed up.



## Add SSH keys

Git has already been installed for you on the Ubuntu Desktop instance. You can verify this from the command line. Open the Terminal application by clicking on the "activities" button on the top left, then typing "terminal" in the search form:



In the terminal window, type "git --version" -- this will validate that Git is installed and functional and of course let us know what version is on the system.



A screenshot of a terminal window titled "ignw@ignw-jumphost: ~". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". Below the menu is a command prompt: "ignw@ignw-jumphost:~\$ git --version". The output of the command is "git version 2.14.1". The window has scroll bars on the right side.

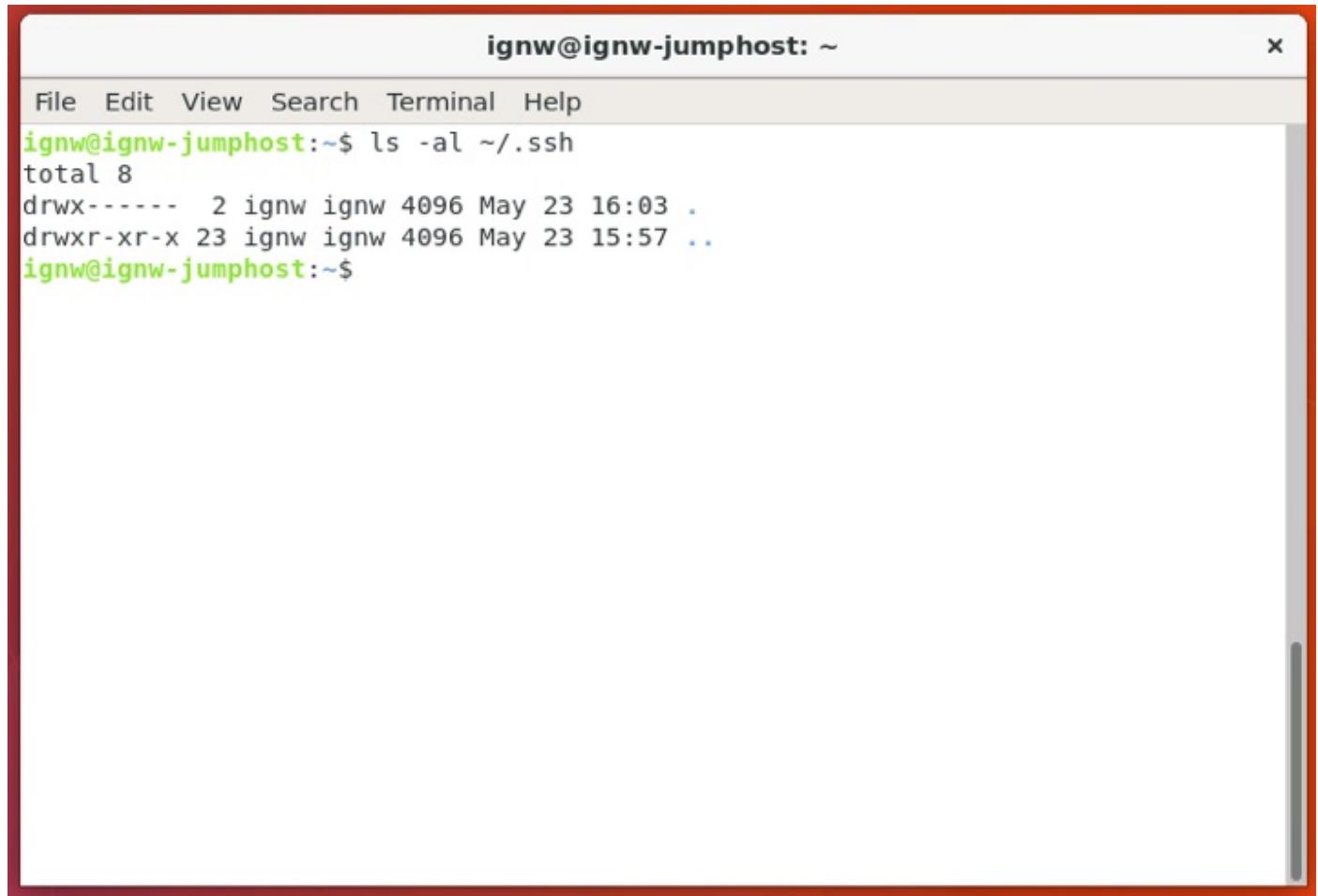
```
ignw@ignw-jumphost:~$ git --version
git version 2.14.1
ignw@ignw-jumphost:~$
```

Now that you have an account and have verified that Git is installed, you need to connect your command line shell to GitHub account for authorizing changes. You can find the instructions [here](#) or follow the steps below.

First, check to see if you have a public SSH key available on the system.

```
ls -al ~/.ssh
```

This command will list all of the keys in the .ssh directory or tell you that the directory doesn't exist yet.



A screenshot of a terminal window titled "ignw@ignw-jumphost: ~". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area shows the command "ls -al ~/.ssh" being run, with the output:

```
ignw@ignw-jumphost:~$ ls -al ~/.ssh
total 8
drwx----- 2 ignw ignw 4096 May 23 16:03 .
drwxr-xr-x 23 ignw ignw 4096 May 23 15:57 ..
ignw@ignw-jumphost:~$
```

We are specifically looking for our public key so that we can provide that to Github to prove our identity. Since we don't have one, we will need to create it and associate it to the email we used for our GitHub account:

```
ssh-keygen -t rsa -C "your_email@example.com"
```

Accept all of the defaults when prompted. You now have an SSH key!

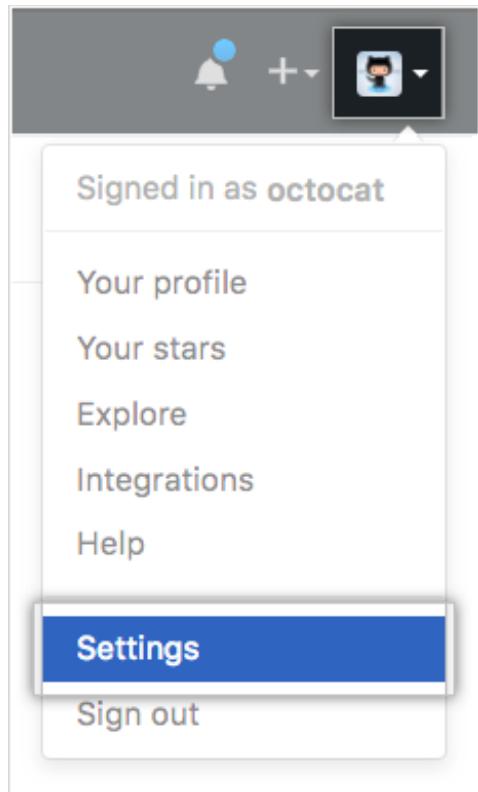
```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ ls -al ~/.ssh
total 8
drwx----- 2 ignw ignw 4096 May 23 16:03 .
drwxr-xr-x 23 ignw ignw 4096 May 23 15:57 ..
ignw@ignw-jumphost:~$ ssh-keygen -t rsa -C "carln@ignw.io"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ignw/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ignw/.ssh/id_rsa.
Your public key has been saved in /home/ignw/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:rP4P+x+MkViPU0xuSCGZDaD1l7v2E8+Zj819yhozvGA carln@ignw.io
The key's randomart image is:
+---[RSA 2048]---+
|   0.0=0.. |
|   0 .00.* |
|   . . = = |
|   . + B   |
|   S * .   |
|   .   B.   |
|   . . E B+ o |
|   . = o.B=+o |
|   ..000.=o+o*|
+---[SHA256]---+
ignw@ignw-jumphost:~$ ls -al ~/.ssh
total 16
drwx----- 2 ignw ignw 4096 May 23 16:03 .
drwxr-xr-x 23 ignw ignw 4096 May 23 15:57 ..
-rw-----  1 ignw ignw 1679 May 23 16:03 id_rsa
-rw-r--r--  1 ignw ignw  395 May 23 16:03 id_rsa.pub
ignw@ignw-jumphost:~$
```

Print the contents of the key to the terminal so you can copy it:

```
cat ~/.ssh/id_rsa.pub
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
Enter file in which to save the key (/home/ignw/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ignw/.ssh/id_rsa.
Your public key has been saved in /home/ignw/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:rP4P+x+MkViPU0xuSCGZDaD1l7v2E8+Zj819yhozvGA carln@ignw.io
The key's randomart image is:
+---[RSA 2048]----+
|   o.o=.. |
|   o .oo.* |
|   . . =   |
|   . + B   |
|   S * .   |
|   .   B.   |
|   . . E B+ o |
|   . = o.B=+o |
|   ..ooo.=o+o*|
+---[SHA256]----+
ignw@ignw-jumphost:~$ ls -al ~/.ssh
total 16
drwx----- 2 ignw ignw 4096 May 23 16:03 .
drwxr-xr-x 23 ignw ignw 4096 May 23 15:57 ..
-rw----- 1 ignw ignw 1679 May 23 16:03 id_rsa
-rw-r--r-- 1 ignw ignw 395 May 23 16:03 id_rsa.pub
ignw@ignw-jumphost:~$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQDFgXZrAhA4V4+5VzT65xMf4kNAV1t7ZwlG9kpYBCLjH4o0erwyDEdazl1
2+kRsM5JLVX1lDQggF9mTolxB2CbXBtheD9N2YxZ++Z5o7afLcoD5tIVXy27/Td0m+G2Y8rIo//5FJ4r3NHmHXnQdxIMtl
0v/aLterFPhw3zcOUUKU++L4Y7EfM30L0kBIBvAL6iirl0pVePo+MLX7C1gij615dD9fFnNSVjd16uvjI9eB02PP8k+JB+0
azQdxc2z9qDb0v0Q2sceHo213tEGZrL95cv/GUS7couLQk/InblsNL0As0+NRQpKmPqQnliEERE2uffFleRvkMfgdC14mjV9
carln@ignw.io
ignw@ignw-jumphost:~$
```

Copy the output -- make sure you snag the final "Student@IGNW" section as well. Head over to your Github account -- on the top right of the web page, click on your profile, then click the "Settings" option.



Then from the menu on the left click on "SSH and GPG keys". Click the "New SSH key" button.

The screenshot shows the GitHub 'Personal settings' page. On the left is a sidebar with links: Personal settings (selected), Profile, Account, Emails, Notifications, Billing, SSH and GPG keys (highlighted with an orange box), Security, Blocked users, Repositories, Organizations, Saved replies, Applications, and Developer settings. The main content area has two sections: 'SSH keys' and 'GPG keys'. Both sections state 'There are no [key type] keys associated with your account.' Below each section is a link to a guide: 'generating SSH keys' or 'common SSH Problems' for SSH, and 'generate a GPG key' for GPG. A green 'New SSH key' button is located at the top right of the 'SSH keys' section, and a green 'New GPG key' button is located at the top right of the 'GPG keys' section.

Provide a title for your key (this can be whatever you'd like), and paste in the contents of your clipboard. Click "Add SSH key". If all went well, your screen should look similar to that shown below.

The screenshot shows the 'SSH keys' section of the GitHub personal settings. It displays a single key entry for 'IGNW-Lab'. The entry includes a key icon, the title 'IGNW-Lab', the fingerprint 'Fingerprint: c8:82:e6:26:47:8b:64:d9:f4:bf:27:b4:ec:02:2a:a9', the date 'Added on Mar 23, 2018', and the status 'Never used — Read/write'. To the right of the key details is a 'Delete' button. A green 'New SSH key' button is located at the top right of the section. Below the key list is a note linking to guides for generating SSH keys and troubleshooting common SSH problems.

We can now test our connection to make sure we are good to go:

```
ssh -T git@github.com
```



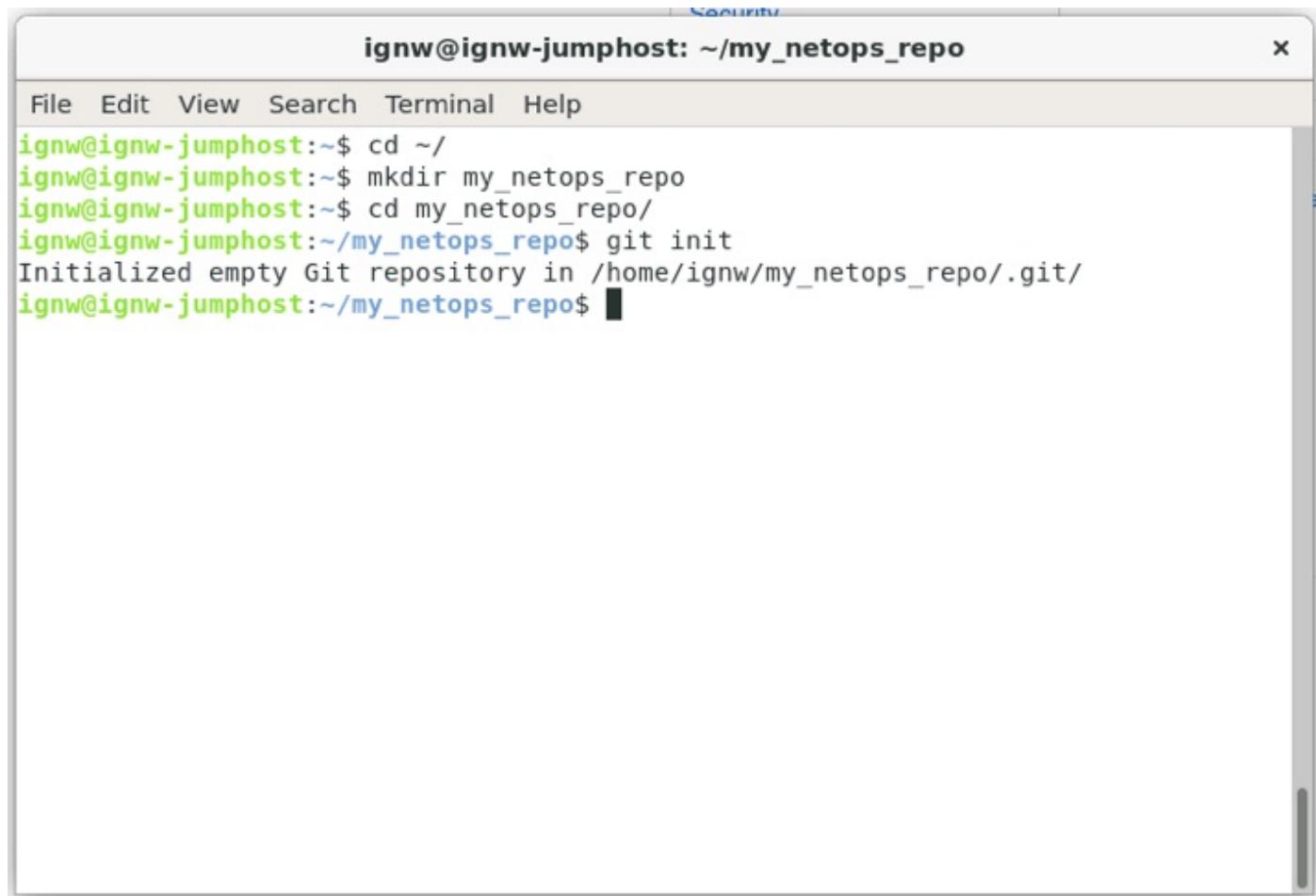
A screenshot of a terminal window titled "ignw@ignw-jumphost: ~". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area shows the command "ssh -T git@github.com" being run, followed by a message from GitHub stating "Hi carlniger! You've successfully authenticated, but GitHub does not provide shell access." The terminal ends with a prompt "ignw@ignw-jumphost:~\$".

## Create a Repo(sitory)

Now that we have our Git/Github communication setup it is time to build a "repo" -- or a repository. A repo is basically a project -- or a home for a project. We'll create a repo now, and perform some basic activities. At the end of the class you can push all of the code you'll produce up to this repository that way you can keep it for reference!

In your home directory (~/) create a new folder called "my\_netops\_repo". "cd" (change directory) into your new repo. At this point we will init(ialize) our repository.

```
$ cd ~/  
$ mkdir my_netops_repo  
$ cd my_netops_repo  
$ git init
```



A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/my\_netops\_repo". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal session shows the following commands being run:

```
ignw@ignw-jumphost:~$ cd ~/
ignw@ignw-jumphost:~$ mkdir my_netops_repo
ignw@ignw-jumphost:~$ cd my_netops_repo/
ignw@ignw-jumphost:~/my_netops_repo$ git init
Initialized empty Git repository in /home/ignw/my_netops_repo/.git/
ignw@ignw-jumphost:~/my_netops_repo$ █
```

Sweet, our first step to getting our repository up and rolling! You'll notice if you log into your Github account (the web page) that nothing has happened. Why is that? You haven't pushed your repository to Github yet. Don't worry, we'll get there!

## Your First Commit

Now that your repository is initialized we'll put something in it. Generally most repos will start with a README.md file, so we'll do that too. Create your new file with the touch command:

```
$ touch README.md
```

A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/my\_netops\_repo". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal session shows the following commands being run:

```
ignw@ignw-jumphost:~$ cd ~/
ignw@ignw-jumphost:~$ mkdir my_netops_repo
ignw@ignw-jumphost:~$ cd my_netops_repo/
ignw@ignw-jumphost:~/my_netops_repo$ git init
Initialized empty Git repository in /home/ignw/my_netops_repo/.git/
ignw@ignw-jumphost:~/my_netops_repo$ touch README.md
ignw@ignw-jumphost:~/my_netops_repo$ ls
README.md
ignw@ignw-jumphost:~/my_netops_repo$
```

Using your favorite editor add the following markdown to the README.md. Note that you are more than welcome to install any editor of your choosing on the system!

```
# My New Repo
All the code for my awesome app is in this repo!
Also, IGNW is the coolest :)
```

A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/my\_netops\_repo". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main pane displays the following text:

```
# My New Repo
All the code for my awesome app is in this repo!
Also, IGNW is the coolest :)
```

Now that we've got *something* in our repo let's commit our new changes. The first thing we need to do is *add* our new file to Git -- this will allow Git to track this file for us. After that we can *commit* our file, and pass it a message.

```
$ git add README.md
$ git commit -m "my first commit"
```

```
ignw@ignw-jumphost: ~/my_netops_repo
File Edit View Search Terminal Help

ignw@ignw-jumphost:~$ cd ~/
ignw@ignw-jumphost:~$ mkdir my_netops_repo
ignw@ignw-jumphost:~$ cd my_netops_repo/
ignw@ignw-jumphost:~/my_netops_repo$ git init
Initialized empty Git repository in /home/ignw/my_netops_repo/.git/
ignw@ignw-jumphost:~/my_netops_repo$ touch README.md
ignw@ignw-jumphost:~/my_netops_repo$ ls
README.md
ignw@ignw-jumphost:~/my_netops_repo$ vi README.md
ignw@ignw-jumphost:~/my_netops_repo$ git add README.md
ignw@ignw-jumphost:~/my_netops_repo$ git commit -m 'my first commit'
[master (root-commit) fc71f63] my first commit
Committer: ignw <ignw@ignw-jumphost.ignw.io>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 3 insertions(+)
create mode 100644 README.md
ignw@ignw-jumphost:~/my_netops_repo$
```

## Time to Push

Now you have a local repo with code committed. However, if you go out to [GitHub.com/username/](https://GitHub.com/username/) you will see that `my_first_repo` doesn't exist yet. Since you haven't pushed your code the remote repository doesn't know about it yet.

To get everything up on Github, We first need to create the repo on GitHub and then push your local commits.

Go to [GitHub.com](https://GitHub.com) and get signed in. Once you're signed in click on the green "New repository" button.

The screenshot shows the GitHub 'Create a New Repository' interface. At the top, the title 'Create a New Repository - Mozilla Firefox' is displayed. Below it, the URL 'GitHub, Inc. (US) https://github.com/new' is shown. The main form has 'Owner' set to 'carlniger' and 'Repository name' set to 'my\_netops\_repo'. A green checkmark icon is next to the repository name input field. Below the repository name, a note says 'Great repository names are short and memorable. Need inspiration? How about [upgraded-lamp](#)'. There is a 'Description (optional)' field with an empty text area. Under 'Visibility', the 'Public' option is selected, with the note 'Anyone can see this repository. You choose who can commit.' The 'Private' option is also available. In the 'Initialize this repository with a README' section, there is a checkbox labeled 'This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.' This checkbox is unselected and highlighted with an orange border. Below this are buttons for 'Add .gitignore: None' and 'Add a license: None'. At the bottom of the form is a large green 'Create repository' button. The footer of the page includes links for 'Terms', 'Privacy', 'Security', 'Status', 'Help', 'Contact GitHub', 'API', 'Training', 'Shop', 'Blog', and 'About'.

Provide a name for your repo (the same name you used before) and ensure that the "initialize this repository with a README" is **not** selected. Finally, click "Create repository".

Now let's connect your local repo to the remote origin and push our local changes.

**Note** Make sure you snag the SSH URL not the HTTP -- it should look like that shown below!

```
$ git remote add origin git@github.com:username/my_netops_repo  
$ git push -u origin master
```

As you can see we are adding a remote repo to Git, and pointing it to the repo previously created on Github. Once that is complete, we simply *push* our local code to the linked Github repository.

```
ignw@ignw-jumphost: ~/my_netops_repo
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_netops_repo$ git remote add origin git@github.com:carlniger/my_netops_repo
ignw@ignw-jumphost:~/my_netops_repo$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 296 bytes | 296.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:carlniger/my_netops_repo
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_netops_repo$
```

Now, if you open your Github account your repo will contain your README file!

carlniger/my\_netops\_repo - Mozilla Firefox

carlniger/my\_netops\_repo | + GitHub, Inc. (US) | https://github.com/carlniger/my\_netops\_repo

This repository Search Pull requests Issues Apps Explore

carlniger / my\_netops\_repo Private

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. Edit Add topics

1 commit 1 branch 0 releases 0 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

Ignw my first commit Latest commit fc71fe63 12 minutes ago

README.md my first commit 12 minutes ago

README.md

## My New Repo

All the code for my awesome app is in this repo! Also, IGNW is the coolest :)

© 2018 GitHub, Inc. Terms Privacy Security Status Help

Contact GitHub API Training Shop Blog About

# Python Hello World

## Overview and Objectives

In this lab you will get started with Python and learn about the basic data structures and logic that you will need to use to be effective with Python.

Objectives:

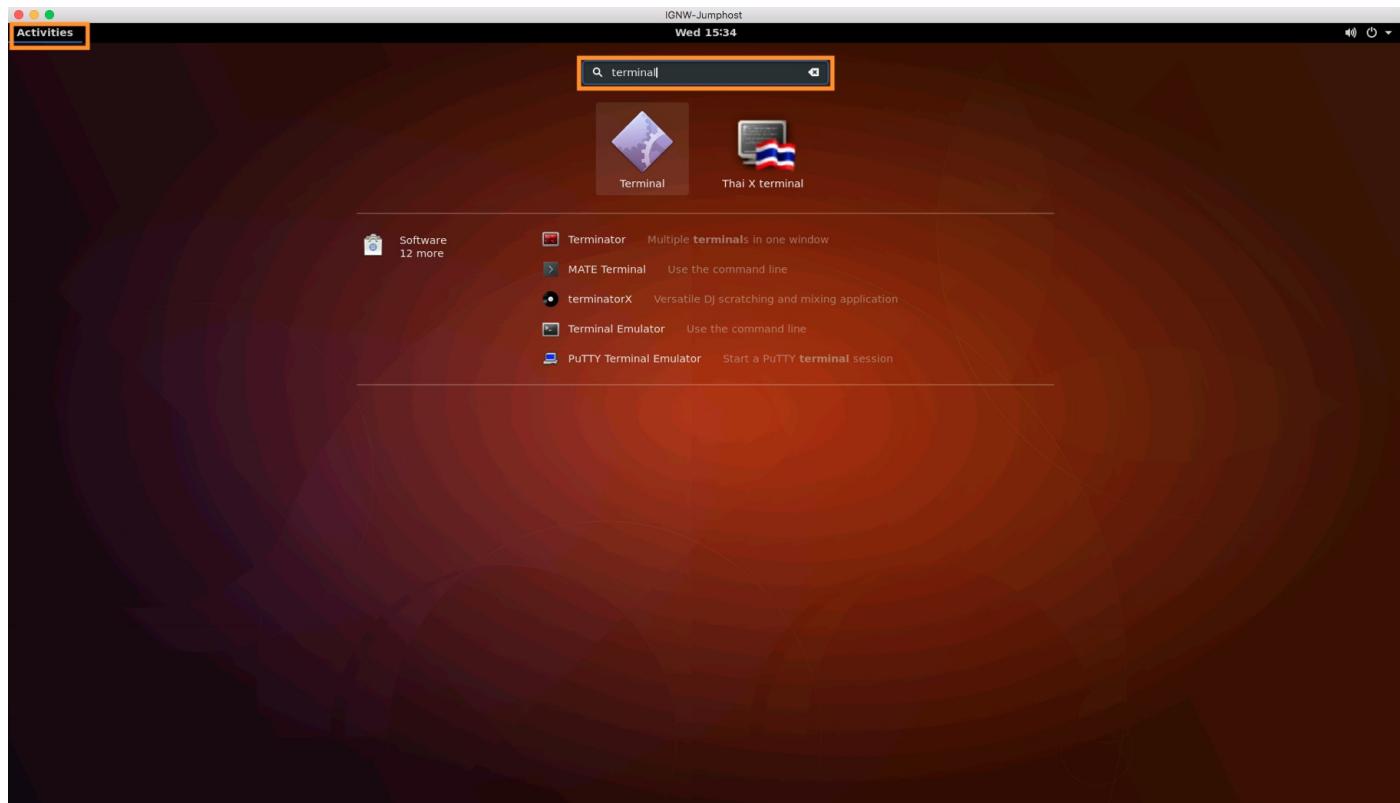
- Execute your "Hello World!" in Python
- Learn about typing
- Learn how to manipulate strings
- Create and access lists and dictionaries
- Start learning the basics of importing modules

## Hello World and the Interactive Interpreter

In this lab we'll play around in the Python Interactive Interpreter shell and begin learning the very basics of Python!

### Access the Interactive Interpreter

Connect to your student RDP server. To access the Python Interactive Interpreter launch the "terminal" app. Open the Terminal application by clicking on the "activities" button on the top left, then typing "terminal" in the search form.

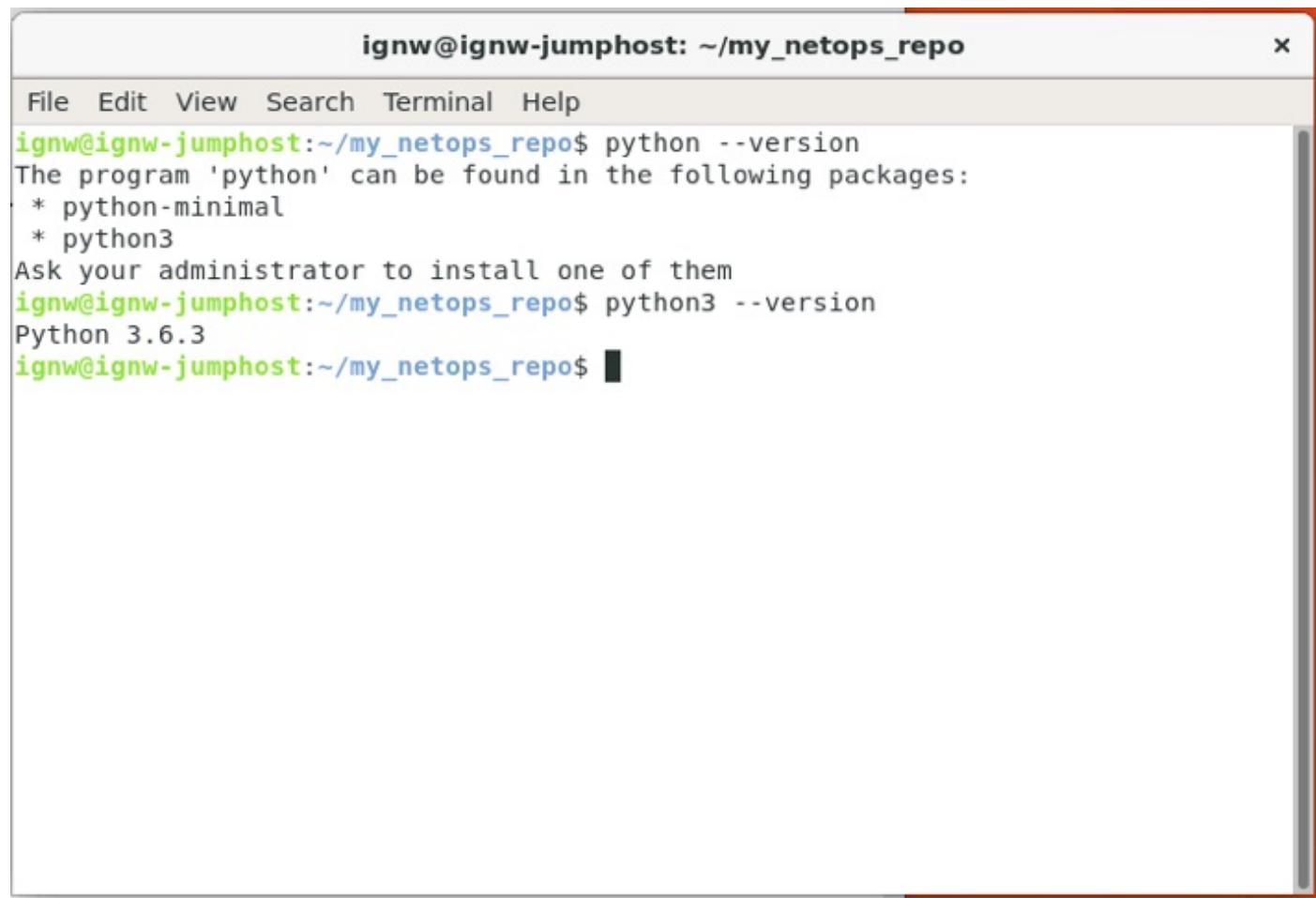


In your terminal window we first need to understand a bit about the Python versions that are installed on the system. Depending on your system there may be different versions installed.

The below screen shot is from Ubuntu 16.04 system, by default Python 2.7 is installed, however for all of our lab tasks we will be using Python 3.6. The simplest way to see which version of Python is available on the system is to use the "--version" argument after the Python keyword. As we can see, the default Python is Python 2.7 as expected. Trying this again for "python3" and "python3.6" shows us that Python 3.5 and Python 3.6 is installed.

The image shows a standard Ubuntu desktop environment. On the left, there is a vertical dock containing icons for various applications: Dash (purple), Terminal (orange), Home (grey), File Manager (grey), Firefox (blue), LibreOffice Writer (blue), LibreOffice Calc (green), LibreOffice Impress (yellow), and LibreOffice Draw (brown). The main workspace is a dark grey color. In the top right corner, there is a terminal window titled "Terminal" with the command line "ubuntu@ip-10-1-1-100: ~". The window contains the following text:  
ubuntu@ip-10-1-1-100:~\$ python --version  
Python 2.7.12  
ubuntu@ip-10-1-1-100:~\$ python3 --version  
Python 3.5.2  
ubuntu@ip-10-1-1-100:~\$ python3.6 --version  
Python 3.6.3  
ubuntu@ip-10-1-1-100:~\$ █

The Jumphost is an Ubuntu 17.10 instance, try running "python --version" on your system:

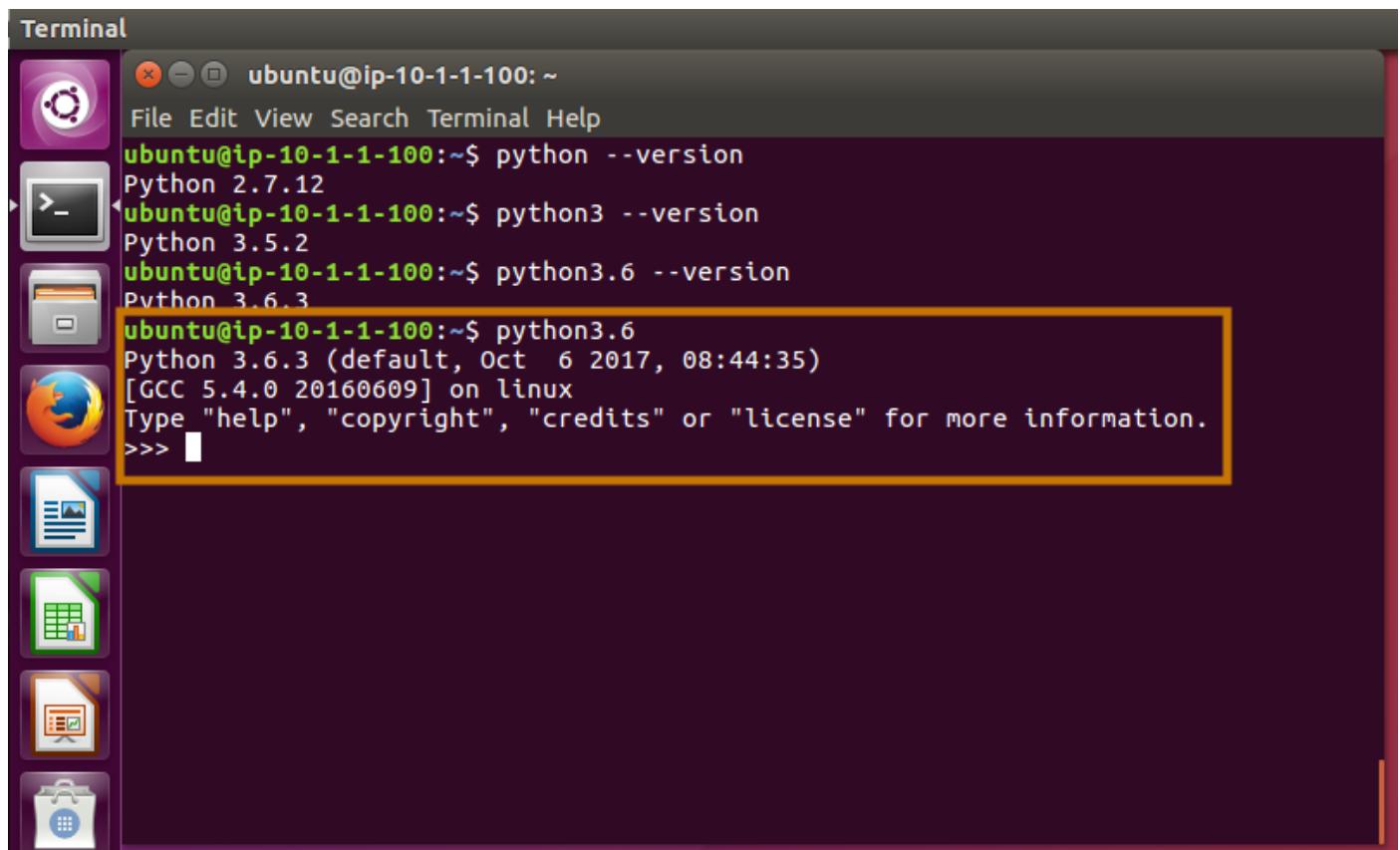


A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/my\_netops\_repo". The window has a standard Linux-style header bar with "File Edit View Search Terminal Help" menu items. The main area of the terminal shows the following command-line session:

```
ignw@ignw-jumphost:~/my_netops_repo$ python --version
The program 'python' can be found in the following packages:
 * python-minimal
 * python3
Ask your administrator to install one of them
ignw@ignw-jumphost:~/my_netops_repo$ python3 --version
Python 3.6.3
ignw@ignw-jumphost:~/my_netops_repo$
```

We now know that we will have to type "python3" at the prompt if we want to ensure we are indeed using Python 3.6. For the majority of the labs Python 3.5 would be just fine (if it were installed, or you were using an older Ubuntu system with it), however there are some neat enhancements in Python 3.6 that will be covered in some of the tasks that are not supported in Python 3.5. Note that the lab may show "python3" or "python3.6" since we are operating with only Python 3.6 installed these commands will have the same result, but again, be cautious if you are operating on a system with multiple versions installed.

Finally, to enter the interactive interpreter simply type "python3.6". The ">>>" prompt signals that you are in the Python interpreter session.



## Variables and Printing

In this step we will begin by assigning the value "Hello World!" to a variable, and then printing the value of our variable to standard out (to our console).

In the interpreter create a variable called "mystring" and assign it the value "Hello World!"

```
mystring = "Hello World!"
```

We can validate that our variable is storing the value of our string as we expect it to by accessing the variable. In the Interpreter type "mystring" and hit enter.

Terminal

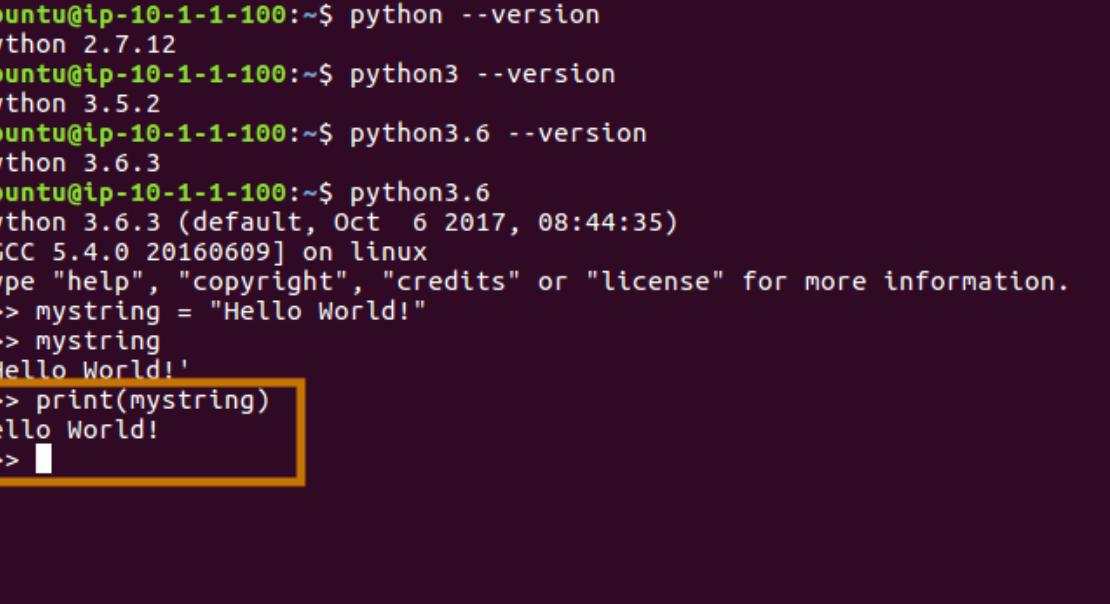
The screenshot shows a terminal window titled "Terminal" running on an Ubuntu desktop. The window title bar says "ubuntu@ip-10-1-1-100: ~". The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal content shows the following:

```
ubuntu@ip-10-1-1-100:~$ python --version
Python 2.7.12
ubuntu@ip-10-1-1-100:~$ python3 --version
Python 3.5.2
ubuntu@ip-10-1-1-100:~$ python3.6 --version
Python 3.6.3
ubuntu@ip-10-1-1-100:~$ python3.6
Python 3.6.3 (default, Oct  6 2017, 08:44:35)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> mystring = "Hello World!"
>>> mystring
'Hello World!'
>>> 
```

What happens? You should see "Hello World!" printed to the screen. While we are in the interactive interpreter we can "print" our variables by simply entering the name of the variable -- once we leave the interpreter that will no longer be the case and we will have to call the "print" function in order to print things to standard out. Let's try that now, in the interpreter enter "print(mystring)" and hit enter.

```
print(mystring)
```



The screenshot shows a terminal window on an Ubuntu desktop. The terminal title bar reads "Terminal" and the session identifier is "ubuntu@ip-10-1-1-100: ~". The window contains the following text:

```
ubuntu@ip-10-1-1-100:~$ python --version
Python 2.7.12
ubuntu@ip-10-1-1-100:~$ python3 --version
Python 3.5.2
ubuntu@ip-10-1-1-100:~$ python3.6 --version
Python 3.6.3
ubuntu@ip-10-1-1-100:~$ python3.6
Python 3.6.3 (default, Oct  6 2017, 08:44:35)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> mystring = "Hello World!"
>>> mystring
'Hello World!'
>>> print(mystring)
Hello World!
>>> 
```

A yellow rectangular box highlights the command `>>> print(mystring)` and its output "Hello World!".

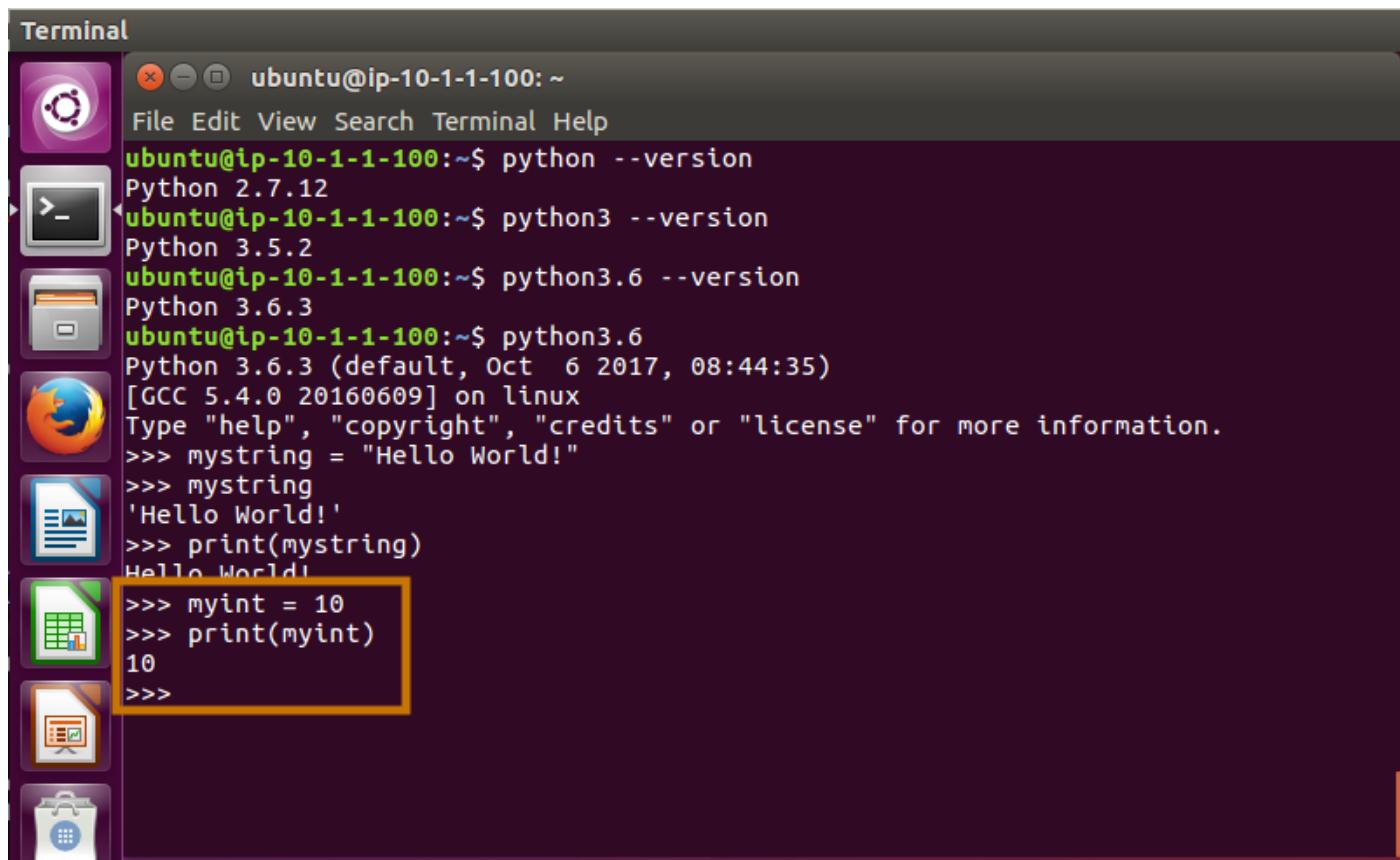
Of course this is *mostly* the same output as before since our variable has not changed, yet before we saw quotes around our string, and this second time we did not. For now you don't need to worry too much about this, but if you're curious -- the interactive interpreter returns the *repr* value of any object when you hit enter -- whereas the print function is outputting the value of the variable as assigned.

Let's create another variable called "myint" and assign it the value of "10".

```
myint = 10
```

Print out your variable "myint".

```
print(myint)
```



A screenshot of an Ubuntu desktop environment. On the left, there's a dock with various icons: Dash, Home, Applications, Places, and others. The main window is a terminal window titled "Terminal". The terminal shows the following session:

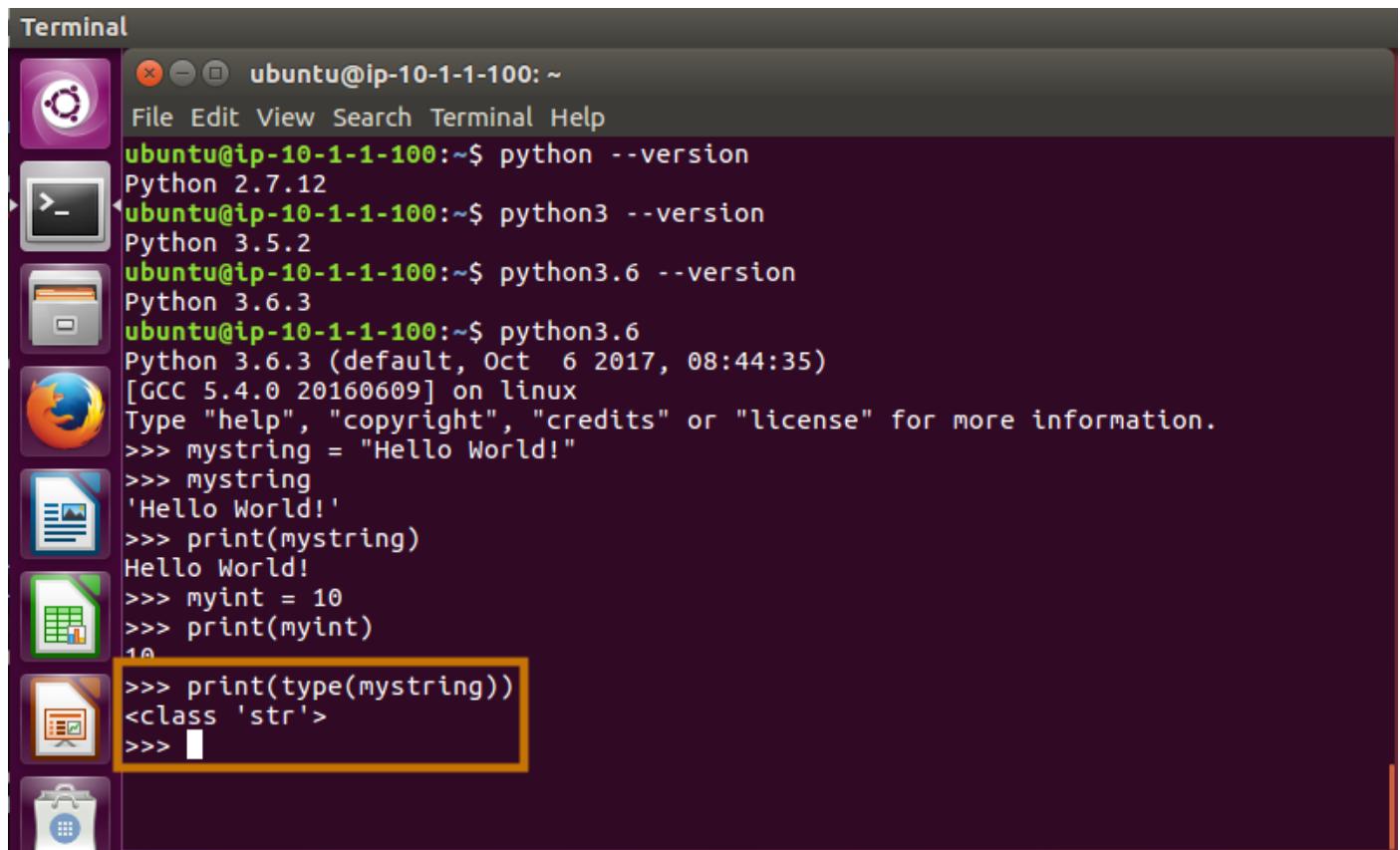
```
ubuntu@ip-10-1-1-100:~$ python --version
Python 2.7.12
ubuntu@ip-10-1-1-100:~$ python3 --version
Python 3.5.2
ubuntu@ip-10-1-1-100:~$ python3.6 --version
Python 3.6.3
ubuntu@ip-10-1-1-100:~$ python3.6
Python 3.6.3 (default, Oct  6 2017, 08:44:35)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> mystring = "Hello World!"
>>> mystring
'Hello World!'
>>> print(mystring)
Hello World!
>>> myint = 10
>>> print(myint)
10
>>>
```

There are other variables types in Python, but for now we'll just stick with strings (text) and ints (numbers).

## Learn about Types

Recall that Python is not a "strongly typed" language -- what does that actually mean? What is a type? A type is just the "kind" of variable the program is working with. In the previous step you created both a "string" and an "int" (integer) -- those are *types* within Python. We can check what type a variable is by using the `type` function -- in this case, we'll print (output to standard out) the type of our "mystring" variable to check to make sure it is in fact a string.

```
print(type(mystring))
```



The screenshot shows a terminal window titled "Terminal" on an Ubuntu desktop. The window contains the following text:

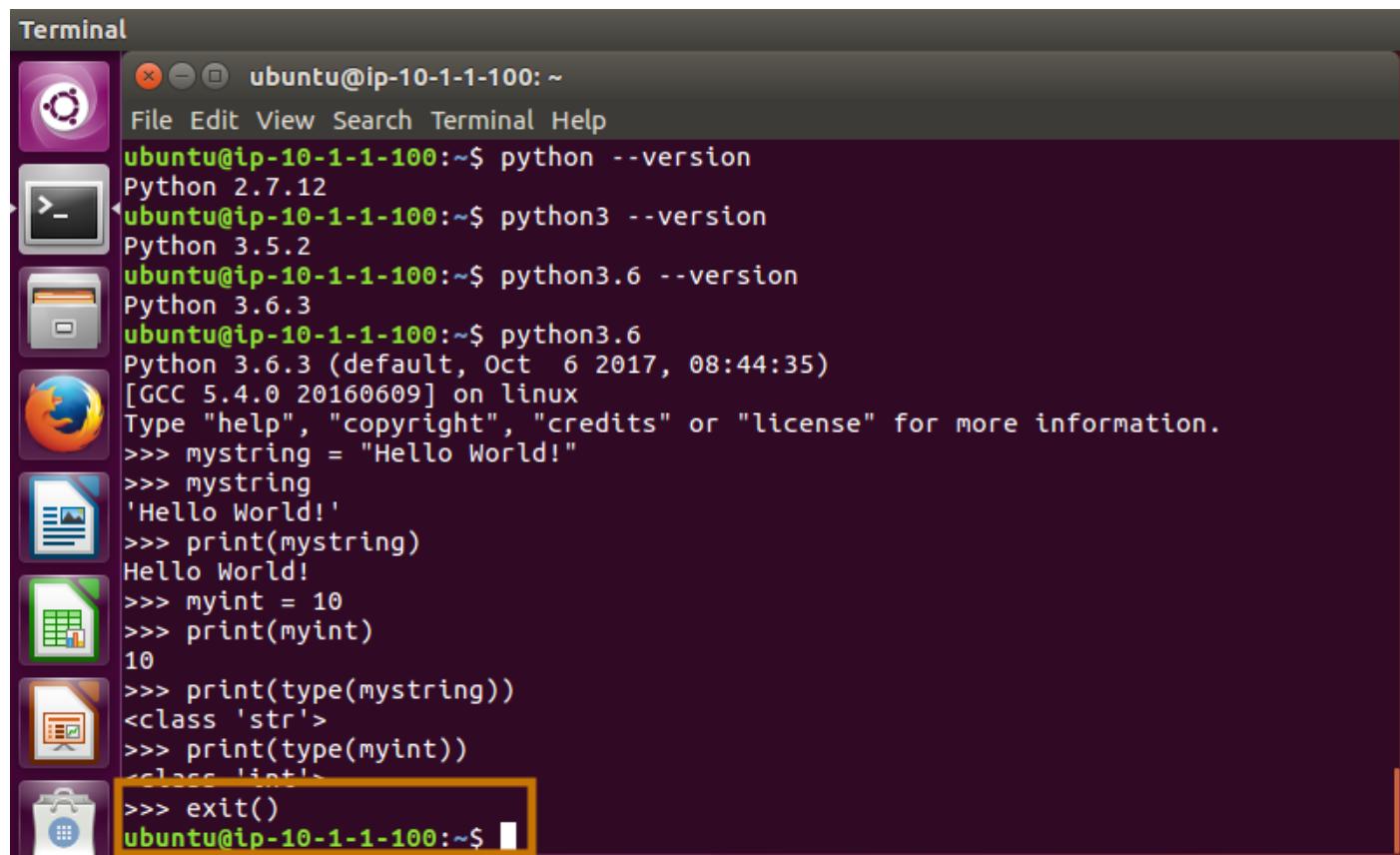
```
ubuntu@ip-10-1-1-100:~$ python --version
Python 2.7.12
ubuntu@ip-10-1-1-100:~$ python3 --version
Python 3.5.2
ubuntu@ip-10-1-1-100:~$ python3.6 --version
Python 3.6.3
ubuntu@ip-10-1-1-100:~$ python3.6
Python 3.6.3 (default, Oct  6 2017, 08:44:35)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> mystring = "Hello World!"
>>> mystring
'Hello World!'
>>> print(mystring)
Hello World!
>>> myint = 10
>>> print(myint)
10
>>> print(type(mystring))
<class 'str'>
>>> 
```

The result of this should not surprise you! Go ahead and do the same for your integer variable.

```
print(type(myint))
```

In the next lab we'll learn what all this really means to us in practical terms! For now, go ahead and exit the interactive interpreter -- you can do so by typing "exit()"

```
exit()
```



A screenshot of an Ubuntu desktop environment. On the left, there is a dock with several icons: Dash, Home, Applications, Places, and a Dash search bar. The main window is a terminal window titled "Terminal". The terminal shows the following session:

```
ubuntu@ip-10-1-1-100:~$ python --version
Python 2.7.12
ubuntu@ip-10-1-1-100:~$ python3 --version
Python 3.5.2
ubuntu@ip-10-1-1-100:~$ python3.6 --version
Python 3.6.3
ubuntu@ip-10-1-1-100:~$ python3.6
Python 3.6.3 (default, Oct  6 2017, 08:44:35)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> mystring = "Hello World!"
>>> mystring
'Hello World!'
>>> print(mystring)
Hello World!
>>> myint = 10
>>> print(myint)
10
>>> print(type(mystring))
<class 'str'>
>>> print(type(myint))
<class 'int'>
>>> exit()
ubuntu@ip-10-1-1-100:~$
```

## Manipulating Strings

In this lab we'll be working with strings to understand how we can format text. We will also learn about indexing and slicing -- two techniques that will be invaluable for you in your journey with Python!

### String Formatting

Enter the interactive interpreter once more.

Create two new variables, "mystr1" and "mystr2" assigning them the values "Hello" and "World!" respectively.

```
python mystr1 = "Hello" mystr2 = "World!"
```

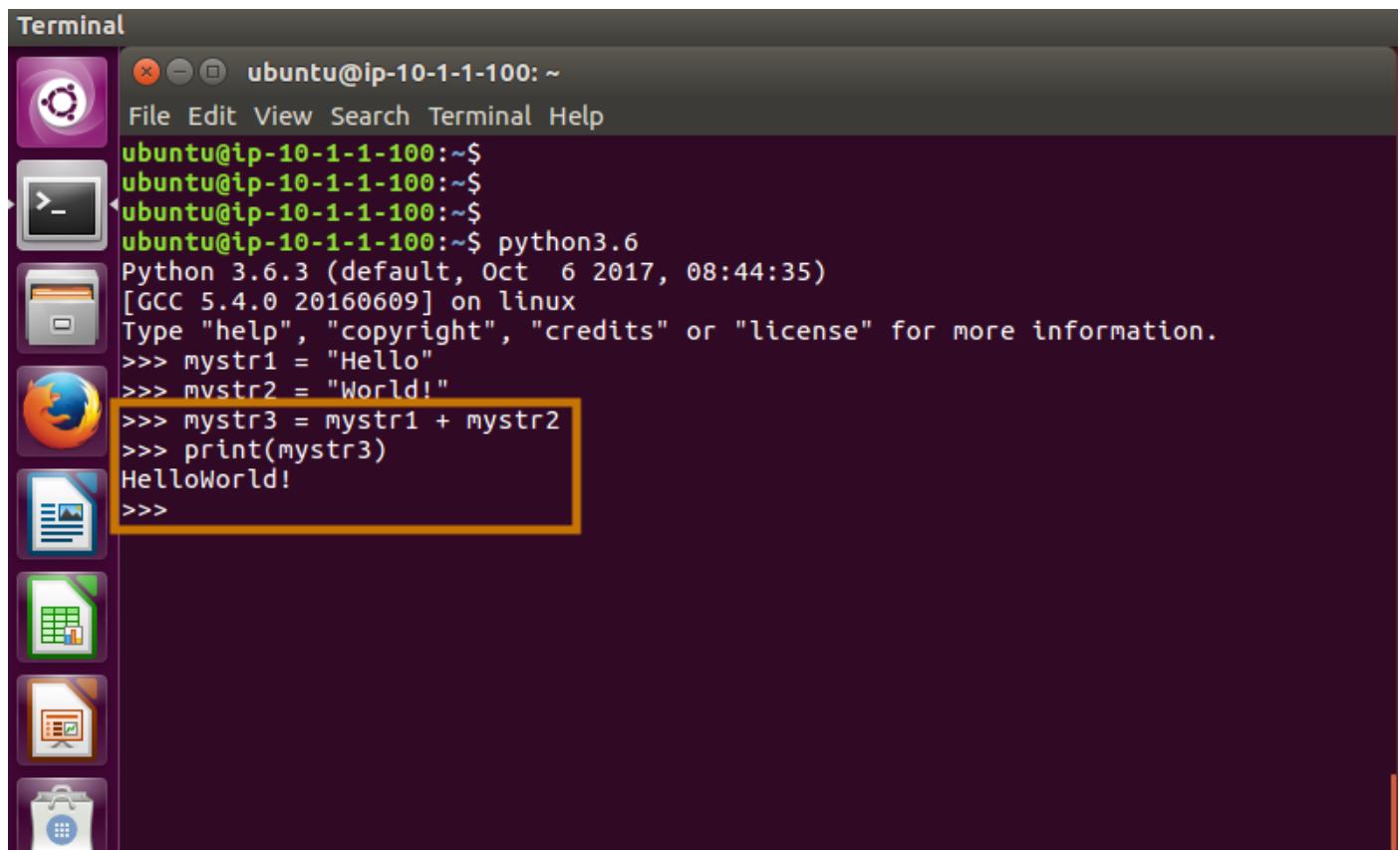
```
Terminal
ubuntu@ip-10-1-1-100: ~
File Edit View Search Terminal Help
ubuntu@ip-10-1-1-100:~$ python3.6
Python 3.6.3 (default, Oct  6 2017, 08:44:35)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> mystr1 = "Hello"
>>> mystr2 = "World!"
>>> 
```

Let's go ahead and combine these strings so that we can print out "Hello World!" like we did in the previous section. In Python there are several ways to go about doing this -- we'll start with perhaps the simplest method -- concatenation. Concatenation is simply combining two variables together to form one. Let's try combining our two variables and assigning the outcome of that to a new variable "mystr3".

```
mystr3 = mystr1 + mystr2
```

If you print this, what do you think the outcome will be? Go ahead and try it to check your thinking.

```
print(mystr3)
```

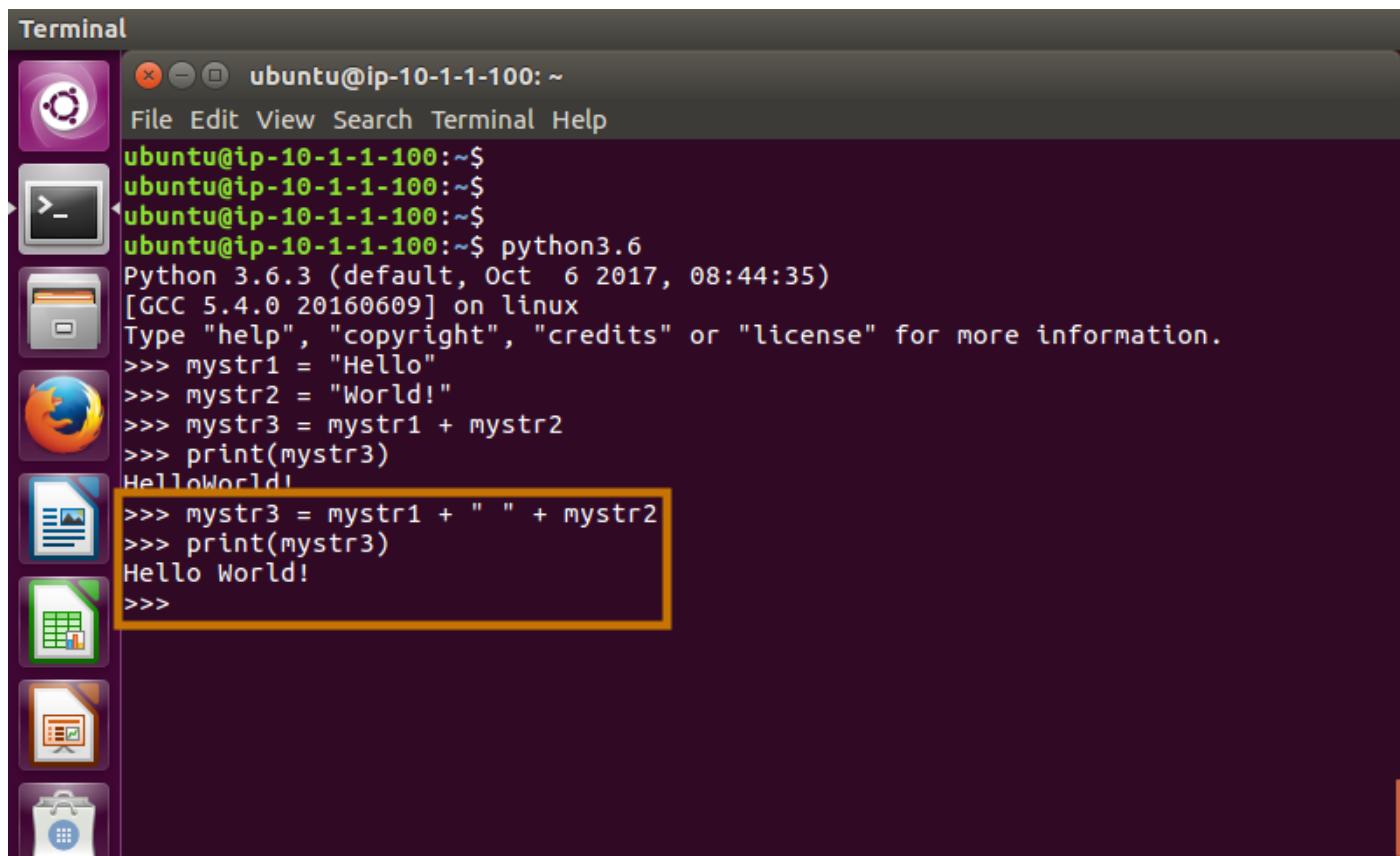


A screenshot of an Ubuntu desktop environment. On the left, there's a dock with various icons: Dash, Home, Applications, Places, System, and Help. The main window is a terminal window titled "Terminal". The terminal shows the following Python session:

```
ubuntu@ip-10-1-1-100: ~
File Edit View Search Terminal Help
ubuntu@ip-10-1-1-100: ~$ python3.6
Python 3.6.3 (default, Oct  6 2017, 08:44:35)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> mystr1 = "Hello"
>>> mystr2 = "World!"
>>> mystr3 = mystr1 + mystr2
>>> print(mystr3)
HelloWorld!
>>>
```

If all went well this will have worked, however it may not be the outcome you were hoping for! There is no space! Let's try to overwrite the value of "mystr3" to contain a space. We can do that by simply adding in an "empty" string (" ") containing a space in between our two "mystr" variables.

```
mystr3 = mystr1 + " " + mystr2
print(mystr3)
```



A screenshot of an Ubuntu desktop environment. On the left, there's a dock with various icons: Dash, Home, Applications, Help, and several system tray icons. The main window is a terminal window titled "Terminal". The terminal shows a command-line session:

```
ubuntu@ip-10-1-1-100:~$ python3.6
Python 3.6.3 (default, Oct  6 2017, 08:44:35)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> mystr1 = "Hello"
>>> mystr2 = "World!"
>>> mystr3 = mystr1 + mystr2
>>> print(mystr3)
HelloWorld!
>>> mystr3 = mystr1 + " " + mystr2
>>> print(mystr3)
Hello World!
>>>
```

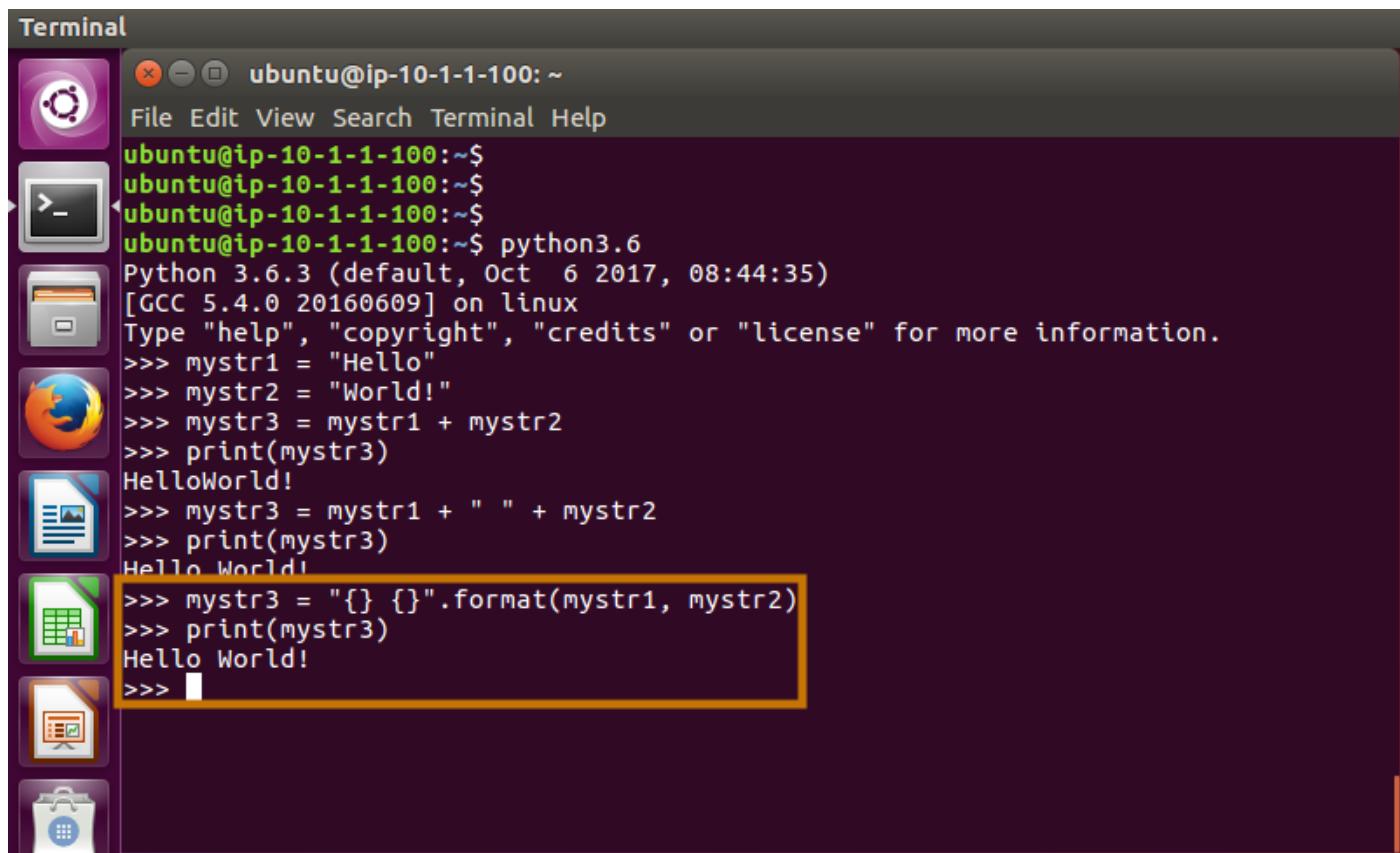
Thats more like it!

Python provides a bunch of other ways to manipulate strings -- two of the more common and most recent ways are via the `format()` method, or string interpolation. Using the same example as before you can use the `format` method like so:

```
mystr3 = "{} {}".format(mystr1, mystr2)
```

In the above example you can see that the "squiggly braces" ({ and }) essentially become placeholders for values that you would like to insert into a string. In this case, we insert the same two strings that we've been working with. There is a bit more to the `format` method, but for now this should get you up and running.

Print your string out to validate that it looks as it should.

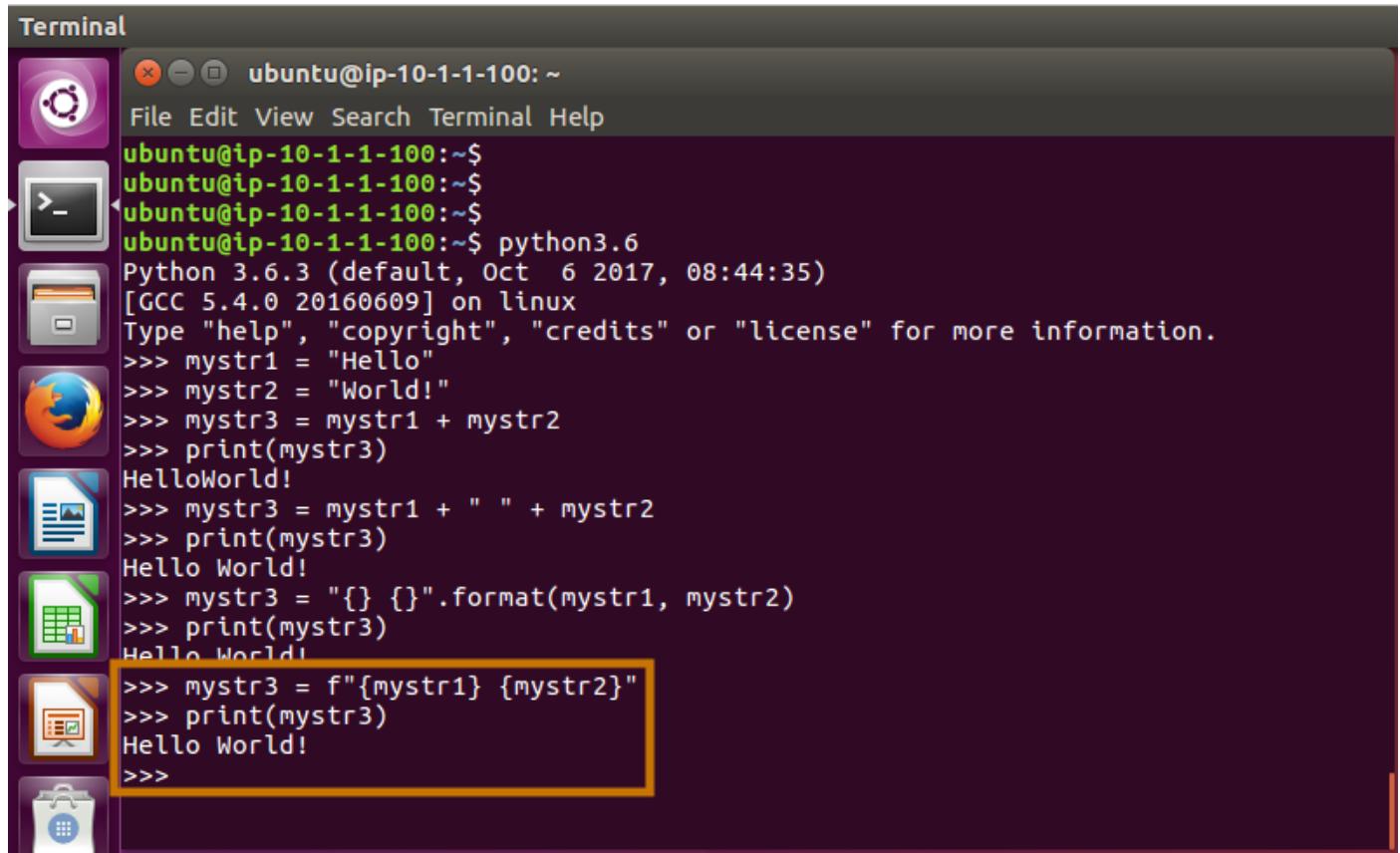
A screenshot of an Ubuntu desktop environment. On the left is a dock with icons for Dash, Home, Applications, and several system monitors. The main window is a terminal window titled "Terminal". The terminal shows a session where Python 3.6 is running. The user has defined two strings, "Hello" and "World!", and concatenated them. They then used string interpolation to print the strings directly from the variables. The command `>>> mystr3 = f'{mystr1} {mystr2}'` is highlighted with a yellow box.

```
ubuntu@ip-10-1-1-100:~$ python3.6
Python 3.6.3 (default, Oct  6 2017, 08:44:35)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> mystr1 = "Hello"
>>> mystr2 = "World!"
>>> mystr3 = mystr1 + mystr2
>>> print(mystr3)
HelloWorld!
>>> mystr3 = mystr1 + " " + mystr2
>>> print(mystr3)
Hello World!
>>> mystr3 = "{} {}".format(mystr1, mystr2)
>>> print(mystr3)
Hello World!
>>> 
```

String interpolation is a more recent implementation of string formatting starting with Python 3.6. Since we are running Python 3 in our lab we can go ahead and test this out.

```
mystr3 = f'{mystr1} {mystr2}'
```

As you can see, it's similar to the formatting method in its most basic usage, but instead of needing to call the ".format()" method on our string we are able to substitute our other variables in line with our string.



The image shows a screenshot of an Ubuntu desktop environment. On the left, there is a dock with various icons: Dash, Home, Applications, Help, and several desktop environment icons like Nautilus, Gnome Terminal, and System Settings. The main window is a terminal window titled "Terminal". The terminal session starts with the prompt "ubuntu@ip-10-1-1-100:~\$". It then shows the Python 3.6 interpreter running a script. The script creates two variables, "mystr1" and "mystr2", with values "Hello" and "World!" respectively. It concatenates them using the "+" operator and prints the result. It then uses the "format" method to create a new string "mystr3" with placeholders {}, and prints it. Finally, it uses f-string interpolation to create another "mystr3" with the same value and prints it. The last command entered is ">>>". A yellow box highlights the last three lines of code.

```
ubuntu@ip-10-1-1-100:~$ python3.6
Python 3.6.3 (default, Oct  6 2017, 08:44:35)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> mystr1 = "Hello"
>>> mystr2 = "World!"
>>> mystr3 = mystr1 + mystr2
>>> print(mystr3)
HelloWorld!
>>> mystr3 = mystr1 + " " + mystr2
>>> print(mystr3)
Hello World!
>>> mystr3 = "{} {}".format(mystr1, mystr2)
>>> print(mystr3)
Hello World!
>>> mystr3 = f'{mystr1} {mystr2}'
>>> print(mystr3)
Hello World!
>>>
```

**Note:** If you run into an error at this step, check to make sure you are using Python 3.6! String interpolation is a new 3.6+ feature!

Go ahead and create a few more variables (strings and integers) and try to combine them using any or all of the above methods.

How did it go? Did you run into any errors? Maybe Python complaining about a "TypeError"?

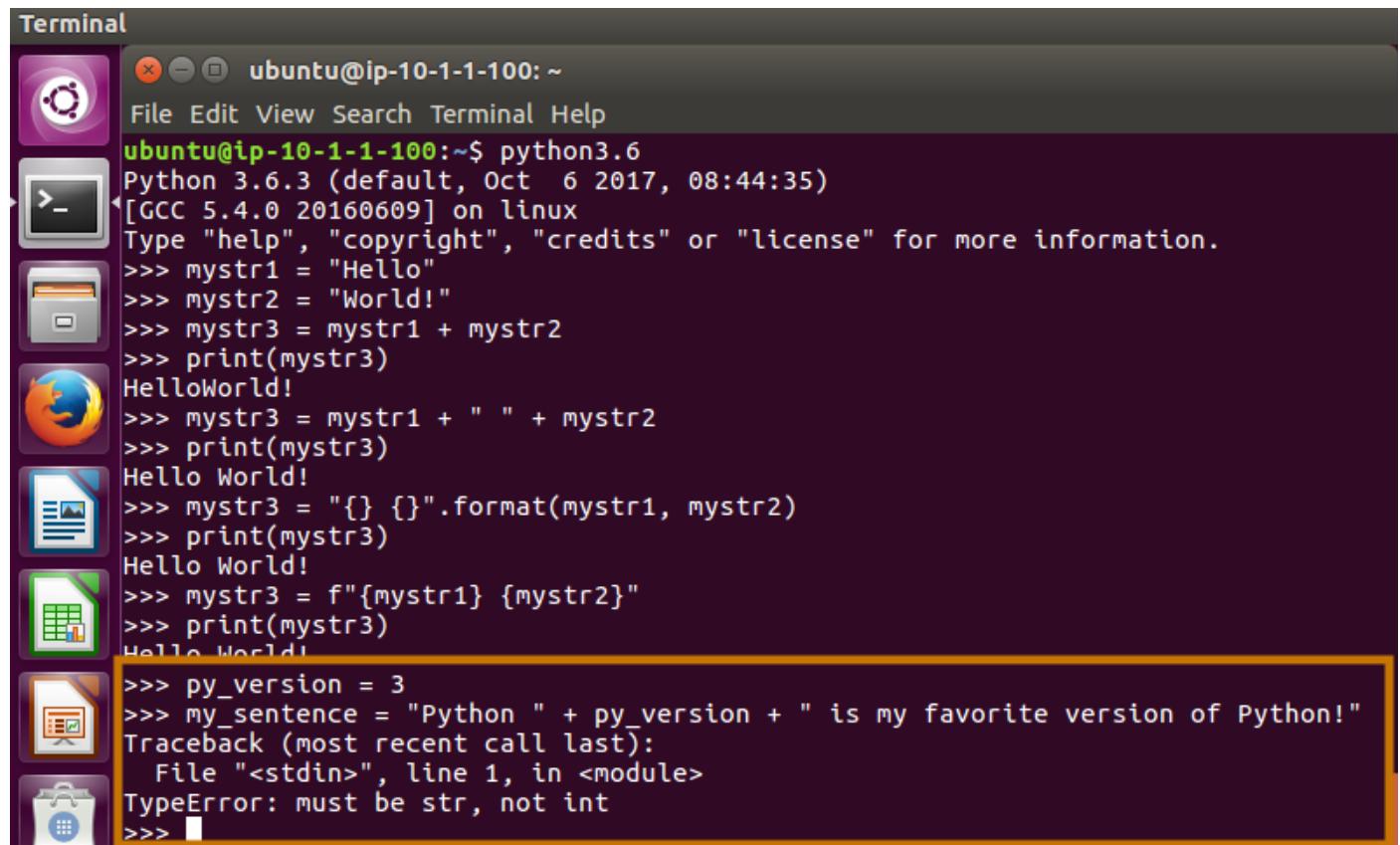
Create a new variable "py\_version" and assign it the value "3" as an integer.

```
py_version = 3
```

Now, let's try to put that into a sentence using concatenation.

```
my_sentence = "Python " + py_version + " is my favorite version of Python!"
```

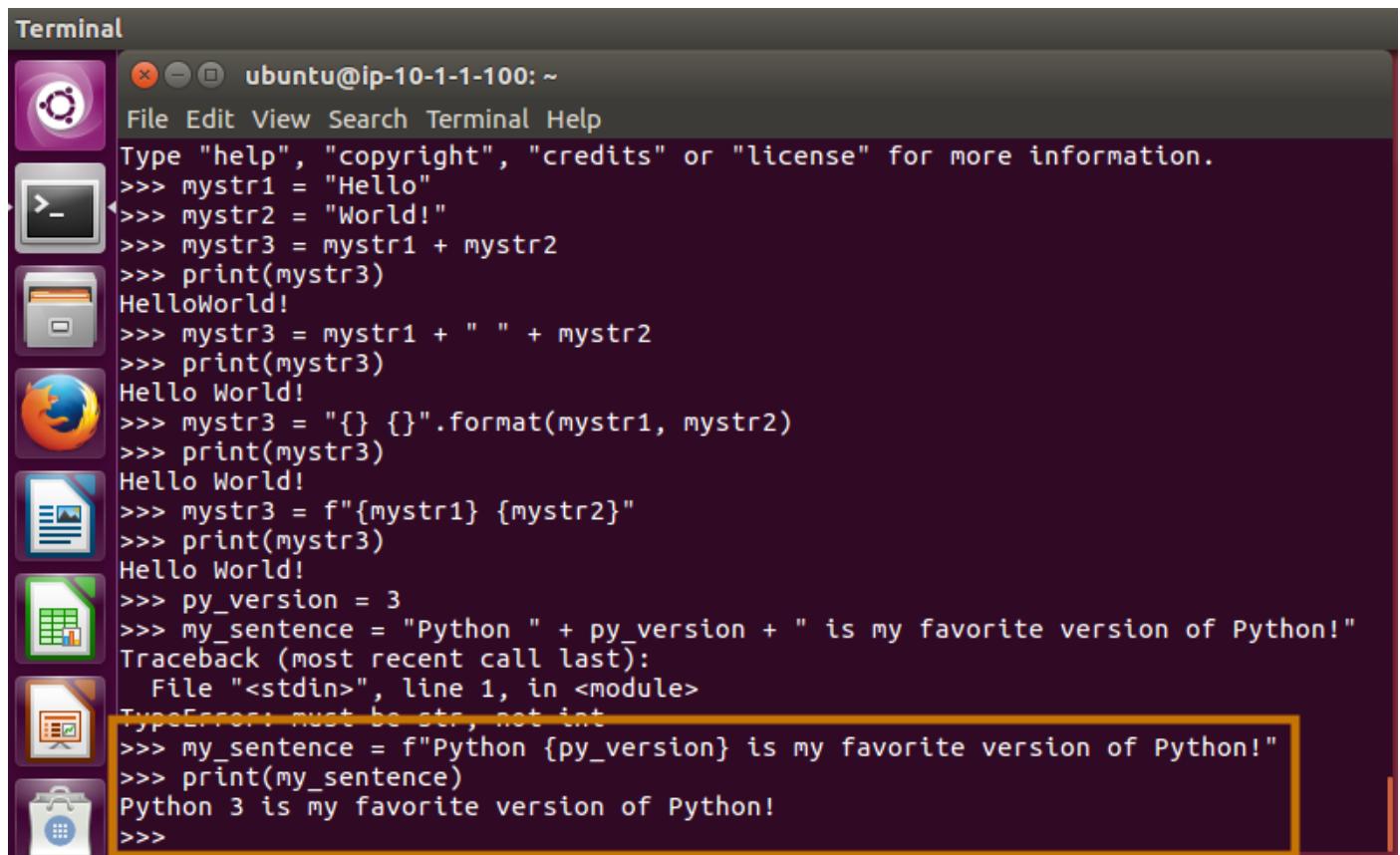
Terminal



```
ubuntu@ip-10-1-1-100: ~
File Edit View Search Terminal Help
ubuntu@ip-10-1-1-100:~$ python3.6
Python 3.6.3 (default, Oct  6 2017, 08:44:35)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> mystr1 = "Hello"
>>> mystr2 = "World!"
>>> mystr3 = mystr1 + mystr2
>>> print(mystr3)
HelloWorld!
>>> mystr3 = mystr1 + " " + mystr2
>>> print(mystr3)
Hello World!
>>> mystr3 = "{} {}".format(mystr1, mystr2)
>>> print(mystr3)
Hello World!
>>> mystr3 = f"{mystr1} {mystr2}"
>>> print(mystr3)
Hello World!
>>> py_version = 3
>>> my_sentence = "Python " + py_version + " is my favorite version of Python!"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> █
```

Uhoh! Looks like we ran into a "TypeError"... let's try it in a different way to see if we run into the same issue.

```
my_sentence = f"Python {py_version} is my favorite version of Python!"
```



A screenshot of a Ubuntu desktop environment. On the left, there's a dock with icons for Dash, Home, Applications, and the Dash search bar. The main window is a terminal window titled "Terminal". The terminal shows a Python session:

```
ubuntu@ip-10-1-1-100: ~
File Edit View Search Terminal Help
Type "help", "copyright", "credits" or "license" for more information.
>>> mystr1 = "Hello"
>>> mystr2 = "World!"
>>> mystr3 = mystr1 + mystr2
>>> print(mystr3)
HelloWorld!
>>> mystr3 = mystr1 + " " + mystr2
>>> print(mystr3)
Hello World!
>>> mystr3 = "{} {}".format(mystr1, mystr2)
>>> print(mystr3)
Hello World!
>>> mystr3 = f"{mystr1} {mystr2}"
>>> print(mystr3)
Hello World!
>>> py_version = 3
>>> my_sentence = "Python " + py_version + " is my favorite version of Python!"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> my_sentence = f"Python {py_version} is my favorite version of Python!"
>>> print(my_sentence)
Python 3 is my favorite version of Python!
>>>
```

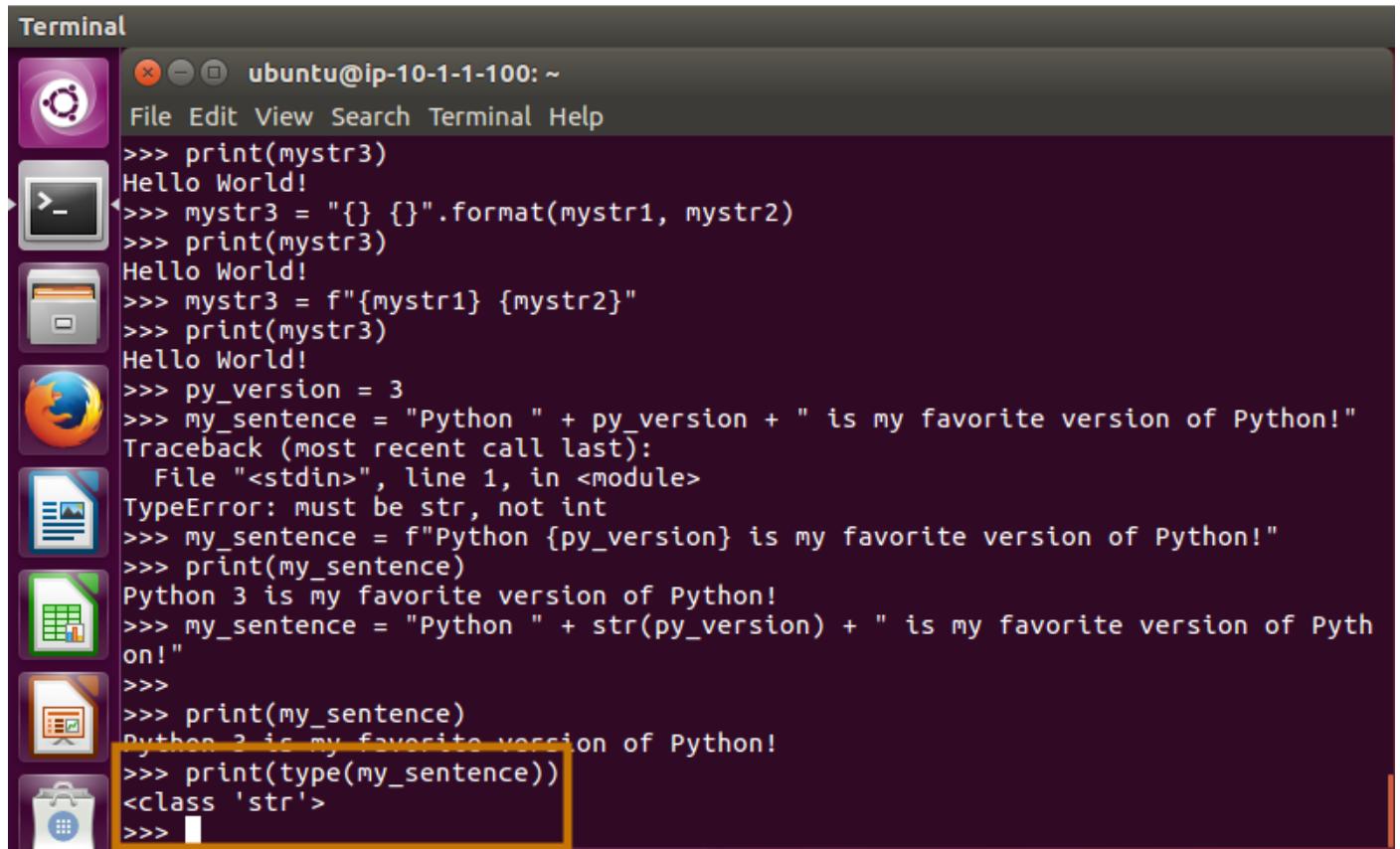
Ahhh, it seems that if we use a string formatting technique as opposed to concatenation, we can avoid this "TypeError" issue. So why did we get the "TypeError" with concatenation in the first place? Simply because a "string" and an "integer" are not of the same *type* in Python, yet we are trying to treat them as if they are. When we use any of the string formatting techniques Python knows we are trying to massage the data to be represented as a string so it takes care of the conversion for us. We can also do this with concatenation if we want:

```
my_sentence = "Python " + str(py_version) + " is my favorite version of Python
!"
print(my_sentence)
```

```
Terminal
ubuntu@ip-10-1-1-100: ~
File Edit View Search Terminal Help
HelloWorld!
>>> mystr3 = mystr1 + " " + mystr2
>>> print(mystr3)
Hello World!
>>> mystr3 = "{} {}".format(mystr1, mystr2)
>>> print(mystr3)
Hello World!
>>> mystr3 = f"{mystr1} {mystr2}"
>>> print(mystr3)
Hello World!
>>> py_version = 3
>>> my_sentence = "Python " + py_version + " is my favorite version of Python!"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> my_sentence = f"Python {py_version} is my favorite version of Python!"
>>> print(my_sentence)
Python 3 is my favorite version of Python!
>>> my_sentence = "Python " + str(py_version) + " is my favorite version of Python!"
>>>
>>> print(my_sentence)
Python 3 is my favorite version of Python!
>>>
```

As you can see we've used the "str()" method to convert our integer to a string to be concatenated into our sentence. As we begin to work with other *types* you will need to be sure you are remembering which variables are which types to avoid running into errors. Always remember that you can ask Python for the type of a particular variable with the "type()" method:

```
print(type(my_sentence))
```



The screenshot shows a terminal window titled "Terminal" running on an Ubuntu desktop. The terminal window has a dark background and contains the following Python code:

```
ubuntu@ip-10-1-1-100: ~
File Edit View Search Terminal Help
>>> print(mystr3)
Hello World!
>>> mystr3 = "{} {}".format(mystr1, mystr2)
>>> print(mystr3)
Hello World!
>>> mystr3 = f"{mystr1} {mystr2}"
>>> print(mystr3)
Hello World!
>>> py_version = 3
>>> my_sentence = "Python " + py_version + " is my favorite version of Python!"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> my_sentence = f"Python {py_version} is my favorite version of Python!"
>>> print(my_sentence)
Python 3 is my favorite version of Python!
>>> my_sentence = "Python " + str(py_version) + " is my favorite version of Python!"
>>>
>>> print(my_sentence)
Python 3 is my favorite version of Python!
>>> print(type(my_sentence))
<class 'str'>
>>>
```

## Working with Lists

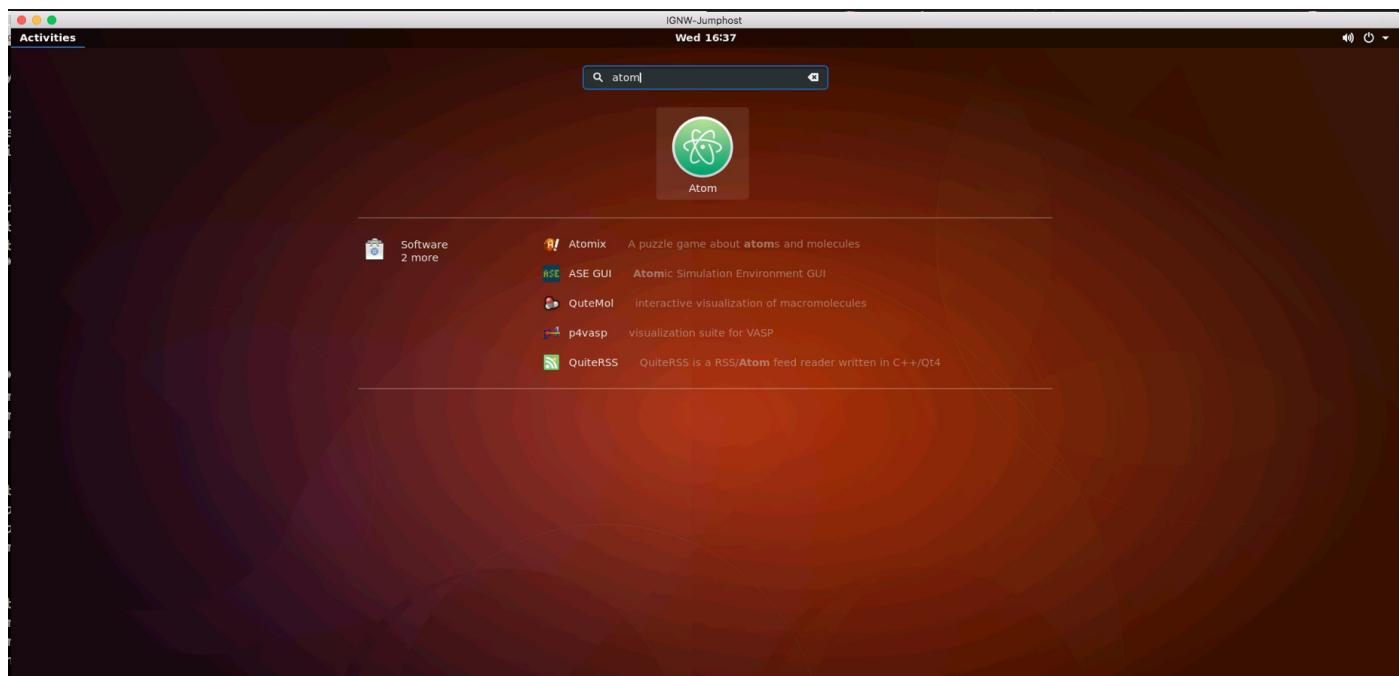
Now that we have a basic understanding of Python, it's time to learn about lists. Lists in Python, sometimes referred to as Arrays (especially by folks who have experience in other languages), are a data structure that contains ordered elements. Lists can contain data of any *type* that Python supports, and can even contain mixed *types* within a list. For example, you can have integers and strings inside a list. For now, we'll focus on the basics.

### Accessing Elements in a List

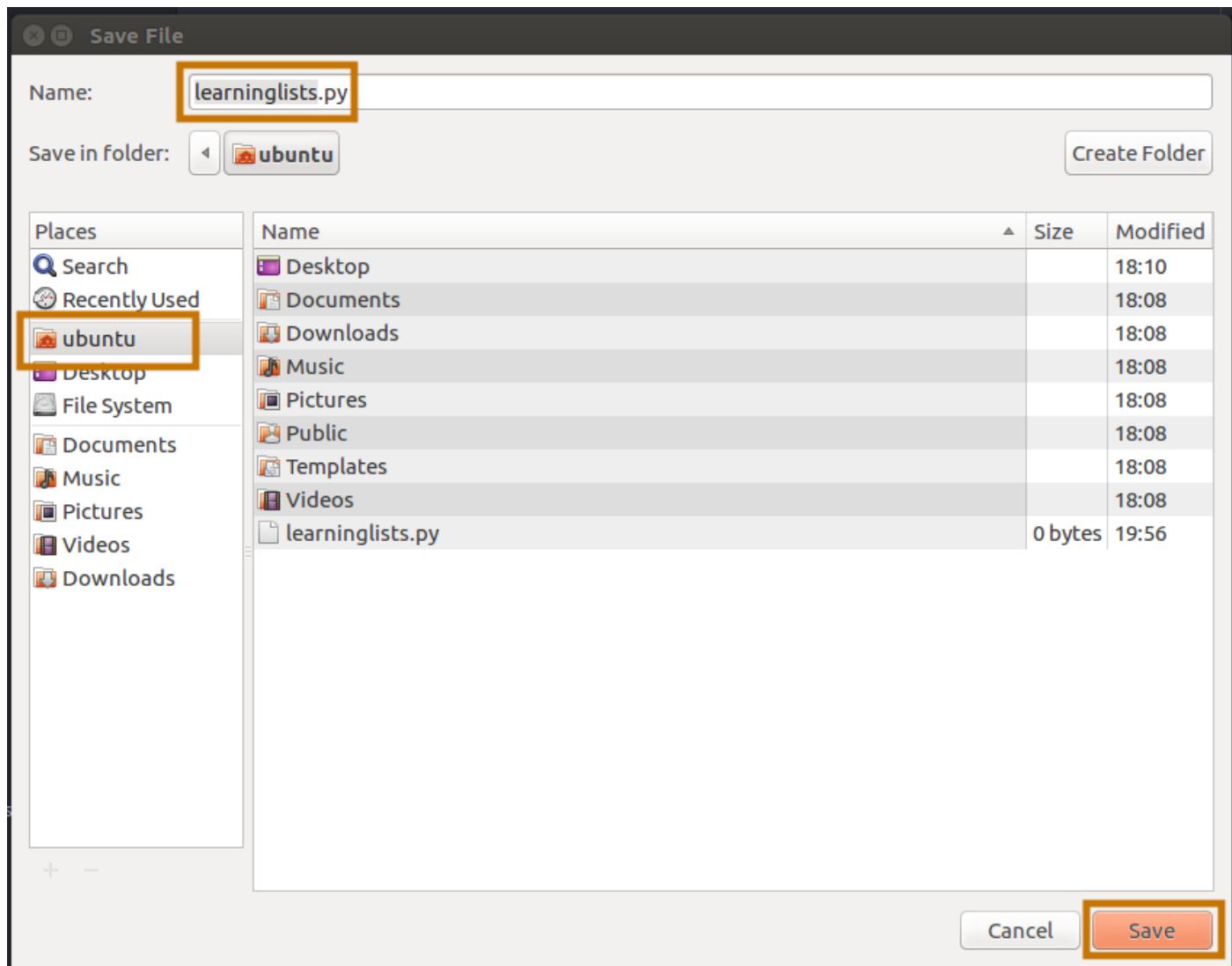
Lists in Python are ordered -- this means that if you insert data into them, the sequence in which you insert the data will be maintained. Since a list is ordered, there must be a way to access each element of a list respecting that order. Accessing an element in a list is accomplished by accessing the list at a particular index number. Lists are zero-indexed, meaning that the first element in a list is the "zeroith" element in the list.

In the previous tasks we've used the interactive interpreter to run Python commands, this is OK for testing and troubleshooting simple tasks, but most Python development is done in flat text files with the ".py" extension.

Open the Atom text editor by clicking on the "Activities" button on the on the menu bar. Type "atom" in the provided search field, and then click on the Atom icon to launch it.



Close all the tabs that Atom opens other than the "untitled" tab. Highlight the top menu bar, click "File", "Save As". Click on the "ubuntu" folder on the left navigation pane, we'll save all of the files for this course here. Save your file as "learninglists.py" -- make sure you have the ".py" extension!



The first thing we need to do is to create a list for us to work with. If you'll recall, lists live inside square brackets ("[" and "]"). Elements within a list are separated by commas within the brackets. Let's create a list that contains the integers "1", "2", and "3".

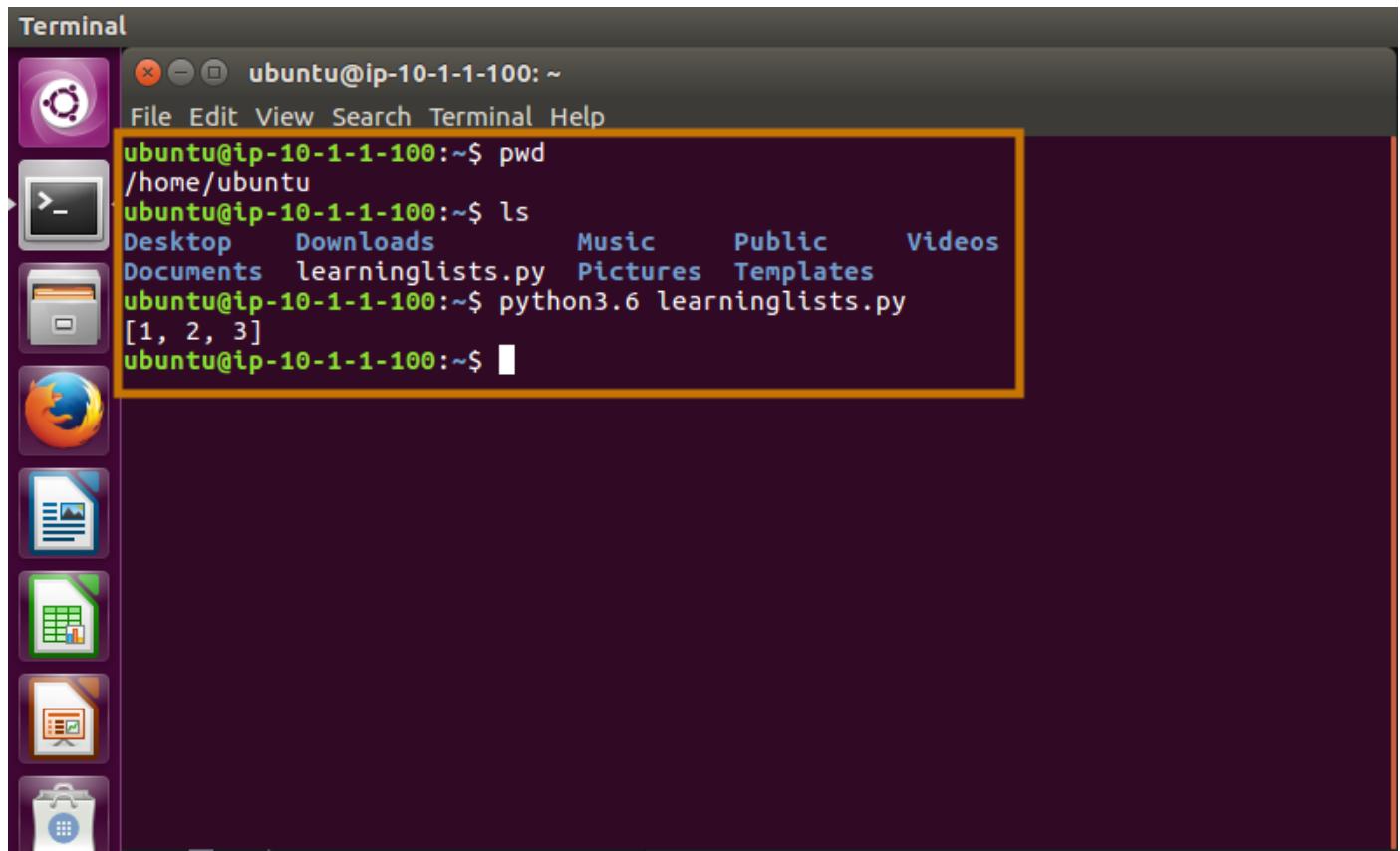
```
my_list = [1, 2, 3]
```

Add a line to print your list, just to make sure everything is working as desired:

```
print(my_list)
```

Save your file. Open another terminal window, it should already be open to the correct directory. Run your script:

```
python3.6 my_file.py
```



If all went as planned, your script should have successfully executed and printed its contents to standard out. If it didn't, take a look at the error and see if you can fix it -- tip remember that Python is very sensitive to white space/formatting! If you run into issues and can't fix it, ask your instructor for help.

Now that we have a basic script working in which we build our first list, we need to understand how we can work with that list. In the previous step we printed the list out, this is OK, but what if we only wanted to access a single element in our list? As outlined, list elements can be accessed by their index -- integer number of their position in the list. This is done by accessing the list and then defining which index you would like to access in square brackets -- ex. `my_list[1]`. Try to add a line to your script to print out the first item in your list (the integer "1").

Did it work? Did you get the number "1" to print out, or did your script print "2"?

```
print(my_list[0])
```

Remember that lists are zero-indexed, so in order to access the integer "1" in our list, we must access the "zeroith" index of our list.

Try to print only the integer "2" from your list.

```
learninglists.py
1 my_list = [1, 2, 3]
2 print(my_list)
3 print(my_list[0])
4 print(my_list[1])
5
```

Terminal

The screenshot shows a standard Ubuntu desktop environment. On the left is a dock with various icons for applications like the Dash, Home, and several productivity tools. The main window is a terminal emulator titled "Terminal". The terminal window has a dark background and contains the following text:

```
ubuntu@ip-10-1-1-100:~$ pwd
/home/ubuntu
ubuntu@ip-10-1-1-100:~$ ls
Desktop  Downloads      Music   Public   Videos
Documents learninglists.py Pictures Templates
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
ubuntu@ip-10-1-1-100:~$
```

The output of the second command, "[1, 2, 3]", is highlighted with a yellow rectangular selection.

## Appending Elements to a List

Now that we know how to print and access specific items in our list, we should learn how to add items to our list. As lists are ordered, for now we'll only worry about adding items to the end of our list (inserting items in the middle is a bit more complicated). Python has a method specifically for this, the aptly called "append" method. Let's go ahead and append the word "four" to the end of our list:

```
my_list.append("four")
print(my_list)
```

Run your script again to make sure that it successfully. Also notice that we have now stored a string in the same list that we were storing integers in. This is one of the great things about lists -- we can store any type of data that we need to in them!

The screenshot shows a terminal window with two main sections. The top section displays a Python script named 'learninglists.py' with line numbers 1 through 7. Lines 5 and 6, which contain the code 'my\_list.append("four")' and 'print(my\_list)', are highlighted with a yellow box. The bottom section shows the terminal interface with the command 'ubuntu@ip-10-1-1-100:~\$'. It lists directory contents ('Desktop', 'Downloads', 'Music', 'Public', 'Videos', 'Documents', 'learninglists.py', 'Pictures', 'Templates'), runs the script ('python3.6 learninglists.py'), and shows the output '[1, 2, 3]'. It then runs the script again ('python3.6 learninglists.py') and shows the output '[1, 2, 3]'. Finally, it runs the script again ('python3.6 learninglists.py') and shows the output '[1, 2, 3, 'four']'. The last command entered is 'ubuntu@ip-10-1-1-100:~\$ %'

```
learninglists.py
1 my_list = [1, 2, 3]
2 print(my_list)
3 print(my_list[0])
4 print(my_list[1])
5 my_list.append("four")
6 print(my_list)
7

ubuntu@ip-10-1-1-100:~%
File Edit View Search Terminal Help
ubuntu@ip-10-1-1-100:~$ pwd
/home/ubuntu
ubuntu@ip-10-1-1-100:~$ ls
Desktop Downloads Music Public Videos
Documents learninglists.py Pictures Templates
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
[1, 2, 3, 'four']
ubuntu@ip-10-1-1-100:~$ %
```

## Deleting List Elements

Great, so we can add to a list, but what if we want to remove things from it? There are several ways to go about doing this, but for now we'll only concern ourselves with the `del` method. The `del` function is used very much in the same way that we used the `print` function on our list -- by accessing a particular index of the list. Let's go ahead and delete the second element in our list.

```
del my_list[2]
print(my_list)
```

Rerun your script and see what happens. Did you delete the list element that you expected would be deleted?

```
learninglists.py
1 my_list = [1, 2, 3]
2 print(my_list)
3 print(my_list[0])
4 print(my_list[1])
5 my_list.append("four")
6 print(my_list)
7 del my_list[2]
8 print(my_list)
9
```

```
ubuntu@ip-10-1-1-100: ~
File Edit View Search Terminal Help
ubuntu@ip-10-1-1-100:~$ pwd
/home/ubuntu
ubuntu@ip-10-1-1-100:~$ ls
Desktop Downloads Music Public Videos
Documents learninglists.py Pictures Templates
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
[1, 2, 3, 'four']
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
[1, 2, 3, 'four']
[1, 2, 'four']
ubuntu@ip-10-1-1-100:~$
```

## Working with Nested Lists

Recall that lists in Python can contain any *type* object that Python contains -- thus far you've configured your list to contain integers (the numbers 1, 2, 3) and a string (the word "four"). Lists can also contain other lists (just like the movie Inception if you're a movie nerd!). To test this out, let's add some code to our script to create a second list.

```
nested_list = []
```

Great, let's put some stuff into our second list -- you can put whatever you want in there, but for purposes of the example we'll put two integers and two strings.

```
nested_list.append(123)
nested_list.append(22)
nested_list.append('ntp')
nested_list.append('ssh')
```

Feel free to re-run your script (and print out your nested list) to make sure everything is going according to plan!

Now that we have a second list, how would we go about adding it to our "parent" list? Since a list treats all of its elements the same, we can just do exactly what we've been doing up till this point and append it.

```
my_list.append(nested_list)
print(my_list)
```

The output shouldn't surprise you at this point -- we have our "parent" list with all of its elements in order, and as one of its elements (the last one) we have our nested list, with all of its elements in their proper order.

```
learninglists.py

1 my_list = [1, 2, 3]
2 print(my_list)
3 print(my_list[0])
4 print(my_list[1])
5 my_list.append("four")
6 print(my_list)
7 del my_list[2]
8 print(my_list)
9 nested_list = []
10 nested_list.append(123)
11 nested_list.append(22)
12 nested_list.append('ntp')
13 nested_list.append('ssh')
14 my_list.append(nested_list)
15 print(my_list)

ubuntu@ip-10-1-1-100:~ File Edit View Search Terminal Help
[1, 2, 3]
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
[1, 2, 3, 'four']
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
[1, 2, 3, 'four']
[1, 2, 'four']
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
[1, 2, 3, 'four']
[1, 2, 'four']
[1, 2, 'four', [123, 22, 'ntp', 'ssh']]
ubuntu@ip-10-1-1-100:~$
```

OK, so now you know how to add a nested list to a list, but what if you wanted to access the elements of that nested list? You can go about that in pretty much the same way as accessing a "regular" list -- we first need to know at what index our nested list "lives". Looking back at the output from the last time you ran and printed your list -- what index contains your nested list? If you've followed along exactly, that should be the third (remember, zero-indexed!) item in your list. Let's print that out just to confirm.

```
print(my_list[3])
```

Great, so no we are accessing (printing in this case) our nested list, how would we go about accessing the second (the word "ntp") item of this list? we could do something like this: "print(my\_list[2])" -- but that won't work because if we simply access the second element of our parent list we'll just print out the number "four". We want to access our nested list instead. So, we access the parent list at the appropriate index, and then we can simply access the appropriate index of the item we are accessing -- that may be a bit confusing, so let's just go ahead and add a line of code to test it out.

```
print(my_list[3][2])
```

Run your script again, and see what happens.

```
learninglists.py
1 my_list = [1, 2, 3]
2 print(my_list)
3 print(my_list[0])
4 print(my_list[1])
5 my_list.append("four")
6 print(my_list)
7 del my_list[2]
8 print(my_list)
9 nested_list = []
10 nested_list.append(123)
11 nested_list.append(22)
12 nested_list.append('ntp')
13 nested_list.append('ssh')
14 my_list.append(nested_list)
15 print(my_list)
16 print(my_list[3])
17 print(my_list[3][2])
ubuntu@ip-10-1-1-100:~
```

File Edit View Search Terminal Help

```
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
[1, 2, 3, 'four']
[1, 2, 'four']
[1, 2, 'four', [123, 22, 'ntp', 'ssh']]
[123, 22, 'ntp', 'ssh']
ntp
ubuntu@ip-10-1-1-100:~$
```

Hopefully the final line of output from your script is the word "ntp". This syntax worked because the third item of our parent list happens to be a list, let's try this syntax on a different element in our parent list, the first element of the zeroith item:

```
print(my_list[0][1])
```

Uhoh! "TypeError: 'int' object is not subscriptable" -- what does this mean?? Under the covers an item in Python is "subscriptable" if it has a method implemented called **getitem()** -- at the moment, in simpler terms -- a "subscript-able" item is something that contains more than one thing. An integer is just a single thing -- whatever the value of the integer is. A list very clearly contains more than one "thing" -- each item in the list is a thing. What about a string? We can test out this theory by attempting to access the n'th position of a string in our list. First, delete or comment out (with a "#" at the start of the line) that last line that is causing the TypeError, then try to access the "o" in the word four (the third item -- so second position -- in our parent list).

```
print(my_list[2][1])
```

The screenshot shows a terminal window on an Ubuntu system. The code in the terminal is:

```
15 print(my_list)
16 print(my_list[3])
17 print(my_list[3][2])
18 # print(my_list[0][1])
19 print(my_list[2][1])
```

The line `# print(my_list[0][1])` is highlighted with a yellow box. The terminal output shows:

```
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
[1, 2, 3, 'four']
[1, 2, 'four']
[1, 2, 'four', [123, 22, 'ntp', 'ssh']]
[123, 22, 'ntp', 'ssh']
ntp
o
```

The terminal prompt is `ubuntu@ip-10-1-1-100:~$`.

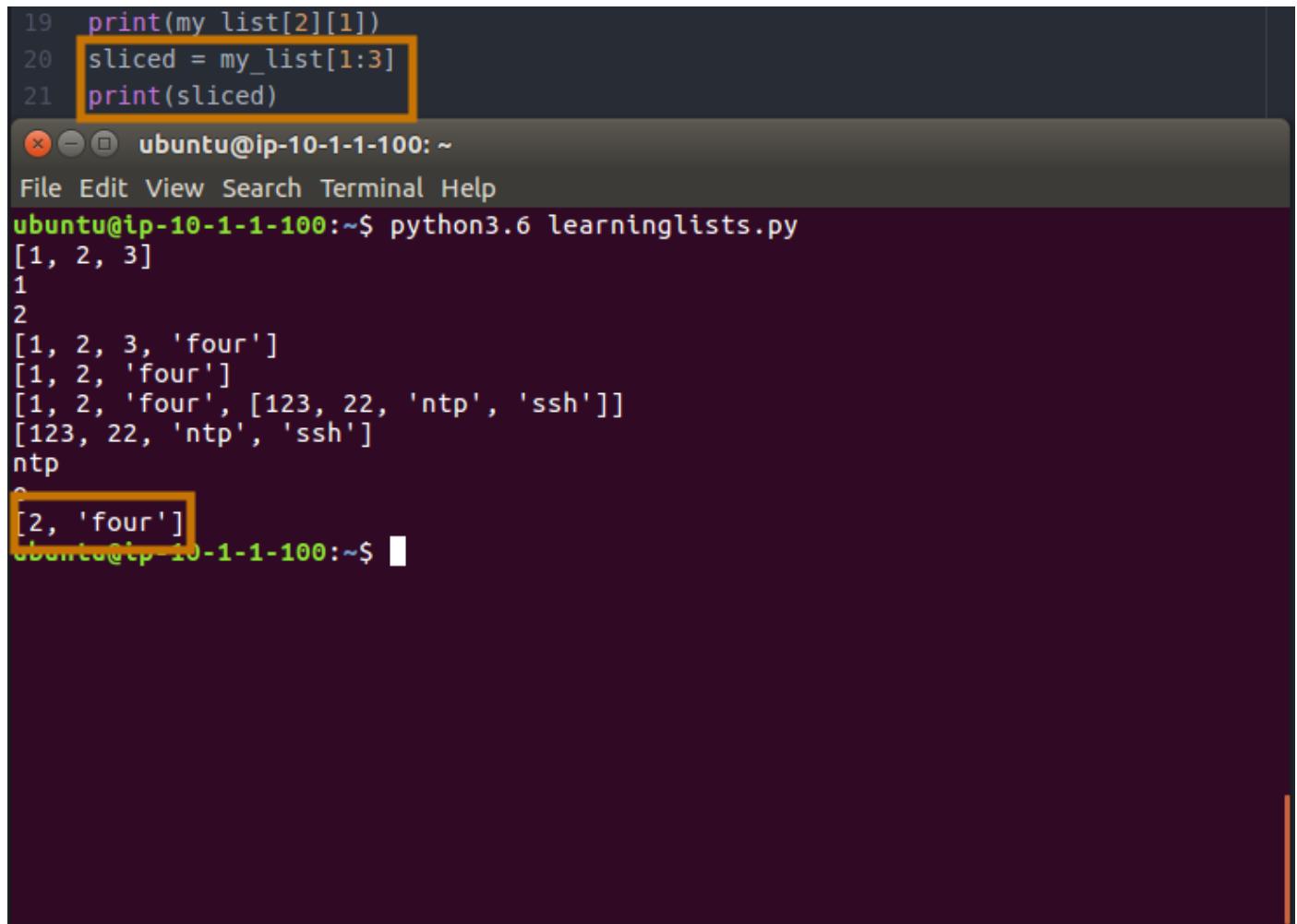
Guess that worked! So lists and strings are "subscriptable" it turns out! The implementation details of this under the covers isn't critical while you're first getting started -- but it is important that you are at least able to understand why you are able to access "indexes" of strings and lists but not integers.

## Slicing

Now that you understand which objects in Python are "subscriptable" its time to learn about one more way to manipulate them: slicing. Slicing is a method for accessing a sub-set of items in a list or a string (or some other types we haven't covered yet). If you've followed along, our list should look something like this at this point: "[1, 2, 'four', [123, 22, 'ntp', 'ssh']]". -- how would you go about retrieving only the middle two items in our list (elements one and two)? Python gives us a handy tool for this with the slicing logic. Add the following lines to your script then run it to see what the output is.

```
sliced = my_list[1:3]
print(sliced)
```

Weird, huh? Slicing acts kind of like accessing a string or a list by elements -- the first digit that we put in the square brackets training our subscriptable item is the first element in the list that we would like to "start our slice" at -- we then have a colon, and then a number which represents the item immediately following the last item we want in our "slice".



```
19 print(my_list[2][1])
20 sliced = my_list[1:3]
21 print(sliced)

ubuntu@ip-10-1-1-100:~
```

File Edit View Search Terminal Help

```
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
[1, 2, 3, 'four']
[1, 2, 'four']
[1, 2, 'four', [123, 22, 'ntp', 'ssh']]
[123, 22, 'ntp', 'ssh']
ntp
[
[2, 'four']

ubuntu@ip-10-1-1-100:~$
```

Slicing can also be used to grab just the beginning or end of a subscriptable item. In the interpreter or at the end of your script, create a variable and assign it the value of "ip address". Using slicing, let's slice off just the "ip" part of our string:

```
slice_me = "ip address"
sliced = slice_me[:2]
print(sliced)
```

The screenshot shows a terminal window with the following content:

```
22 slice_me = "ip address"
23 sliced = slice_me[:2]
24 print(sliced)
ubuntu@ip-10-1-1-100:~$ python3.6 learninglists.py
[1, 2, 3]
1
2
[1, 2, 3, 'four']
[1, 2, 'four']
[1, 2, 'four', [123, 22, 'ntp', 'ssh']]
[123, 22, 'ntp', 'ssh']
ntp
o
[2, 'four']
ip
ubuntu@ip-10-1-1-100:~$
```

The code at the top (lines 22-24) is highlighted with a yellow box. The output of the script is shown below the command line.

Ok, we got "ip" to be sliced off, but why? Kind of weird syntax to get the first two elements of our string. You can think of slicing as follows:

```
[start:end - 1]
[start:]
[:end]
```

Huh?! The first example above is what you've already done when you sliced out the two "middle" items from your list: [start at this index number : go up to this index number, but don't include it].

The third example we just used to snag the "ip" part of our new string: [start from the beginning (you don't have to put anything here, or you could simply put 0) : up to this index, but don't include it].

The middle example is just the opposite of the third: [start at this index : go to the end]

Slicing can get more complicated than this, but for just starting out, and for most "network"/"systems" related tasks this is a great starting point!

## Working with Dicts

Lists are great, however sometimes you need to be able to access data in a more structured manner -- i.e. retrieve some data by its UUID (universally unique ID) as opposed to trying to recall which element in a list

you are hoping to access. Dictionaries give us exactly this capabilities. Dictionaries, sometimes called hash tables, are a key/value store.

## Creating a Dictionary

Create a new text document with a .py extension in Atom, save the file as "learningdicts.py" in the home directory we used previously, we'll be working out of this file for the task.

Dictionaries are identified by "curly braces" ("{" and "}"). Create a new empty dictionary object, this should be familiar as it is very similar to how we created a list, only now with curly braces instead of square brackets.

```
my_dictionary = {}
```

Let's confirm that we did in fact create a dictionary by asking Python to tell us what the type of the variable "my\_dictionary" is.

```
print(type(my_dictionary))
```

Run your script to validate that the type of your variable is what we expect it to be.

```
learninglists.py          learningdicts.py
1 my_dictionary = {}      1
2 print(type(my_dictionary))  2
3
ubuntu@ip-10-1-1-100:~  3
File Edit View Search Terminal Help
ubuntu@ip-10-1-1-100:~$ python3.6 learningdicts.py
<class 'dict'>
ubuntu@ip-10-1-1-100:~$
```

Now that we have a dictionary object we need to go ahead and put some data into it.

We add items to a dictionary in a similar way as we do to a list (but not quite the same!) – first we need to tell Python what position in the dictionary we want to add to. In a dict, the position is the "key" (as opposed to an "index" in a list). Let's say that a "key" in our new dictionary is "gigE0" and we want to assign it a "value" of "Link to ISP" -- how do you think we would do that?

```
my_dictionary["gigE0"] = "Link to ISP"
```

Go ahead and try that out and add a print statement to print the contents of your dictionary variable to see what it looks like.

Once you've got that working, add a three more fake interfaces and descriptions so you have some data in your dictionary to work with. Print it out when you're done to get the feel for the structure of a dictionary.

The screenshot shows a terminal window with two tabs: 'learninglists.py' and 'learningdicts.py'. The 'learningdicts.py' tab is active and contains the following Python code:

```
1 my_dictionary = {}
2 print(type(my_dictionary))
3 my_dictionary["gigE0"] = "Link to ISP"
4 my_dictionary["gigE1"] = "DNS is the root of all problems"
5 my_dictionary["gigE2"] = "An IPv4 address walks into the bar and yells, 'Bartender!"
6 my_dictionary["gigE3"] = "You know the thing about NTP jokes? It's all about the t
7 print(my_dictionary)
8
```

Below the code, the terminal prompt is 'ubuntu@ip-10-1-1-100: ~'. The user runs the script with 'python3.6 learningdicts.py' and prints the type of the variable. Then they run it again to see the dictionary output:

```
ubuntu@ip-10-1-1-100:~$ python3.6 learningdicts.py
<class 'dict'>
ubuntu@ip-10-1-1-100:~$ python3.6 learningdicts.py
<class 'dict'>
{'gigE0': 'Link to ISP', 'gigE1': 'DNS is the root of all problems', 'gigE2': "A
n IPv4 address walks into the bar and yells, 'Bartender! Give me a cider, I'm ex
hausted!", 'gigE3': "You know the thing about NTP jokes? It's all about the timi
ng!"}
```

## Accessing a Dictionary

Now that we've created a dictionary we need to know how to access the value for a particular key. Recall that dictionaries are not indexed numerically, but instead the "key" is used for indexing. Try to print out the value of the key "gigE0".

```
print(my_dictionary["gigE0"])
```

Notice how the syntax is very similar to a list, but we simply access the dictionary via the name of the key.

The screenshot shows a terminal window titled "ubuntu@ip-10-1-1-100:~". It displays two Python files: "learninglists.py" and "learningdicts.py". The "learninglists.py" file contains a list of strings. The "learningdicts.py" file defines a dictionary with four key-value pairs. The terminal output shows the execution of "learningdicts.py" three times, each time printing the dictionary's contents and then the value associated with the key "gigE0". The line "print(my\_dictionary['gigE0'])" is highlighted with a yellow box.

```
learninglists.py | learningdicts.py
1 my_dictionary = []
2 print(type(my_dictionary))
3 my_dictionary["gigE0"] = "Link to ISP"
4 my_dictionary["gigE1"] = "DNS is the root of all problems"
5 my_dictionary["gigE2"] = "An IPv4 address walks into the bar and yells, 'Bartender! Give me a cider, I'm exhausted!', 'gigE3': "You know the thing about NTP jokes? It's all about the timing!"}
6 my_dictionary["gigE3"] = "You know the thing about NTP jokes? It's all about the timing!"
7 print(my_dictionary)
8 print(my_dictionary['gigE0'])
9
ubuntu@ip-10-1-1-100:~$ python3.6 learningdicts.py
<class 'dict'>
ubuntu@ip-10-1-1-100:~$ python3.6 learningdicts.py
<class 'dict'>
{'gigE0': 'Link to ISP', 'gigE1': 'DNS is the root of all problems', 'gigE2': "A n IPv4 address walks into the bar and yells, 'Bartender! Give me a cider, I'm ex hausted!", 'gigE3': "You know the thing about NTP jokes? It's all about the timi ng!"}
ubuntu@ip-10-1-1-100:~$ python3.6 learningdicts.py
<class 'dict'>
{'gigE0': 'Link to ISP', 'gigE1': 'DNS is the root of all problems', 'gigE2': "A n IPv4 address walks into the bar and yells, 'Bartender! Give me a cider, I'm ex hausted!", 'gigE3': "You know the thing about NTP jokes? It's all about the timi ng!"}
Link to ISP
ubuntu@ip-10-1-1-100:~$
```

## What else can go in Dictionaries?

Much like lists, dictionaries can contain other data types -- including lists and other dictionaries. Create a small list of whatever data you'd like, as well as another dictionary with at least one key/value pair.

```
my_list = [3, 2, 1]
my_other_dictionary = {}
my_other_dictionary["thisisakey"] = "thisisavalue"
```

Now store your newly created variables in your dictionary -- use whatever names you would like as they "key" for these entries.

```
my_dictionary["nested_list"] = my_list
my_dictionary["nested_dict"] = my_other_dictionary
```

Print out your parent dictionary and see what the structure looks like. Kind of like the nested lists from before, right? Dictionaries are arguably a little more "structured" and are often easier to work with because you don't need to recall where in a list you've stored a particular item.

The screenshot shows two code files: `learninglists.py` and `learningdicts.py`. The `learningdicts.py` file contains the following code:

```
1 my_dictionary = {}
2 print(type(my_dictionary))
3 my_dictionary["gigE0"] = "Link to ISP"
4 my_dictionary["gigE1"] = "DNS is the root of all problems"
5 my_dictionary["gigE2"] = "An IPv4 address walks into the bar and yells, 'Bartender'
6 my_dictionary["gigE3"] = "You know the thing about NTP jokes? It's all about the t
7 print(my_dictionary)
8 print(my_dictionary['gigE0'])
9 my_list = [3, 2, 1]
10 my_other_dictionary = {}
11 my_other_dictionary["thisisakey"] = "thisisavalue"
12 my_dictionary["nested_list"] = my_list
13 my_dictionary["nested_dict"] = my_other_dictionary
14 print(my_dictionary)
15
```

The terminal window below shows the execution of `learningdicts.py`:

```
ubuntu@ip-10-1-1-100:~$ python3.6 learningdicts.py
<class 'dict'>
{'gigE0': 'Link to ISP', 'gigE1': 'DNS is the root of all problems', 'gigE2': "A
n IPv4 address walks into the bar and yells, 'Bartender! Give me a cider, I'm ex
hausted!", 'gigE3': "You know the thing about NTP jokes? It's all about the timi
ng!"}
Link to ISP
{'gigE0': 'Link to ISP', 'gigE1': 'DNS is the root of all problems', 'gigE2': "A
n IPv4 address walks into the bar and yells, 'Bartender! Give me a cider, I'm ex
hausted!", 'gigE3': "You know the thing about NTP jokes? It's all about the timi
ng!", 'nested_list': [3, 2, 1], 'nested_dict': {'thisisakey': 'thisisavalue'}}
ubuntu@ip-10-1-1-100: ~
```

## Accessing nested Dictionaries

Now that we've stored another dictionary in our main dictionary, how do we go about accessing the variables that we've stored in there? If you guessed pretty much the same as we did with lists, you've guessed correctly!

Print out the value of a key in your nested dictionary.

```
print(my_dictionary["nested_dict"]["thisisakey"])
```

The screenshot shows a terminal window with two tabs at the top: "learninglists.py" and "learningdicts.py". The "learningdicts.py" tab is active, displaying the following Python code:

```
1 my_dictionary = {}
2 print(type(my_dictionary))
3 my_dictionary["gigE0"] = "Link to ISP"
4 my_dictionary["gigE1"] = "DNS is the root of all problems"
5 my_dictionary["gigE2"] = "An IPv4 address walks into the bar and yells, 'Bartender! Give me a cider, I'm exhausted!'"
6 my_dictionary["gigE3"] = "You know the thing about NTP jokes? It's all about the timing!"
7 print(my_dictionary)
8 print(my_dictionary['gigE0'])
9 my_list = [3, 2, 1]
10 my_other_dictionary = {}
11 my_other_dictionary["thisisakey"] = "thisisavalue"
12 my_dictionary["nested_list"] = my_list
13 my_dictionary["nested_dict"] = my_other_dictionary
14 print(my_dictionary)
15 print(my_dictionary["nested_dict"]["thisisakey"])
16
```

The command `print(my_dictionary["nested_dict"]["thisisakey"])` is highlighted with a yellow box.

The terminal output below the code shows the execution of the script:

```
ubuntu@ip-10-1-1-100:~$ python3.6 learningdicts.py
<class 'dict'>
{'gigE0': 'Link to ISP', 'gigE1': 'DNS is the root of all problems', 'gigE2': "An IPv4 address walks into the bar and yells, 'Bartender! Give me a cider, I'm exhausted!", 'gigE3': "You know the thing about NTP jokes? It's all about the timing!"}
Link to ISP
{'gigE0': 'Link to ISP', 'gigE1': 'DNS is the root of all problems', 'gigE2': "An IPv4 address walks into the bar and yells, 'Bartender! Give me a cider, I'm exhausted!", 'gigE3': "You know the thing about NTP jokes? It's all about the timing!", 'nested_list': [3, 2, 1], 'nested_dict': {'thisisakey': 'thisisavalue'}}
thisisavalue
ubuntu@ip-10-1-1-100:~$
```

## Conditionals

Now that we've got the basic data structures out of the way, we need to learn how to apply logic to our code, in Python we can do that with "Conditional" statements.

### Introducing the "if" Statement

Create a new text document called "learningconditionals.py" in the home directory we used previously, we'll be working out of this file for the task.

In Python the "if" statement is used to evaluate conditions to see if they are True. IF something is True, do something. That implies that IF something is *not* True, we'll simply continue on without executing the code that applies if the statement were True.

To illustrate this, let's build a simple "if" statement to check to see if an integer that we create is positive.

```
my_integer = 443
if my_integer > 0:
    print("Hey, that looks like its a positive number!")
```

Run your code, do you get any errors? Does your code indicate that you entered a positive number?

The screenshot shows a terminal window with three tabs at the top: 'learninglists.py', 'learningdicts.py', and 'learningconditionals.py'. The 'learningconditionals.py' tab is active. Below the tabs, there is a code editor with the following Python script:

```
1 my_integer = 443
2 if my_integer > 0:
3     print("Hey, that looks like a positive number!")
4
```

Below the code editor is a terminal window titled 'ubuntu@ip-10-1-1-100:~'. The terminal shows the command `python3.6 learningconditionals.py` being run, followed by the output: 'Hey, that looks like a positive number!'. The entire terminal window is highlighted with a yellow box.

Try to change your integer to a negative number, what happens?

What happens if you change your variable "my\_integer" to a string?

```
learninglists.py          learningdicts.py          learningconditionals.py
1 my_integer = 443        2 my_integer > 0:           3 print("Hey, that looks like a positive number!")
2 if my_integer > 0:      3     print("Hey, that looks like a positive number!")
3     print("Hey, that looks like a positive number!")
4
5 my_integer = -443       6 if my_integer > 0:         7     print("Hey, that looks like a positive number!")
6 if my_integer > 0:      8
7     print("Hey, that looks like a positive number!")
8
9 my_integer = 'mmmm tacos!' 10 if my_integer > 0:        11     print("Hey, that looks like a positive number!")
10 if my_integer > 0:      12
11     print("Hey, that looks like a positive number!")
12
```

```
ubuntu@ip-10-1-1-100:~$ python3.6 learningconditionals.py
Hey, that looks like a positive number!
Traceback (most recent call last):
  File "learningconditionals.py", line 10, in <module>
    if my_integer > 0:
TypeError: '>' not supported between instances of 'str' and 'int'
ubuntu@ip-10-1-1-100:~$
```

Create a new variable and assign it the value "you call that a string?"

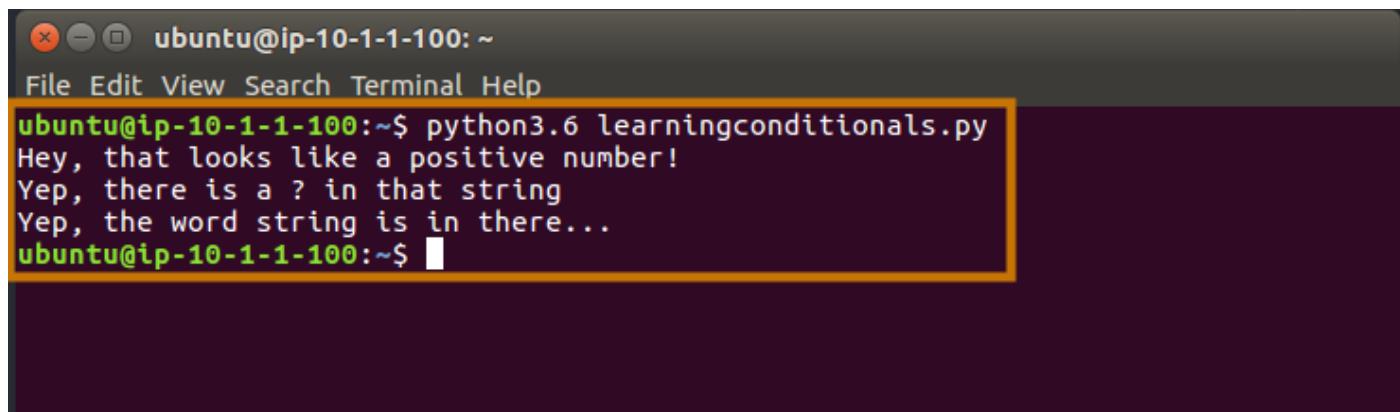
Try to write an "if" statement that determines if a question mark ("?") is **in** (Tip, Internet search "Python in" -- it may help!) a string -- if it is, print "Yep, there is a ? in that string"

Note: From here on there will be less detailed instructions, don't hesitate to ask your instructor if you need a hand!

Now, add another "if" statement to see if a colon ":" is in that variable, print out "Yep, there is a colon...".

And one more "if" statement to see if the word "string" is in that variable, print out "Yep, the word string is in there...".

What happens when you run the code? If your script works as it should, print out "Yep, there is a ? in that string", because the first "if" statement is True. It should then "skip" the second "if" statement because there is no colon in your string, finally it should print "Yep, the word string is in there..." because of course that is true as well.



A screenshot of a terminal window titled "ubuntu@ip-10-1-1-100: ~". The window shows the following text:

```
ubuntu@ip-10-1-1-100:~$ python3.6 learningconditionals.py
Hey, that looks like a positive number!
Yep, there is a ? in that string
Yep, the word string is in there...
ubuntu@ip-10-1-1-100:~$
```

## Elif

In the last section each of the "if" statements you created had to be evaluated. What happens if you would rather stop processing once a condition is met? Using the previous section as an example, what if you wanted to stop evaluating the if statements the first time something was True? Python provides us the "elif" statement for this -- this is a shortened syntax for "else if".

Try to re-factor (adjust or rewrite) your previous code to change the second and third "if" statements to be "elif" statements. Run your code, what changed?

Try to change your variable so that it only evaluates to True on the final "elif" clause, try to change your variable so that it doesn't ever evaluate True -- does your script behave the way you think it should?

**Else**

Python also has an "else" clause to use as a "catch all" for anything that does not evaluate to True on the previous conditionals

If you followed along and your string now no longer evaluates to True at any of the if/elif statements, try to add an "else" statement at the end -- print out "Whoa, we got a catch-all now!"

# While Loops

One of the great parts about learning Python is the ability to use computers do perform repetitive tasks... it turns out computers are much, much better at monotonous things than humans are! One way to do this is through the use of a "while loop" -- a block of code that will continually run *while* some condition is True.

## Build a Countdown Timer

Create a new text document called "learningwhileloops.py" in the home directory we used previously, we'll be working out of this file for the task.

Let's build a very simple block of code that will continually print "Python is the coolest!" *while* the code within the loop is True.

```
while True:  
    print("Python is the coolest!")
```

Save and run your code.

What happens? Is it still going? You can hit "ctrl"+"c" to exit Python. So... why did Python just continually print how cool it is to the screen? It turns out that the code within the while loop was *always* True, so Python continually looped through the while loop over and over and over and over and over...

The screenshot shows a terminal window titled "learningwhileloops.py". The code in the editor is:

```
1 while True:  
2     print("Python is the coolest!")
```

The terminal output shows the string "Python is the coolest!" printed multiple times, indicating the loop is running. The terminal window title is "ubuntu@ip-10-1-1-100: ~". The terminal prompt is "ubuntu@ip-10-1-1-100:~\$". A yellow box highlights the entire code block in the editor and the first few iterations of the print statement in the terminal output.

Let's re-factor this while loop so that it doesn't run forever, and so that it is actually doing something more useful -- we'll turn it into a countdown timer. Create a new variable and assign a value you would like to count down from (10 for example, make sure this is an integer!).

Now, instead of running your while loop "while True", let's change it to be "while [your variable] > 0". What do you think that will do? It will run the block of code within the loop as long as your variable is greater than 0. Try to run your code once more.

Yep, you'll have to break out of the loop again because we never decremented the variable so the value always stayed the same. Recall the "`-=`" operator -- try to use that to decrement your variable after your code prints out "Python is the coolest!".

Run your code again, did that work? If not, what issues are you running into? Ask your instructor if you need some help.

A screenshot of a terminal window titled "ubuntu@ip-10-1-1-100: ~". The window shows a Python script named "learningwhileloops.py" being run. The script contains a while loop that prints the value of the variable "count\_down" and then decrements it by 1. The output shows the count down from 10 to 1.

```
count_down = 10
while count_down > 0:
    print(f"COUNT DOWN = {count_down}")
    count_down -= 1
```

```
ubuntu@ip-10-1-1-100:~$ python3.6 learningwhileloops.py
COUNT DOWN = 10
COUNT DOWN = 9
COUNT DOWN = 8
COUNT DOWN = 7
COUNT DOWN = 6
COUNT DOWN = 5
COUNT DOWN = 4
COUNT DOWN = 3
COUNT DOWN = 2
COUNT DOWN = 1
ubuntu@ip-10-1-1-100:~$
```

## For Loops

Python also has the concept of a "for" loop (like `foreach` in other languages). The for loop will execute a block of code (the code within the loop) for each element of a subscriptable item (for example a list).

### Iterate over lists

Create a new text document called "learningforloops.py" in the home directory we used previously, we'll be working out of this file for the task.

Create a list of at least three items, feel free to put any type of objects in your list (even nested lists or dictionaries!).

The syntax of a for loop is relatively straight forward -- "`for [item] in [iterable]`" is the base syntax. The "iterable" is simply any subscriptable item in Python -- as we've seen this can be a list or a string, but there are others that we haven't covered yet. For now, let's keep things simple and assume that our iterable will be a list or a string. In the provided example "`item`" is essentially a new variable -- as we iterate (loop) over each item in our subscriptable object ("iterable") we assign the value of that item to the variable we provide in the place of "`item`".

To make that real, let's add to your new script and loop over the list you created.

```
for item in [your list here]:  
    print(item)
```

What do you think will be the output of this script? Save and run your script to check it out!

The screenshot shows a terminal window titled "ubuntu@ip-10-1-1-100: ~". The code in the terminal is:

```
learningforloops.py  
1 myitem1 = "Tacos"  
2 myitem2 = "Racecars"  
3 myitem3 = [myitem1, myitem2]  
4  
5 mylist = []  
6 mylist.append(myitem1)  
7 mylist.append(myitem2)  
8 mylist.append(myitem3)  
9  
10 for item in mylist:  
11     print(item)  
12
```

The terminal output is:

```
ubuntu@ip-10-1-1-100:~$ python3.6 learningforloops.py  
Tacos  
Racecars  
['Tacos', 'Racecars']  
ubuntu@ip-10-1-1-100:~$
```

Did it work the way you thought it should? Did you have any nested dicts/lists in your parent list (if not, go back and try that!), did that work the way you thought it would?

A final note about loops and conditionals -- you can combine these concepts by nesting loops, or putting conditional statements at each iteration of a loop. The possibilities are endless, but just remember the Zen of Python (if you forget, open the interpreter and type "import this") -- "Simple is better than complex." and "Flat is better than nested."!

## Importing Modules

One thing that hasn't been covered much yet is that Python has a great many third party modules/libraries. That is there are "plugins" to the base functionality of the language itself. Even though the core language contains tons and tons of functionality it may not contain optimal ways to perform certain tasks, or interact with certain platforms -- third party modules exist to bridge these gaps. In the network/systems world the "requests" library is perhaps the most well known -- in fact it is the single most popular third party Python library! Requests is essentially an HTTP driver for Python -- why would networking and systems folks care about this? Simple,

most vendors are moving towards HTTP based APIs, and requests is one of the best ways to manipulate HTTP with Python! We'll discuss requests more later, but for now let's just focus on modules, and how to import them into a Python script.

Normally you'll need to install third party modules that you wish to import -- for the purposes of this lab that has been done for you. If you wish to do this on your machine later do a safe bet for installing modules is to execute the "pip install [module name]" -- pip (a recursive name for Pip installs Python Packages) is a Python package management system that is widely used; most popular packages will be available via pi.

## Import a Module

In the interactive interpreter or a .py file, let's go ahead and import the requests module.

**Tip:** it is common convention to perform all imports at the "top" of your script.

```
import requests
```

If you aren't using the interpreter, go ahead and run your script. If nothing happens, that's a good thing! That means that Python was able to import your package, and all of the functions that are contained within that package are now available to you.

For kicks, let's try to import a package that doesn't exist (but maybe it should!).

```
import ignwisthecoolest
```

Run your script again -- what happens? You likely saw a "ModuleNotFoundError" -- this is because the package does not exist on the local system. This very well could be a "real" Python package but if it is not installed on the local system Python will still raise this error.

For the purposes of the labs you'll be doing during this course, you can safely assume that all libraries that you need will be installed on your system, but when you go back to writing scripts on your own system, don't forget to install packages!

## Capturing User Input

So far we've written very self-contained scripts -- meaning that all of the logic/input that has been required came from within our script. What happens if you need data from a human? There must be some way to ask a human for data and bring that into your script right?

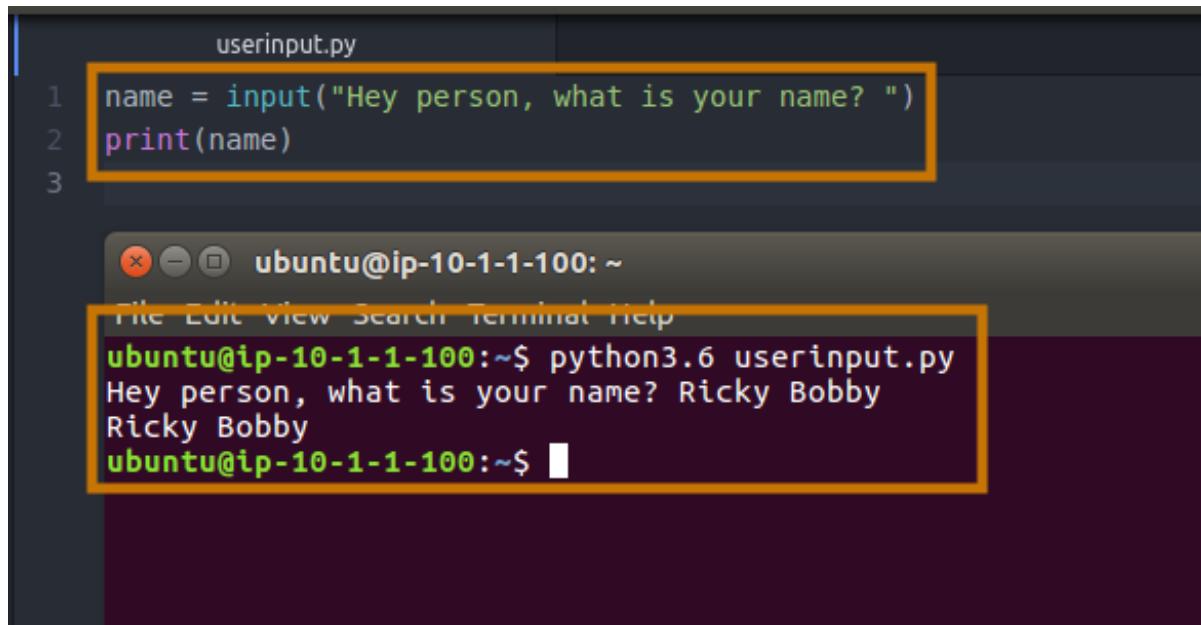
### Input

Create a new text document called "userinput.py" in the home directory we used previously, we'll be working out of this file for the task.

Python has a very simple way for us to acquire input from a person, and it is very aptly named: `input()`! `Input` accepts a string that we can pass to it that Python will display as the prompt for our question. Let's go ahead and try to ask the user for their name, then print it out to make sure we got what we needed:

```
name = input("Hey person, what is your name? ")
print(name)
```

Whoa, way too easy right? Python is a pretty approachable language, it's one of the reasons we love it so much!



The screenshot shows a terminal window on an Ubuntu system. The terminal title is "ubuntu@ip-10-1-1-100: ~". The command entered is "python3.6 userinput.py". The script content is:

```
1 name = input("Hey person, what is your name? ")
2 print(name)
3
```

The output of the script is:

```
Hey person, what is your name? Ricky Bobby
Ricky Bobby
```

For now this was OK, but you'll need to be careful when prompting users for input in the future -- sometimes users will provide input that you didn't expect; they may use all caps, or all lower case, or input special characters, or spaces or whatever else you can think of.

Try to write a script that prompts the user to enter their name over and over again until they enter a string with no integers in it -- this will require you to combine many of the steps that you've gone through so far. Ask your instructor for a hint if you get stuck!

# CheckiO Labs

## Overview and Objectives

In this lab you will use CheckiO to expand on your Python knowledge by solving some fun and interesting tasks!

Objectives:

- Continue to improve your Python skills
- Learn how to leverage some of Python's built in modules

## Introduction

The CheckiO environment provides you with challenges to solve using Python.

The CheckiO labs can be completed from the student Ubuntu VM, or from your local computer as all of these tasks will be contained within a browser window. Instructions are written for the student Ubuntu VM but things should work just the same on your local machine!

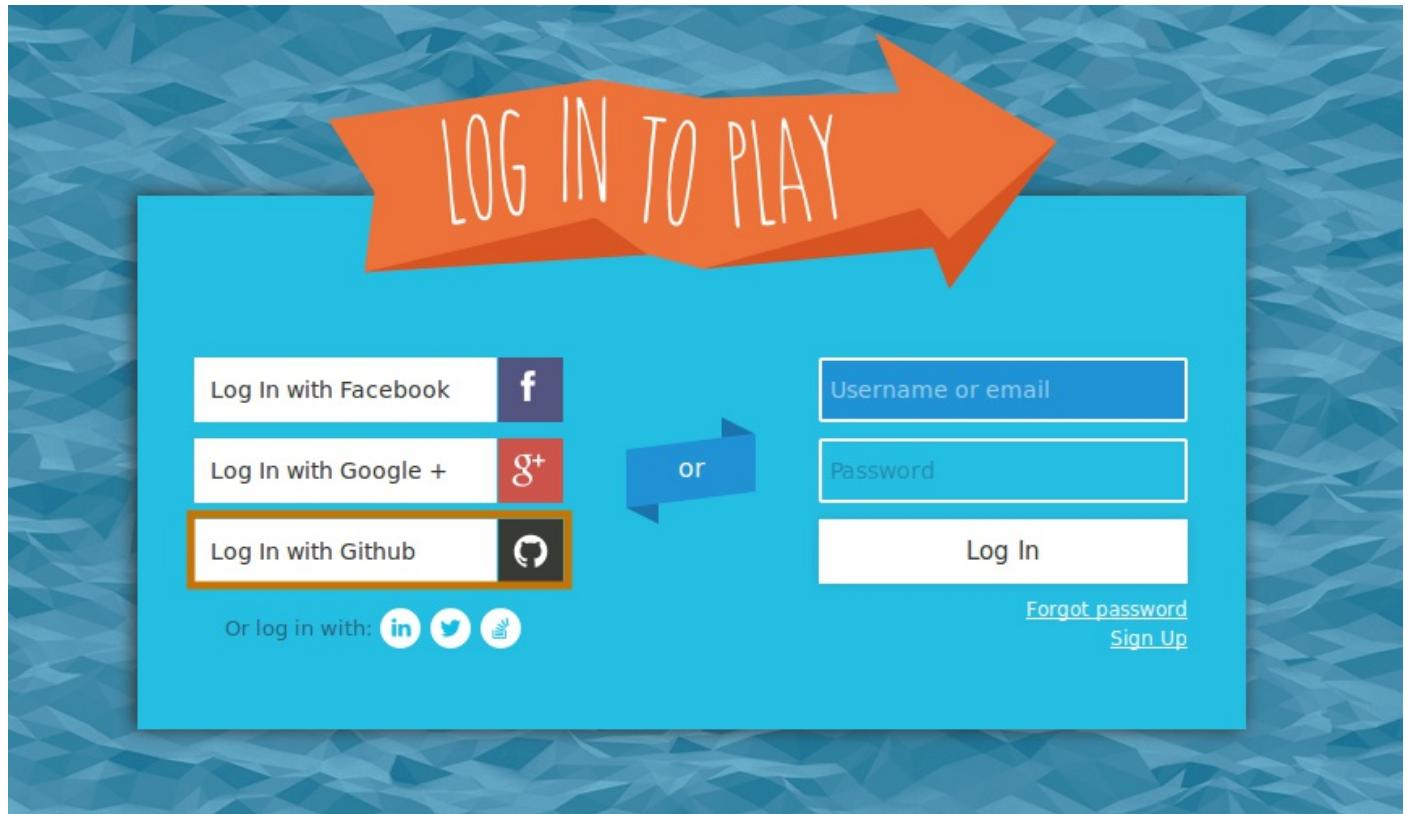
Launch the Firefox browser by clicking on the Firefox icon on the dock.



Navigate to "checkio.org". Click on the "Python" box in the "CheckiO" section as outlined below.



Log in with the Github account you created in earlier in the course (feel free to log in via Facebook or Google if you'd prefer!). You will be redirected to the sign in service you selected, enter your credentials as appropriate.



Log In with Facebook 

Log In with Google + 

Log In with Github 

Or log in with:   

or

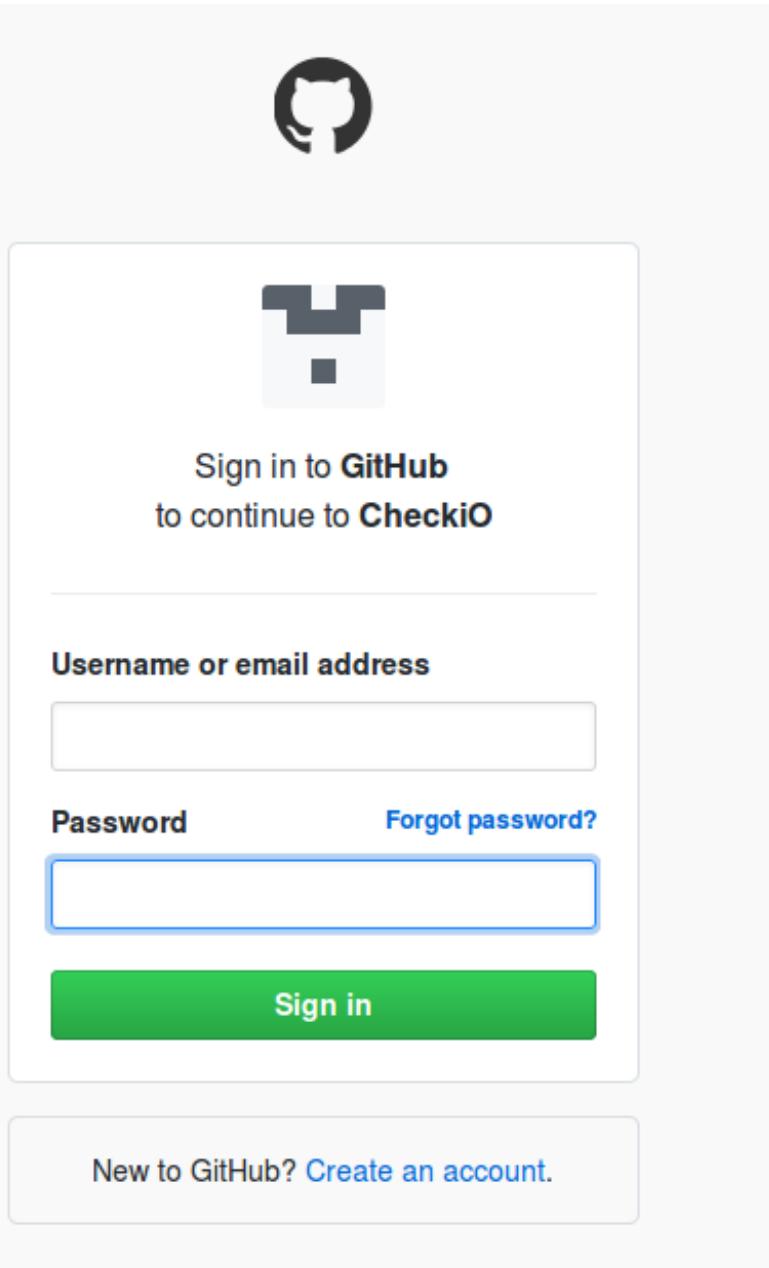
Username or email

Password

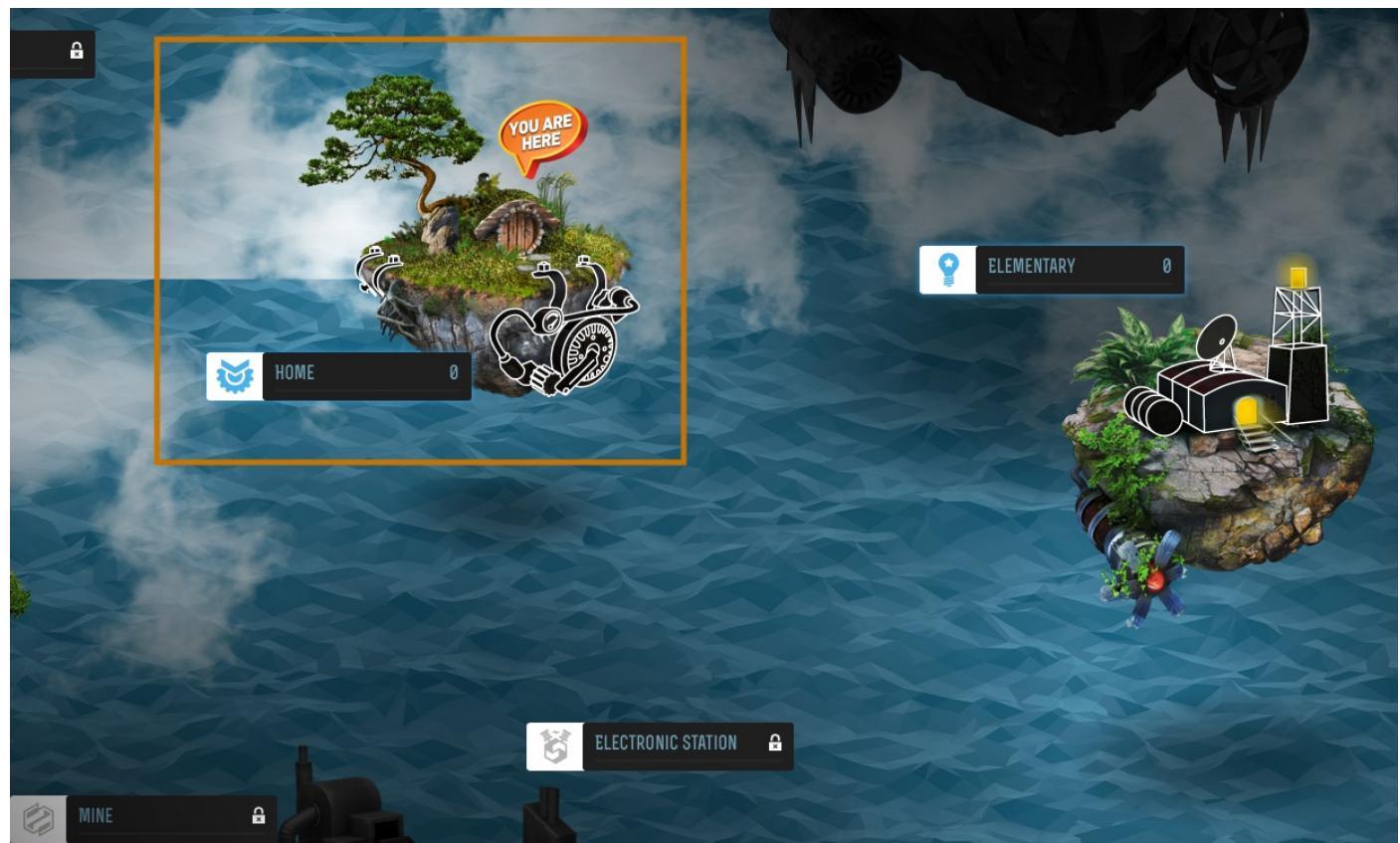
Log In

[Forgot password](#)

[Sign Up](#)



Once logged in you will arrive at the CheckiO home screen -- this is comprised of different "stations", each station has a set of tasks to complete. As you complete tasks you will unlock access to more stations. Click on the "HOME" station as outlined below.



Once you are "in" the station, you are presented a list of tasks. The tasks have a brief description, a difficulty rating, and a tag that indicates the type of things you'll be working with in the task (i.e. ext, statistics, games, etc.).

Click on the "House Password" task.

The screenshot shows a user interface for a programming challenge. At the top left, there's a 'Home' button and a progress bar showing '0 %'. Below this, a welcome message reads: 'Welcome home! Take your time getting all pumped up and ready for the journey that's about to begin.' To the right of the message is a whimsical illustration of a small island with a tree, a house, and pipes.

The main area displays five challenges:

- House Password**: Check the strength of your favorite password. Category: Text. Difficulty: Elementary. A small icon of three stars is highlighted with an orange border.
- The Most Wanted Letter**: Find out which is the most wanted letter. Category: Statistics | text. Difficulty: Elementary.
- Non-unique Elements**: Trim an array down to its non-unique elements. Category: Structures. Difficulty: Elementary.
- Monkey Typing**: Put enough robots in a room with typewriters and they'll produce Shakespeare. Category: Text. Difficulty: Elementary.
- Xs and Os Referee**: Referee Tic-Tac-toe game. Category: Games | structures. Difficulty: Simple.

Each challenge has a circular 'Solve it!' button to its right.

This next screen provides some more details about the task. (for now) We're just using this task to get you familiar with CheckiO, so you don't need to read it all if you don't want to! Click on the "Solve it" button.



## House Password

**Translation:** English (original) ▾

 We have prepared a set of Editor's Choice Solutions. You will see them first after you solve the mission. In order to see all other solutions you should change the filter.



Stephan and Sophia forgot about security and use simple passwords for everything. Help Nikola develop a password security check module. The password will be considered strong enough if its length is greater than or equal to 10 symbols, it has at least one digit, as well as containing one uppercase letter and one lowercase letter in it. The password contains only ASCII latin letters or digits.

**Input:** A password as a string.

**Output:** Is the password safe or not as a boolean or any data type that can be converted and processed as a boolean. In the results you will see the converted results.

**Example:**

```

1 checkio('A1213pokl') == False
2 checkio('bAse730onE') == True
3 checkio('asasasasasasaas') == False
4 checkio('QWERTYqwerty') == False
5 checkio('123456123456') == False
6 checkio('QwErTy911poqqqq') == True

```

**How it is used:** If you are worried about the security of your app or service, you can check your users' passwords for complexity. You can use these skills to require that your users passwords meet more conditions (punctuations or unicode).

**Precondition:**  
`re.match("[a-zA-Z0-9]+", password)`  
`0 < len(password) ≤ 64`

[How to improve this mission?](https://github.com/Bryukh-Checkio-Tasks/checkio-task-house-password.git { 29 }) <https://github.com/Bryukh-Checkio-Tasks/checkio-task-house-password.git { 29 }>

HOME  
0 0 %

**Story** Solve it **Discuss** **Best Solutions** **Rand. Solution** **Get next task**

Share: [!\[\]\(a1654e458897b066477a7e0ddc021d7c\_img.jpg\)](#) [!\[\]\(4d597557c8b49491fd8454c2a24c878d\_img.jpg\)](#)

<https://py.checkio.org/r>

Users attempted: 42709
Users succeeded: 32678
Score for solving: 10
Score for publication: 5
Score for every vote: 5

**Fresh Awesome**

		<b>6</b>	spanchbob
		<b>8</b>	evarome10
		<b>3</b>	deleted

[Become Awesome](#)

The screenshot shows the CheckiO platform interface. At the top, there's a navigation bar with 'CheckiO' logo, 'House Password', 'Run', 'Check', 'Reset code', 'Mission Info...', 'Use local editor', 'Help', and a user profile 'carin'. Below the navigation bar is the code editor window.

**Code Editor:**

```

1 def checkio(data):
2
3     #replace this for solution
4     return True or False
5
6 #Some hints
7 #Just check all conditions
8
9
10 if __name__ == '__main__':
11     #These "asserts" using only for self-checking and not necessary for auto-testing
12     assert checkio('A1213pokl') == False, "1st example"
13     assert checkio('bAse730onE4') == True, "2nd example"
14     assert checkio('asasasasasasaas') == False, "3rd example"
15     assert checkio('QWERTYqwerty') == False, "4th example"
16     assert checkio('123456123456') == False, "5th example"
17     assert checkio('QWeRTy91lpqqqq') == True, "6th example"
18
19     print("Coding complete? Click 'Check' to review your tests and earn cool rewards!")

```

**Check results:**

Stephan and Sophia forgot about security and use simple passwords for everything. Help Nikola develop a password security check module. The password will be considered strong enough if its length is greater than or equal to 10 symbols, it has at least one digit, as well as containing one uppercase letter and one lowercase letter in it. The password contains only ASCII latin letters or digits.

**Input:** A password as a string.  
**Output:** Is the password safe or not as a boolean or any data type that can be converted and processed as a boolean. In the results you will see the converted results.

checkio('A1213pokl') == False  
checkio('bAse730onE4') == True  
checkio('asasasasasasaas') == False  
checkio('QWERTYqwerty') == False  
checkio('123456123456') == False  
checkio('QWeRTy91lpqqqq') == True

**Precondition:**  
re.match("[a-zA-Z0-9]+", password)  
0 < len(password) ≤ 64

**Some hints are available. Would you like to use them?**

I have no idea how to start solving this mission

**Run Output & Console:**

Click on "Run" to view results or Ctrl + /  
Click on "Check" to test your solution or Ctrl + Enter  
Click on "Try it" to play with your code in the console

**Partner:**

 PyCharm.  
The Python IDE for Professional Developers.  
Enjoy productive Python, web and scientific development with PyCharm. Download it now!

In the editor screen you will see two main sections -- at top you will see "def checkio..." -- this is defining a function and is where you will enter your code. In the parentheses following "checkio" you will see what (if any) objects are being passed into your code -- we'll go over this as we walk through the first lab together. The second section "if **name == "main"**" is a common Python convention that basically says if the script is executed (python my\_script.py) it will run the code in this section. In this case, CheckiO will populate this section for you and use it to "assert" (test for Truth) that your function (the first block) returns (outputs) the appropriate response when passed some parameters.

The screenshot shows the CheckiO IDE interface. On the left, there's a code editor with Python code for a password strength checker. The code includes a function `checkio` and several assert statements. A yellow box highlights the first few lines of the function definition and the assert statements. On the right, there's a "Check results" panel with mission details for "House Password". It includes input examples, output descriptions, and a "Precondition" section. Below the main panels is a "Run Output & Console" section with instructions and a "Check" button.

```

1 def checkio(data):
2
3     #replace this for solution
4     return True or False
5
6 #Some hints
7 #Just check all conditions
8
9
10 if __name__ == '__main__':
11     #These "asserts" using only for self-checking and not necessary for auto-testing
12     assert checkio('A1213pokl') == False, "1st example"
13     assert checkio('bAse730onE4') == True, "2nd example"
14     assert checkio('asasasasasasaas') == False, "3rd example"
15     assert checkio('QWERTYqwerty') == False, "4th example"
16     assert checkio('123456123456') == False, "5th example"
17     assert checkio('QWErTy911poqqqq') == True, "6th example"
18     print("Coding complete? Click 'Check' to review your tests and earn cool rewards!")
19

```

Click on the CheckiO icon on the top left of the screen to move back to the "station map"/home screen of Checkio.

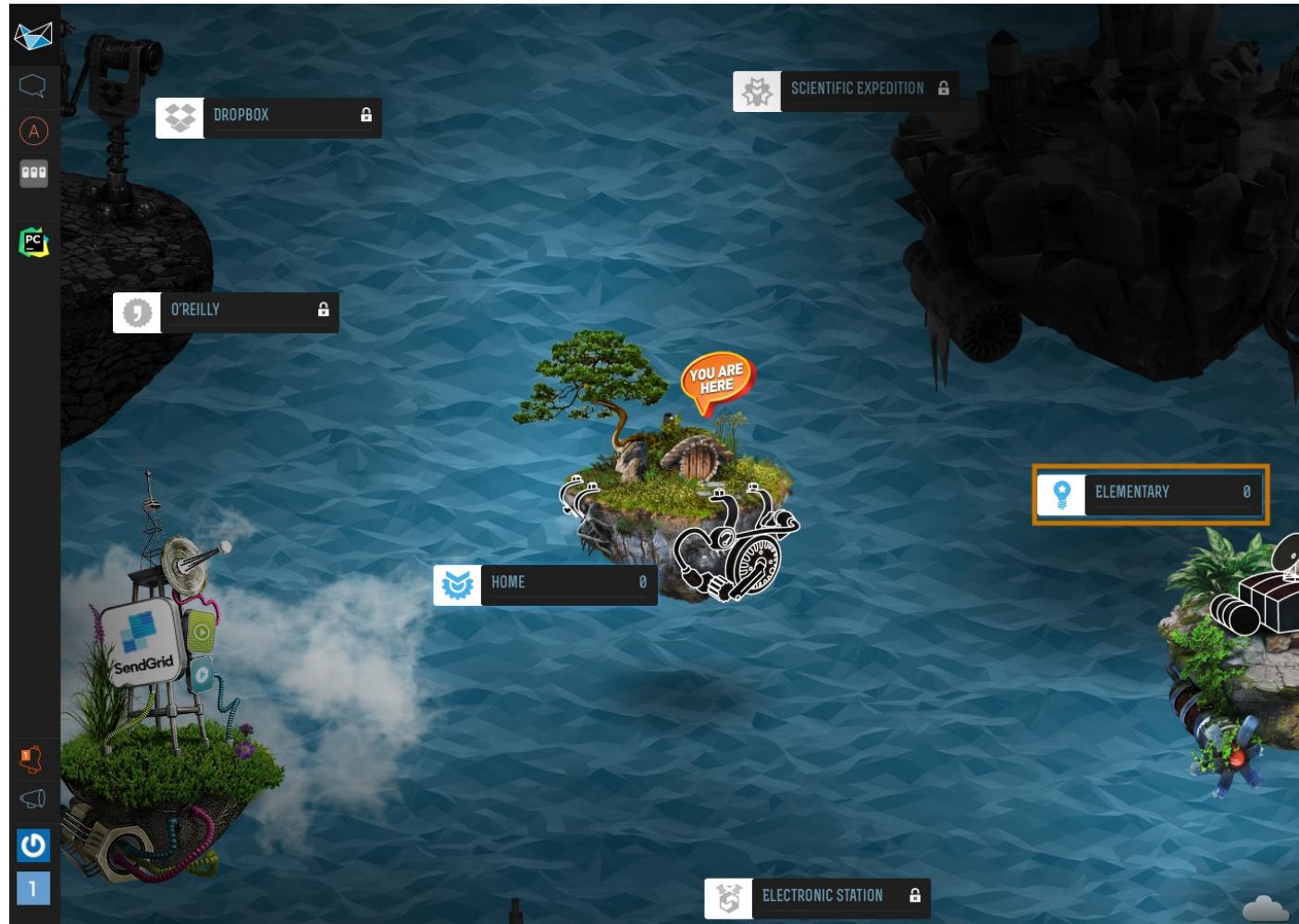
This screenshot shows the CheckiO IDE interface after clicking the icon to return to the home screen. The main area displays the same Python code for the password strength checker. The top navigation bar now shows the station map icon, indicating the user is back on the home screen.

Now that we are jumping into the CheckiO labs, we'll walk through the first two labs, but then you'll be on your own for the remainder! We haven't covered all of the techniques you'll need to complete these labs yet, but

that's part of the fun! Use your peers, use the Internet, and of course ask your instructor if you want some help!

## Three Words

To access the "Three Words" task, click on the "ELEMENTARY" station from the home screen:



Find and select the "Three Words" lab from the list:

The image shows a Scratch workspace interface. On the left, there is a vertical sidebar with various tool icons: a blue envelope, a speech bubble, a letter 'A', a calculator, and a green 'PC' icon. Below these are icons for a magnifying glass, a megaphone, a blue circle with a 'G', and a blue number '1'. The main area displays a list of projects:

- Fizz Buzz**: A word game used to teach robots about division. Level: Elementary. Tags: Numbers, text.
- The Most Numbers**: Determine the difference between the smallest and the biggest numbers. Level: Elementary. Tags: Built-In, numbers.
- Even the last**: How to work with arrays indexes. Level: Elementary. Tags: Numbers, structures.
- Secret Message**: Coding is fun. Heck you could do it too. Ever try that? Code away! Krummy results? Interesting. Onto something else then! Level: Elementary. Tags: Parsing, text.
- Three Words**: How to discern words and numbers. Level: Elementary. Tags: Text.
- Index Power**: What is the power hidden within indexes? Level: Elementary. Tags: Numbers, structures.
- Right to Left**: "Left, right, left, right, left, left, left, left. Your destination is...

A yellow border highlights the 'Three Words' project. The 'Index Power' project has a large '42' icon above its title.

Click on "Solve it" to get started...

In this lab we are provided strings that contain words and/or numbers -- we are asked to determine if there are three words (in this example a word is just a string with no numbers in it, not necessarily a "real" word) consecutively, if so we should return "True", if not, we return "False"

Before writing any code we need to think about the task we're being given -- we're provided a string (the first example is "Hello World hello") that has words and numbers separated by a space. Hmm... separated by a space is an important point! We also know that the values will either be words or integers -- another interesting fact...

Lastly, we know that we need to count -- we know that because we need to determine if we have three "words" in a row. We also need to ensure that we "reset" our counting if we encounter a number as we see in the second "assert" example ("He is 123 man"). OK, let's get started...

A good Internet search at this point could be something like "how can I split a string on a space" -- we'll cheat and jump right to one way to do it -- the split method. The split method takes an argument that defines what value you would like to split a string on -- in our case a space -- it then returns a list of all elements from the original string without the delimiter.

```
split_words = words.split(' ')
print(split_words)
```

Run your code by clicking the "Run" button on the top menu bar. You'll see the output in the bottom section of the screen. Notice that we now are printing a list of each of the elements that made up the original screen we were provided. You'll also notice that we hit an "AssertionError" -- that means we failed the test, but that's OK, we're not done yet!

The screenshot shows the CheckiO IDE interface. At the top, there's a toolbar with icons for CheckiO logo, Three Words, Run, Check, Reset code, Mission Info..., Use local editor, and Help. Below the toolbar is a code editor window containing Python code. Lines 1 through 13 are shown:

```
1 def checkio(words):
2     split_words = words.split(' ')
3     print(split_words)
4
5 #These "asserts" are used only for self-checking and not necessary for auto-testing
6 if __name__ == '__main__':
7     assert checkio("Hello World hello") == True, "Hello"
8     assert checkio("He is 123 man") == False, "123 man"
9     assert checkio("1 2 3 4") == False, "Digits"
10    assert checkio("bla bla bla bla") == True, "Bla Bla"
11    assert checkio("Hi") == False, "Hi"
12    print("Coding complete? Click 'Check' to review your tests and earn cool rewards!")
13
```

Below the code editor is a 'Run Output & Console' window. It contains two sections: 'Run Output' and 'Console'. The 'Run Output' section shows the following text:

```
['Hello', 'World', 'hello']
AssertionError: Hello
<module>, 7
```

The 'Console' section has an input field labeled 'Enter a text:' with the value 'Hi you are over 9000'. There are 'Check' and 'Random' buttons at the bottom.

We now have a list that we can work with, but how do we go about validating that the items in it are not a number? A good search may be "Python how to tell if a string contains only letters". The "isalpha" method does exactly this for us.

```
for word in split_words:
    if word.isalpha():
        print('this is a word not a number!')
```

Run your code again -- now you should be able to glean from the output that we are in fact validating which items in our list are words, now we just need to count!

```

1 def checkio(words):
2     split_words = words.split(' ')
3     print(split_words)
4     for word in split_words:
5         if word.isalpha():
6             print('this is a word, not a number!')
7
8 #These "asserts" using only for self-checking and not necessary for auto-testing
9 if __name__ == '__main__':
10    assert checkio("Hello World hello") == True, "Hello"
11    assert checkio("He is 123 man") == False, "123 man"
12    assert checkio("1 2 3 4") == False, "Digits"
13    assert checkio("bla bla bla bla") == True, "Bla Bla"
14    assert checkio("Hi") == False, "Hi"
15    print("Coding complete? Click 'Check' to review your tests and earn cool rewards!")
16

```

Run Output & Console

```

['Hello', 'World', 'hello']
this is a word, not a number!
this is a word, not a number!
this is a word, not a number!
AssertionError: Hello
<module>, 10

```

Enter a text:

Hi you are over 9000

Check Random

Let's add a counter variable before we start our for loop, then for every word we validate increment it by 1 -- if we can't validate that one of our list elements is a word, then we should reset our counter to 0.

```

counter = 0
...
counter += 1
...
else:
    counter = 0

```

```

1 def checkio(words):
2     split_words = words.split(' ')
3     print(split_words)
4     counter = 0
5     for word in split_words:
6         if word.isalpha():
7             print('this is a word, not a number!')
8             counter += 1
9         else:
10            counter = 0

```

Finally we need to check if our counter got to three -- if it did, we should return True indicating that there are three words in a row in the originally provided string. Otherwise, we should return False. We want to make sure that we return True as soon as the condition is met, so we'll leave that part in our "for" loop, and only return False at the end of our loop if we didn't already return True!

```

if counter >= 3:
    return True
return False

```

The screenshot shows the CheckIO IDE interface with the 'Three Words' mission selected. The code editor contains the following Python code:

```

1 def checkio(words):
2     split_words = words.split(' ')
3     print(split_words)
4     counter = 0
5     for word in split_words:
6         if word.isalpha():
7             print('this is a word, not a number!')
8             counter += 1
9         else:
10            counter = 0
11        if counter >= 3:
12            return True
13    return False
14

```

Run your code -- if all is well you should see the "Coding complete?" message at the bottom of the screen, you can now hit the "Check" button on the top menu bar to run your code through even more testing to validate it works as desired.

The screenshot shows the CheckIO IDE interface with the 'Three Words' mission selected. The code editor contains the following Python code with test assertions:

```

1 def checkio(words):
2     split_words = words.split(' ')
3     print(split_words)
4     counter = 0
5     for word in split_words:
6         if word.isalpha():
7             print('this is a word, not a number!')
8             counter += 1
9         else:
10            counter = 0
11        if counter >= 3:
12            return True
13    return False
14
15 #These "asserts" using only for self-checking and not necessary for auto-testing
16 if __name__ == '__main__':
17     assert checkio("Hello World hello") == True, "Hello"
18     assert checkio("He is 123 man") == False, "123 man"
19

```

The 'Run Output & Console' panel shows the following test results:

```

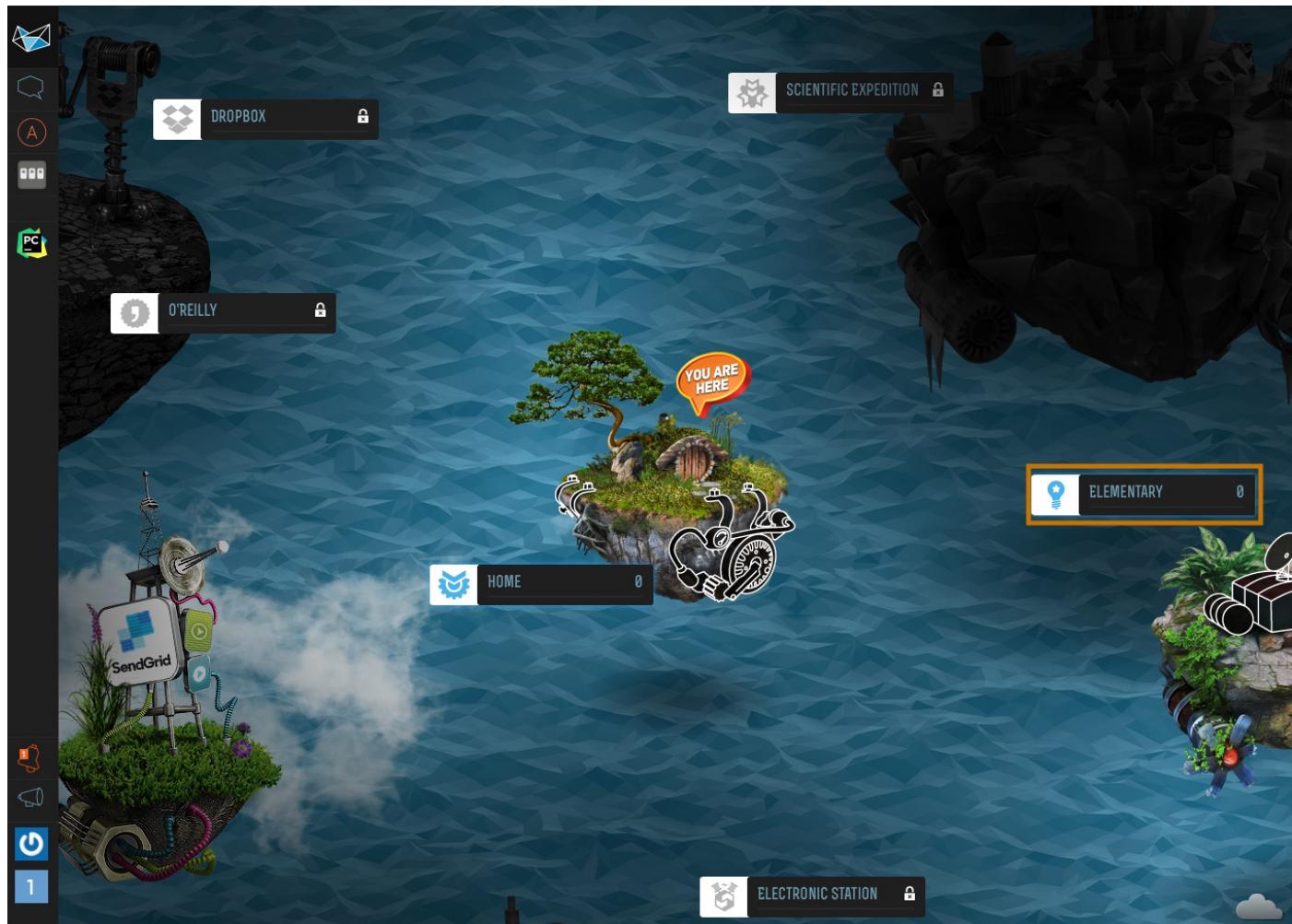
['He', 'is', '123', 'man']
this is a word, not a number!
this is a word, not a number!
this is a word, not a number!
[1', '2', '3', '4']
['bla', 'bla', 'bla', 'bla']
this is a word, not a number!
this is a word, not a number!
this is a word, not a number!
['Hi']
this is a word, not a number!
Coding complete? Click 'Check' to review your tests and earn cool rewards!

```

The 'Enter a text:' input field contains "Hi you are over 9000". The 'Check' and 'Random' buttons are visible at the bottom right of the console area.

## Fizz Buzz

To access the "Fizz Buzz" task, click on the "ELEMENTARY" station from the home screen:



Find and select the "Fizz Buzz" lab from the list

Click on "Solve it" to get started...

For this one, we'll just talk through the logic needed to complete the task, then you're on your own!

In this lab we want to evaluate if a number is divisible by 3, 5, or both. Depending on the outcome, we then return a string "Fizz Buzz" for divisible by both 3 and 5, "Fizz" if the number is divisible by 3, and "Buzz" if divisible by 5. Lastly, if the provided number is not divisible by 3 or 5, simply return the number that was provided.

Recall how we've used the "if" statement to evaluate conditions -- for this task you can use if, elif, elif, else conditionals to evaluate the provided number against each "question" (divisible by 3 and 5?, divisible by 3?, divisible by 5?, other?).

## Additional Labs

Try to work through the following labs next!

## Secret Message

For this task the ".isupper()" method will come in handy. If you're getting the hang of things you may also want to look into list comprehension!

## Right to Left

You probably want to investigate the "replace" method for this task. Replace does exactly what you think it should -- replaces a given string with a different string.

## Index Power

Uhoh, this one will make you do a bit of math! Thats OK though, Python can help. Check out the " $**$ " operator to handle exponents. Conditionals can be your friend for this task too -- is "n" bigger than the *length* of the array?

## Bonus Labs

If you're feeling up to it, here are two "bonus" labs that are a little bit tougher to solve!

### House Password

This one can be solved with a handful of "if"s chained together, but can you find a more efficient way? Regular Expressions may come in handy (but aren't required). If you solve it, take a look at some of the "Best Solutions" to see other ways you could have gone about doing it!

### Most Wanted Letter

As per usual... lots of ways to go about solving this one! "isalpha" is handy here, and depending on how you go about tackling this one you may want to look into the data structure called a "set" -- a set is kind of like a list, but it contains *only* unique elements.

# Python for Network and Systems Engineers Labs

## Overview and Objectives

In this lab you will explore Python modules that can be used to interact with network devices -- in both a "legacy" fashion, as well as more modern API driven ways.

Objectives:

- Learn about pexpect and "screen scraping"
- Play with Netmiko to abstract away some of the pain of screen scraping
- Configure and validate devices with Netmiko

## Introducing Pexpect

In this lab we'll learn the basics of Expect using the Python Pexpect module. Pexpect is a utility that is modeled after (but does not require) Expect. From the Pexpect documentation: "Pexpect is a pure Python module for spawning child applications; controlling them; and responding to expected patterns in their output. Pexpect works like Don Libes' Expect. Pexpect allows your script to spawn a child application and control it as if a human were typing commands."

What does that mean to us? Simply put, Pexpect will allow us to spawn SSH sessions, and interact with that process. We'll learn how to use Pexpect by logging into a virtual router and gathering data. We'll also learn about perhaps the most treacherous part of "screen scraping" -- parsing loads of unstructured data!

### Logging into a Router with pexpect

Before we can get any data from, or make any configurations to a device we first must of course figure out how to log in! As Pexpect is going to "spawn" a normal SSH session for us, the fundamentals of how to do this should be very familiar to you as it is pretty much exactly how you would manually do this!

Create a new text document called "pexpect\_task1.py" in the home directory we used previously, we'll be working out of this file for the task.

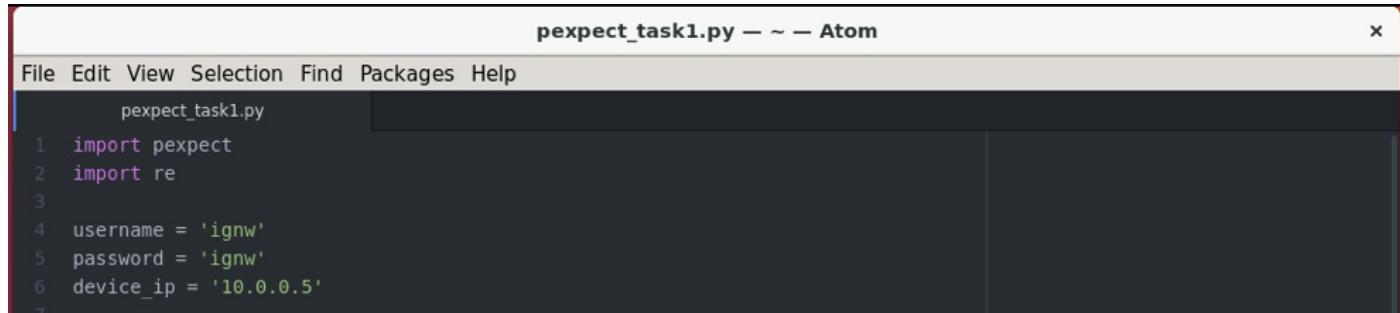
The very first thing we will need to do is import the Pexpect module -- this is because this is a "third-party" module (not included in the standard library). As previously outlined, it is common convention -- and a good practice -- to perform all import tasks at the beginning of our script, so that's where we will put our import.

```
import pexpect
```

Now that we've got that out of the way, think about the data you would normally need in order to log into a device -- you'll of course need the device IP address or hostname if it is DNS resolvable, and you'll need to know the username and password too. Let's go ahead and capture these inputs and store them in variables so that we can use them throughout our script as needed. By storing these inputs as variables we can ensure that if a device input (user/pass/IP) changes we only have to modify it in a single location as opposed to every instance of the field in our script.

*Note:* We'll be interacting with the *dev* environment for these tasks!

```
username = 'ignw'  
password = 'ignw'  
device_ip = '10.0.0.5'
```



```
pexpect_task1.py -- ~ -- Atom  
File Edit View Selection Find Packages Help  
pexpect_task1.py  
1 import pexpect  
2 import re  
3  
4 username = 'ignw'  
5 password = 'ignw'  
6 device_ip = '10.0.0.5'  
7
```

With that out of the way we now need to "spawn" an SSH session. You've probably noticed the intentional use of the word "spawn" several times so far -- that's because that is the name of the method (a function that is part of a class) that we will be using to create our SSH session. Just like if you were to manually SSH to a device, we'll need to pass a command to the spawn method that contains the IP address of our target device as well as our username.

One more important note is that we will need to assign the value of our spawned session to a variable. So far we've talked about strings, integers, lists, and dictionaries -- in this case our variable will be an "object" of type pexpect.spawn with the attributes that we apply to it (the command we send to it), let's create our pexpect.spawn object:

```
connection = pexpect.spawn(f'ssh {username}@{device_ip}'')
```

Go ahead and add a line to print out the value of connection, as well as the *type* of connection, then run your script to make sure it works and to see what the output looks like.

What does your script output? It should look similar to below:

The screenshot shows two windows. The top window is an Atom code editor with the file 'pexpect\_task1.py' open. The code imports pexpect and re, defines variables for username, password, and device\_ip, and then creates a connection object using pexpect.spawn with the command 'ssh {username}@{device\_ip}'. The bottom window is a terminal window titled 'ignw@ignw-jumphost: ~'. It shows the command 'python3 pexpect\_task1.py' being run, followed by a large amount of detailed output from the pexpect.spawn object, including its command, args, buffer, before, after, match, match\_index, exitstatus, flag\_eof, pid, child\_fd, closed, timeout, delimiter, logfile, logfile\_read, logfile\_send, maxread, ignorecase, searchwindowsize, delaybefore resend, delayafter close, and delavafter terminate. The last line of output is '<class \'pexpect.pty\_spawn.spawn\'>'. The entire terminal output is highlighted with an orange rectangle.

```
pexpect_task1.py -- ~ -- Atom
File Edit View Selection Find Packages Help
pexpect_task1.py
1 import pexpect
2 import re
3
4 username = 'ignw'
5 password = 'ignw'
6 device_ip = '10.0.0.5'
7
8 connection = pexpect.spawn(f'ssh {username}@{device_ip}')
9 print(connection)
10 print(type(connection))

ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 pexpect_task1.py
<pexpect.pty_spawn.spawn object at 0x7f1160563470>
command: /usr/bin/ssh
args: [ '/usr/bin/ssh', 'ignw@10.0.0.5' ]
buffer (last 100 chars): b''
before (last 100 chars): None
after: None
match: None
match_index: None
exitstatus: None
flag_eof: False
pid: 2779
child_fd: 5
closed: False
timeout: 30
delimiter: <class 'pexpect.exceptions.EOF'>
logfile: None
logfile_read: None
logfile_send: None
maxread: 2000
ignorecase: False
searchwindowsize: None
delaybefore resend: 0.05
delayafter close: 0.1
delavafter terminate: 0.1
<class 'pexpect.pty_spawn.spawn'>
ignw@ignw-jumphost:~$
```

There is some interesting data in here! We can see near the top what the "command" we ran is -- this should be the path to the ssh binary, we can also see that there is a list of "args" -- this contains the command we want to execute, as well as the input we've provided -- in this case "ignw@10.0.0.5". There is also a bunch of other data here that may come in handy later, but we can ignore for the time being. Finally, the last line of output is printing out the "type" of our connection object -- we can see that it is a type of "class 'pexpect.pty\_spawn.spawn'" just as we *expected* it to be!

Now that we've initiated our connection, what is the next step? What would be the next step if you were doing this manually? Almost certainly you'll be prompted for a password. Pexpect works exactly how it sounds like it should work -- it "expects" things, and then can react. We now need to tell our connection object to "expect" to see a password prompt. You can SSH to the router yourself so you know exactly what the prompt should look like. Once you know what to expect, we can tell our connection object with the "expect" method:

```
connection.expect('WHAT TO EXPECT')
```

The next step is to actually send the password, we can do that with the "sendline" method passing in our password variable as the argument:

```
connection.sendline(password)
```

Run your script again -- feel free to comment out the print statements from before if you get tired of seeing that output -- does it work? How can you tell? Go back and change your password to something that you know is incorrect and re-run your code -- what happened?

The screenshot shows a development environment with two windows. The top window is an Atom editor titled 'pexpect\_task1.py'. It contains a Python script with code for connecting via SSH and handling a password prompt. The lines 'connection.expect('Password:')

```
connection.sendline(password)
```

The bottom window is a terminal window titled 'ignw@ignw-jumphost: ~'. It shows the command 'ssh -l ignw 10.0.0.5' being run, followed by a password prompt 'Password:'. The terminal then shows a connection attempt to '10.0.0.5' failing with the message 'Connection to 10.0.0.5 closed by remote host.' The user then runs 'python3 pexpect\_task1.py', which successfully connects to the device.

```
ignw@ignw-jumphost:~$ ssh -l ignw 10.0.0.5
Password:

ignw-csr#
ignw-csr#
ignw-csr#exit
Connection to 10.0.0.5 closed by remote host.
Connection to 10.0.0.5 closed.
ignw@ignw-jumphost:~$ python3 pexpect_task1.py
ignw@ignw-jumphost:~$
```

The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~". It displays the command "ssh -l ignw 10.0.0.5" followed by a password prompt "Password:". Below the prompt, the terminal shows the user's session on a Cisco router, starting with "ignw-csr#". The user then enters "exit" and "Connection to 10.0.0.5 closed by remote host.". The terminal then shows two failed attempts to run the script "pexpect\_task1.py" with "python3".

```
pexpect_task1.py -- -- Atom
File Edit View Selection Find Packages Help
1 import pexpect
2 import re
3
4 username = 'ignw'
5 password = '1111ignw'
6 device_ip = '10.0.0.5'
7
8 connection = pexpect.spawn(f'ssh {username}@{device_ip}')
9 # print(connection)
10 # print(type(connection))
11
12
13 connection.expect('Password:')
14 connection.sendline(password)
15

ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ ssh -l ignw 10.0.0.5
Password:

ignw-csr#
ignw-csr#
ignw-csr#exit
Connection to 10.0.0.5 closed by remote host.
Connection to 10.0.0.5 closed.
ignw@ignw-jumphost:~$ python3 pexpect_task1.py
ignw@ignw-jumphost:~$ python3 pexpect_task1.py
ignw@ignw-jumphost:~$
```

Clearly we need to find out what's going on! Pexpect has "before" and "after" methods -- these methods print out the captured data from before or after the previous "expected" string. Let's go ahead and try to print them both out for kicks:

```
print(connection.before)
print(connection.after)
```

```
pexpect_task1.py -- ~ Atom
File Edit View Selection Find Packages Help
1 import pexpect
2 import re
3
4 username = 'ignw'
5 password = '1111ignw'
6 device_ip = '10.0.0.5'
7
8 connection = pexpect.spawn(f'ssh {username}@{device_ip}')
9 # print(connection)
10 # print(type(connection))
11
12 connection.expect('Password:')
13 connection.sendline(password)
14
15 print(connection.before)
16 print(connection.after)

ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 pexpect_task1.py
b''
b'Password: '
ignw@ignw-jumphost:~$
```

Interesting -- the "before" line prints an empty string. This is because there was nothing "before" the previous expect... that makes sense of course because this is the very beginning of our connection to this router.

The "after" method however does print out something at least, unfortunately it's not printing out anything terribly useful.

We need to try to "expect" something else and then try this again. What happens after a successful login? Try to configure your script to expect the prompt that the router provides once logged in. Then once more try to print out the "before" and "after" methods to see what data you receive. Don't forget to set the password back to "ignw"!

```
connection.expect('EXPECT WHAT?')
```

At this point you should be seeing some output that you can work with!



pexpect\_task1.py — — Atom

```
File Edit View Selection Find Packages Help
pexpect_task1.py
1 import pexpect
2 import re
3
4 username = 'ignw'
5 password = 'ignw'
6 device_ip = '10.0.0.5'
7
8 connection = pexpect.spawn(f'ssh {username}@{device_ip}')
9 # print(connection)
10 # print(type(connection))
11
12 connection.expect('Password:')
13 connection.sendline(password)
14
15 print(connection.before)
16 print(connection.after)
17
18 connection.expect('ignw-csr#')
19 print(connection.before)
20 print(connection.after)
21
```

ignw@ignw-jumphost: ~

```
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 pexpect_task1.py
b''
b'Password:'
b' \r\n\r\n\r\n\r\n\r\n'
b'ignw-csr#'
ignw@ignw-jumphost:~$
```

## Get the Hostname

Now that we're logged in and getting at least some data back from our device, let's try to get the hostname from the router, store it in a variable, and then print it out to standard out with a nice message.

For this task you're on your own -- you're most of the way there, if you've followed along so far, but if you get stuck feel free to ask your instructor for help!



ignw@ignw-jumphost: ~

```
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 pexpect_task1.py
b''
b'Password:'
b' \r\n\r\n\r\n\r\n\r\n'
b'ignw-csr#'
b'ignw-csr'
ignw@ignw-jumphost:~$
```

## Capture Relevant Interface Configuration

Let's try something a bit more complex now -- capture the running configuration of an interface from your router (one with an IP address on it) and assign it to a variable. By now you probably have a good idea how to accomplish this, it may look something like:

```
connection.sendline('show run interface g1')
connection.expect('ignw-csr#')
```

Why do we have to do the "expect" line? The short answer is that under the covers Pexpect is basically grep-ing to capture output for us -- we are grep-ing for all data "before" the expected output, if we use the "after" argument, Pexpect greps for all data including and after the expected string. Now that we understand roughly how Pexpect is grabbing output, let's assign the "before" output ("before" the expected hostname prompt) to a variable, and print it out for good measure:

```
interface_output = connection.before
print(interface_output)
```

Your output should look similar to what is shown below:

The screenshot shows two windows. The top window is an Atom code editor with the file 'pexpect\_task1.py' open. The bottom window is a terminal window titled 'ignw@ignw-jumphost: ~'. The terminal output shows the execution of the Python script, which prints the raw output of the 'show run interface g1' command. A yellow box highlights the command and its output in the terminal.

```

pexpect_task1.py -- ~ -- Atom
File Edit View Selection Find Packages Help
pexpect_task1.py
10 # print(type(connection))
11
12 connection.expect('Password:')
13 connection.sendline(password)
14
15 print(connection.before)
16 print(connection.after)
17
18 connection.expect('ignw-csr#')
19 print(connection.before)
20 print(connection.after)
21
22 hostname = connection.after[:-1]
23 print(hostname)
24
25 connection.sendline('show run interface g1')
26 connection.expect('ignw-csr#')
27 interface_output = connection.before
28 print(interface_output)
29
30
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 pexpect_task1.py
b''
b'Password:'
b' \r\n\r\n\r\n\r\n\r\n'
b'ignw-csr#'
b'ignw-csr'
b'show run interface g1\r\nBuilding configuration...\r\n\r\nCurrent configuration : 144 bytes\r\n!\r\ninterface
GigabitEthernet1\r\n  vrf forwarding management\r\n  ip address 10.0.0.5 255.255.255.0\r\n  negotiation auto\r\n  no
  mop enabled\r\n  no mop sysid\r\n  end\r\n\r\n'
ignw@ignw-jumphost: ~

```

This is... "OK"... clearly we have all of the output from the router, but obviously there are some difficulties with formatting. This is one of the chief difficulties with screen-scraping: data is returned to us in an unstructured format.

Let's try to parse the data and store the interface name, IP address, and description as fields in a list. Look at the data that we received, how is it different from if we were to manually run our show commands in an SSH session?

You'll probably notice that what we received from Pexpect is one line with a bunch of "\r" and "\n" characters inserted in-line. These are escape sequences that represent "Carriage Return" and "Line Feed" respectively -- basically, these are the text representation of the formatting that you're used to seeing in a terminal window. We can use these formatting markers to our benefit by "splitting" output into new lines at each of these tags. Let's try that:

```

split_output = interface_output.split('\r\n')
print(split_output)

```

Run your script to see if that works...

```
pexpect_task1.py -- ~ Atom
File Edit View Selection Find Packages Help
pexpect_task1.py
10 # print(type(connection))
11
12 connection.expect('Password:')
13 connection.sendline(password)
14
15 print(connection.before)
16 print(connection.after)
17
18 connection.expect('ignw-csr#')
19 print(connection.before)
20 print(connection.after)
21
22 hostname = connection.after[:-1]
23 print(hostname)
24
25 connection.sendline('show run interface g1')
26 connection.expect('ignw-csr#')
27 interface_output = connection.before
28 print(interface_output)
29
30 split_output = interface_output.split('\r\n')
31 print(split_output)
32
33 ...
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 pexpect_task1.py
b''
b'Password:'
b' \r\n\r\n\r\n\r\n\r\n'
b'ignw-csr#'
b'ignw-csr'
b'show run interface g1\r\nBuilding configuration...\r\n\r\nCurrent configuration : 144 bytes\r\n!\r\ninterface
GigabitEthernet1\r\n  vrf forwarding management\r\n  ip address 10.0.0.5 255.255.255.0\r\n  negotiation auto\r\n  no
mop enabled\r\n  no mop sysid\r\n  send\r\n  no cdp
Traceback (most recent call last):
  File "pexpect_task1.py", line 30, in <module>
    split_output = interface_output.split('\r\n')
TypeError: a bytes-like object is required, not 'str'
ignw@ignw-jumphost: ~
```

Uh-oh, looks like Python is complaining about another kind of "TypeError" that we haven't ran into yet -- "a bytes-like object is required, not "str". Let's try to print out what "type" our interface\_output variable is so we can understand a bit better what's going on:

```
print(type(interface_output))
```

```
pexpect_task1.py -- -- Atom
File Edit View Selection Find Packages Help
pexpect_task1.py
10 # print(type(connection))
11
12 connection.expect('Password:')
13 connection.sendline(password)
14
15 print(connection.before)
16 print(connection.after)
17
18 connection.expect('ignw-csr#')
19 print(connection.before)
20 print(connection.after)
21
22 hostname = connection.after[:-1]
23 print(hostname)
24
25 connection.sendline('show run interface g1')
26 connection.expect('ignw-csr#')
27 interface_output = connection.before
28 print(interface_output)
29
30 print(type(interface_output))
31 split_output = interface_output.split('\r\n')
32 print(split_output)
33
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 pexpect_task1.py
b''
b'Password:'
b' \r\n\r\n\r\n\r\n\r\n'
b'ignw-csr#'
b'ignw-csr'
b'show run interface g1\r\nBuilding configuration...\r\n\r\nCurrent configuration : 144 bytes\r\n!\r\ninterface
GigabitEthernet1\r\n vrf forwarding management\r\n ip address 10.0.0.5 255.255.255.0\r\n negotiation auto\r\n no
mop enabled()\r\n no mop sysid\r\nend\r\n\r\n\r\n'
<class 'bytes'>
Traceback (most recent call last):
  File "pexpect_task1.py", line 31, in <module>
    split_output = interface_output.split('\r\n')
TypeError: a bytes-like object is required, not 'str'
ignw@ignw-jumphost:~$
```

Hmm, the `interface_output` variable is actually a "bytes" type object already, so this error is a little bit confusing! What we need to do is actually *decode* our bytes object to a "normal" string so that the `split` method can do its job. We do that very simply with the "decode" method:

```
split_output = interface_output.decode().split('\r\n')
```

Run your script again.

```

pexpect_task1.py -- ~ -- Atom
File Edit View Selection Find Packages Help
pexpect_task1.py
10 # print(type(connection))
11
12 connection.expect('Password:')
13 connection.sendline(password)
14
15 print(connection.before)
16 print(connection.after)
17
18 connection.expect('ignw-csr#')
19 print(connection.before)
20 print(connection.after)
21
22 hostname = connection.after[:-1]
23 print(hostname)
24
25 connection.sendline('show run interface g1')
26 connection.expect('ignw-csr#')
27 interface_output = connection.before
28 print(interface_output)
29
30 print(type(interface_output))
31 split_output = interface_output.decode().split('\r\n')
32 print(split_output)
33

ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 pexpect_task1.py
b''
b'Password:'
b' \r\n\r\n\r\n\r\n\r\n'
b'ignw-csr#'
b'ignw-csr'
b'show run interface g1\r\nBuilding configuration...\r\n\r\nCurrent configuration : 144 bytes\r\n!\r\ninterface
GigabitEthernet1\r\n vrf forwarding management\r\n ip address 10.0.0.5 255.255.255.0\r\n negotiation auto\r\n no
 mop enabled\r\n no mop sysid\r\nend\r\n\r\n'
<class 'bytes'>
['show run interface g1', 'Building configuration...', '', 'Current configuration : 144 bytes', '!', 'interface GigabitEthernet1', ' vrf forwarding management', ' ip address 10.0.0.5 255.255.255.0', ' negotiation auto', ' no mop enabled', ' no mop sysid', 'end', '', '']
ignw@ignw-jumphost:~$
```

That is much easier to look at isn't it! We now have a list with each section of the configuration (including undesirable parts like the "!" and blank entries).

Using a for loop, parse each item of the list and store the interface name, IP address, and description into variables, finally print out your results to standard out. For now, try to accomplish this by using conditionals (*if* statements) and slicing. You may also want to Internet search for the ".startswith()" method -- it may come in handy for this task! If you get stuck ask your instructor for a tip!

The screenshot shows a dual-pane interface. The left pane displays the Python script `pexpect_task1.py` with syntax highlighting. The right pane shows a terminal window titled `ignw@ignw-jumphost: ~` running on a Linux system. The terminal output shows the execution of the script, which prints the configuration details of the 'GigabitEthernet1' interface.

```
split_output = interface_output.decode().split('\r\n')
# print(split_output)
interface_description = 'N/A'

for line in split_output:
    if line.startswith('interface'):
        interface_name = line[10:]
    elif line.startswith(' ip address'):
        interface_ip_address = line[12:]
    elif line.startswith(' description'):
        interface_description = line[12:]

print(f'Interface: {interface_name}, Description: {interface_description},'
      f'IP: {interface_ip_address}')



```

**Bonus:** How can you handle scenarios where there is no IP address or description configured?

**Bonus Part 2:** Try to complete the same task using regular expressions to find the desired fields without using `split!`

The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~". The terminal menu bar includes File, Edit, View, Search, Terminal, and Help. The command "python3 pexpect\_task1.py" is run, followed by its output: "Interface: GigabitEthernet1, Description: Management - Do not Modify!, IP: 10.0.0.5 255.255.255.0". Below the terminal window is a code editor showing the script "pexpect\_task1.py" with line numbers 47 to 54. The code uses regular expressions to parse interface information from a string. At the bottom of the code editor, it says "pexpect\_task1.py 47:1". To the right of the code editor are file status indicators: LF, UTF-8, Python, and 0 files.

```
47
48 interface_name = re.findall(rb'interface.[A-Z, a-z]*?Ethernet[\r]*', interface_output)
49 interface_description = re.findall(rb'description[\r]*', interface_output)
50 if not interface_description:
51     interface_description.append(b'          N/A')
52 interface_ip_address = re.findall(rb'ip address.[0-255]{0,3}.[0-255]{0,3}.[0-255]{0,3}.[0-255]{0,3}.[0-255]{0,3}.[0-255]{0,3}', interface_output)
53 print(f'Interface: {interface_name[0].decode()[10:]}, Description: {interface_description[0].decode()[12:]}, IP: {interface_ip_address[0].decode()}'
```

## Introducing Netmiko

Now that you've had some experience with "screen-scraping" you can probably understand why it's not the greatest way to interact with devices. The reality of most networks, however, is that a high percentage of devices do not have an API or any way to return structured data (this is changing, thankfully!), so screen-scraping is still very much a necessary evil. Pexpect is a fantastic library, but it was built as a general purpose utility -- not specifically for networking. Netmiko is a screen-scraping module that is built on top of Paramiko (another Pexpect like library) that is 100% purpose built for interacting with Networking devices. In this task we'll perform the exact same tasks as we did with Pexpect, but this time we'll use Netmiko.

### Logging into a Router with Netmiko

Create a new text document called "netmiko\_task1.py" in the home directory we used previously, we'll be working out of this file for the task.

As with Pexpect, Netmiko is a third party library. It's already been installed for you, but you will need to import it into your script. Unlike Pexpect, however, we will want to import only a particular class from the library. Usually you can find out the basic usage of a library from the libraries Github or documentation page, for Netmiko there are some examples on the Github page that we can use as a reference for basic usage, click [here](#) to link to the Netmiko Github page.

The basic examples many libraries provide will usually let you know how to import the module -- i.e. import the

whole module as we did with Pexpect, or import only a particular class as in this instance.

```
from netmiko import ConnectHandler
```

If you haven't already, take a look at the Netmiko Github page and the example for how to connect to a Cisco router -- we'll copy that format to build our test connection. Much like Pexpect, we will be creating a connection "object" and passing the appropriate information to it.

We'll define the required arguments in a dictionary which we will then pass to the Netmiko ConnectHandler class with the "\*\*\*" annotation -- this "double splat" -- annotation unpacks dictionary key/value pairs and sends them as keyword arguments to the class.

```
cisco_cloud_router = {'device_type': 'cisco_ios',
                      'ip': '10.0.0.5',
                      'username': 'ignw',
                      'password': 'ignw'}
connection = ConnectHandler(**cisco_cloud_router)
```

Let's also print out the "type", and the value of our connection variable so we can check that out. Run your script and see what happens.

The screenshot shows two windows. The top window is an Atom code editor with the file 'netmiko\_task1.py' open. The code defines a dictionary 'cisco\_cloud\_router' with keys 'device\_type', 'ip', 'username', and 'password' set to 'cisco\_ios', '10.0.0.5', 'ignw', and 'ignw' respectively. It then creates a 'connection' object by calling 'ConnectHandler' with '\*\*cisco\_cloud\_router' as an argument. Finally, it prints the 'connection' variable and its type. The bottom window is a terminal window titled 'ignw@ignw-jumphost: ~'. It shows the command 'python3 netmiko\_task1.py' being run, followed by the output: '<netmiko.cisco.cisco\_ios.CiscoIosSSH object at 0x7f4d486b1710>' and '<class 'netmiko.cisco.cisco\_ios.CiscoIosSSH'>'. Both the code in the editor and the terminal output are highlighted with orange boxes.

```
netmiko_task1.py — — Atom
File Edit View Selection Find Packages Help
netmiko_task1.py
1 from netmiko import ConnectHandler
2
3 cisco_cloud_router = {'device_type': 'cisco_ios',
4                      'ip': '10.0.0.5',
5                      'username': 'ignw',
6                      'password': 'ignw'}
7 connection = ConnectHandler(**cisco_cloud_router)
8
9 print(connection)
10 print(type(connection))
11
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost: ~$ python3 netmiko_task1.py
<netmiko.cisco.cisco_ios.CiscoIosSSH object at 0x7f4d486b1710>
<class 'netmiko.cisco.cisco_ios.CiscoIosSSH'>
ignw@ignw-jumphost: ~$
```

Unsurprisingly the "type" of our connection variable is a class of type netmiko -- the type also tells us that we have an attribute of "cisco\_ios", which makes sense since that is what we created our object as! Printing just the value of the connection variable shows us again that it is a netmiko object and also where in memory that object is stored.

In theory at this point we've logged into the router, let's verify that by running a command and printing the output. Using the Netmiko documentation as a guide, try to perform a "show run interface g1" and print the output.



A screenshot of a terminal window titled "ignw@ignw-jumphost: ~". The window shows the following command-line session:

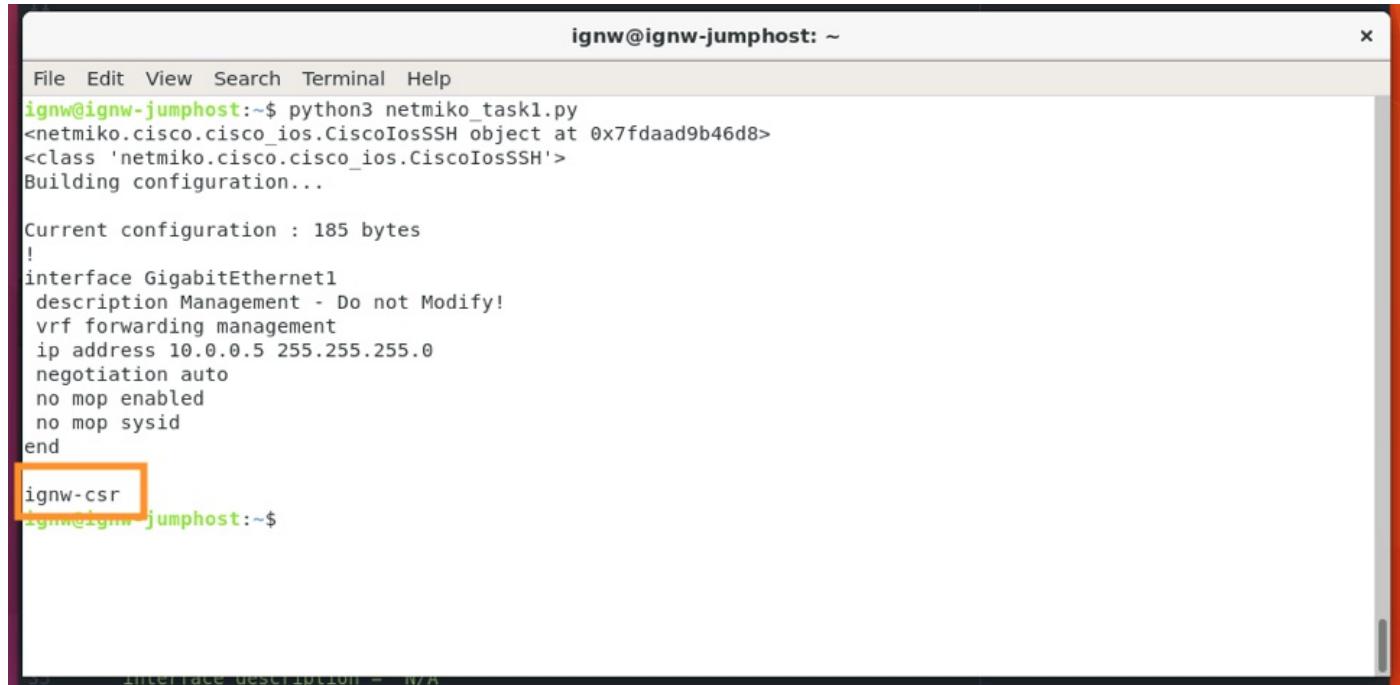
```
ignw@ignw-jumphost:~$ python3 netmiko_task1.py
<netmiko.cisco.cisco_ios.CiscoIosSSH object at 0x7f4d486b1710>
<class 'netmiko.cisco.cisco_ios.CiscoIosSSH'>
ignw@ignw-jumphost:~$ python3 netmiko_task1.py
<netmiko.cisco.cisco_ios.CiscoIosSSH object at 0x7fa22e97a710>
<class 'netmiko.cisco.cisco_ios.CiscoIosSSH'>
Building configuration...
Current configuration : 185 bytes
!
interface GigabitEthernet1
description Management - Do not Modify!
vrf forwarding management
ip address 10.0.0.5 255.255.255.0
negotiation auto
no mop enabled
no mop sysid
end
ignw@ignw-jumphost:~$
```

Your output should look similar to that shown above. If you run into issues, check out the examples or ask your instructor for a tip. Now that we know our connection is up and running, we'll try to complete the same tasks we did with Pexpect.

## Get the Hostname

To capture the hostname of our router, we could very easily pass a "show run | include hostname", but there is an even easier way to do this with Netmiko. Try to search through the Netmiko documentation to find it, this will give you a bit of practice working with Python documentation (Netmiko is a pretty well documented library so it's a good place to start!).

**Tip:** look for the keyword "prompt" -- that should help you find the one-liner to capture the hostname!



A terminal window titled "ignw@ignw-jumphost: ~". The window shows the output of a Python script named "netmiko\_task1.py". The script connects to a Cisco router via SSH and prints its configuration. The configuration includes an interface configuration for GigabitEthernet1 and a host entry for "ignw-csr". The host entry is highlighted with a red box.

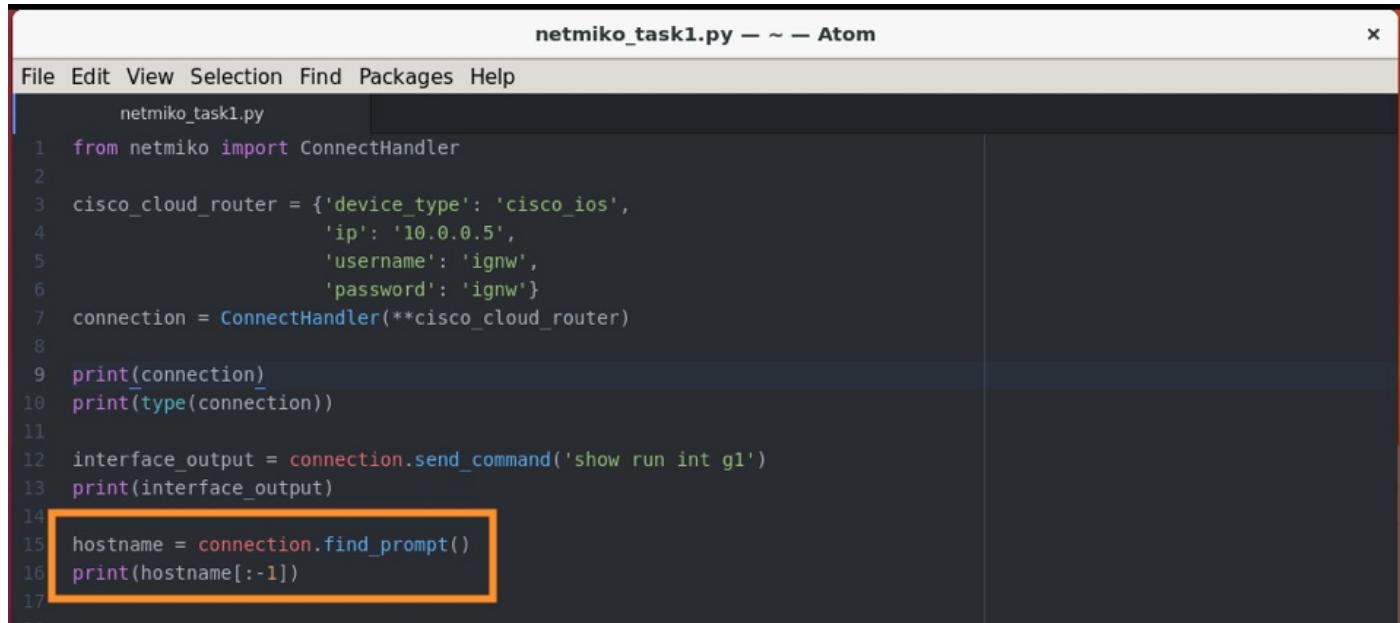
```
ignw@ignw-jumphost:~$ python3 netmiko_task1.py
<netmiko.cisco.cisco_ios.CiscoIosSSH object at 0x7fdaad9b46d8>
<class 'netmiko.cisco.cisco_ios.CiscoIosSSH'>
Building configuration...

Current configuration : 185 bytes
!
interface GigabitEthernet1
description Management - Do not Modify!
vrf forwarding management
ip address 10.0.0.5 255.255.255.0
negotiation auto
no mop enabled
no mop sysid
end

ignw-csr
ignw@ignw-jumphost:~$
```

The "find\_prompt" method does exactly what we need. If we didn't want to see the "#" symbol we could even slice that off while we print:

```
hostname = connection.find_prompt()
print(hostname[:-1])
```



An Atom code editor window titled "netmiko\_task1.py — ~ — Atom". The file contains a Python script for connecting to a Cisco router. Line 15 contains the code "hostname = connection.find\_prompt()", which is highlighted with a red box.

```
netmiko_task1.py — ~ — Atom
File Edit View Selection Find Packages Help
netmiko_task1.py
1 from netmiko import ConnectHandler
2
3 cisco_cloud_router = {'device_type': 'cisco_ios',
4                      'ip': '10.0.0.5',
5                      'username': 'ignw',
6                      'password': 'ignw'}
7 connection = ConnectHandler(**cisco_cloud_router)
8
9 print(connection)
10 print(type(connection))
11
12 interface_output = connection.send_command('show run int g1')
13 print(interface_output)
14
15 hostname = connection.find_prompt()
16 print(hostname[:-1])
17
```

## Capture Relevant Interface Configuration

Now, we'll capture the same data that we captured before: interface name, description (if applicable), and IP address (if applicable).

We already ran and captured the output of our show interface command, but there has to be an easier way than splitting lines and using conditionals or relying on regular expressions right?

Let's work smarter not harder this time and use the built in pipe ("|") filtering options on our router. To get the interface name we can use our pipe and a single regex character "^" (indicating the start of a line):

```
interface_name = connection.send_command('show run int g2 | i ^interface')
```

Print out your interface name to make sure we captured it correctly.

The screenshot shows a development environment with two windows. The top window is an Atom code editor titled "netmiko\_task1.py". It contains a Python script that imports netmiko, defines a device configuration for a Cisco IOS router, establishes a connection, prints the connection object, sends a show run command for interface g1, prints the output, finds the prompt, prints the hostname, and finally sends a show run command for interface g1 filtered by the '^interface' regex and prints the result. The line containing the regex filter is highlighted with an orange box. The bottom window is a terminal window titled "ignw@ignw-jumphost: ~". It shows the execution of the script, the connection object, the building configuration, the current configuration details, and the final output which includes the interface configuration. The interface configuration line is also highlighted with an orange box.

```
netmiko_task1.py — ~ — Atom
File Edit View Selection Find Packages Help
File Edit View Selection Find Packages Help
1 from netmiko import ConnectHandler
2
3 cisco_cloud_router = {'device_type': 'cisco_ios',
4                      'ip': '10.0.0.5',
5                      'username': 'ignw',
6                      'password': 'ignw'}
7 connection = ConnectHandler(**cisco_cloud_router)
8
9 print(connection)
10 print(type(connection))
11
12 interface_output = connection.send_command('show run int g1')
13 print(interface_output)
14
15 hostname = connection.find_prompt()
16 print(hostname[:-1])
17
18 interface_name = connection.send_command('show run int g1 | i ^interface')
19 print(interface_name)
20
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 netmiko_task1.py
<netmiko.cisco.cisco_ios.CiscoIosSSH object at 0x7f7540e956d8>
<class 'netmiko.cisco.cisco_ios.CiscoIosSSH'>
Building configuration...
Current configuration : 185 bytes
!
interface GigabitEthernet1
description Management - Do not Modify!
vrf forwarding management
ip address 10.0.0.5 255.255.255.0
negotiation auto
no mop enabled
no mop sysid
end

ignw@ignw-jumphost:~$
```

That was a lot easier than splits and regex, right? Using a similar technique, capture the description and IP address of the interface.

**Bonus:** How can you handle scenarios where there is no IP address or description configured?

**Bonus 2:** Can you add some logic to handle secondary IP addresses? Configure your router with a secondary IP address and play with ways to capture and print them out in a human readable fashion.

The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~". The window has a menu bar with File, Edit, View, Search, Terminal, and Help. The terminal output shows the command "python3 netmiko\_task1.py" being run, which prints the IP address [ '10.0.0.5 255.255.255.0' ] and the description "Management - Do not Modify!". Below the terminal window, a code editor displays the Python script "netmiko\_task1.py". The code uses Netmiko to send commands to a device and extract IP addresses and descriptions. Lines 21 through 35 are highlighted with a yellow box.

```
21 ip_output = connection.send_command('show run int g1 | i ip address')
22 if 'no' in ip_output:
23     interface_ip_address = []
24     interface_ip_address[0] = 'N/A'
25 else:
26     interface_ip_address = []
27     for line in ip_output.split('\n'):
28         interface_ip_address.append(line[12:])
29 print(interface_ip_address)
30
31 interface_description = connection.send_command('show run int g1 | i des')
32 if not interface_description:
33     interface_description = 'N/A'
34 print(interface_description)
```

## Netmiko Basic Configuration

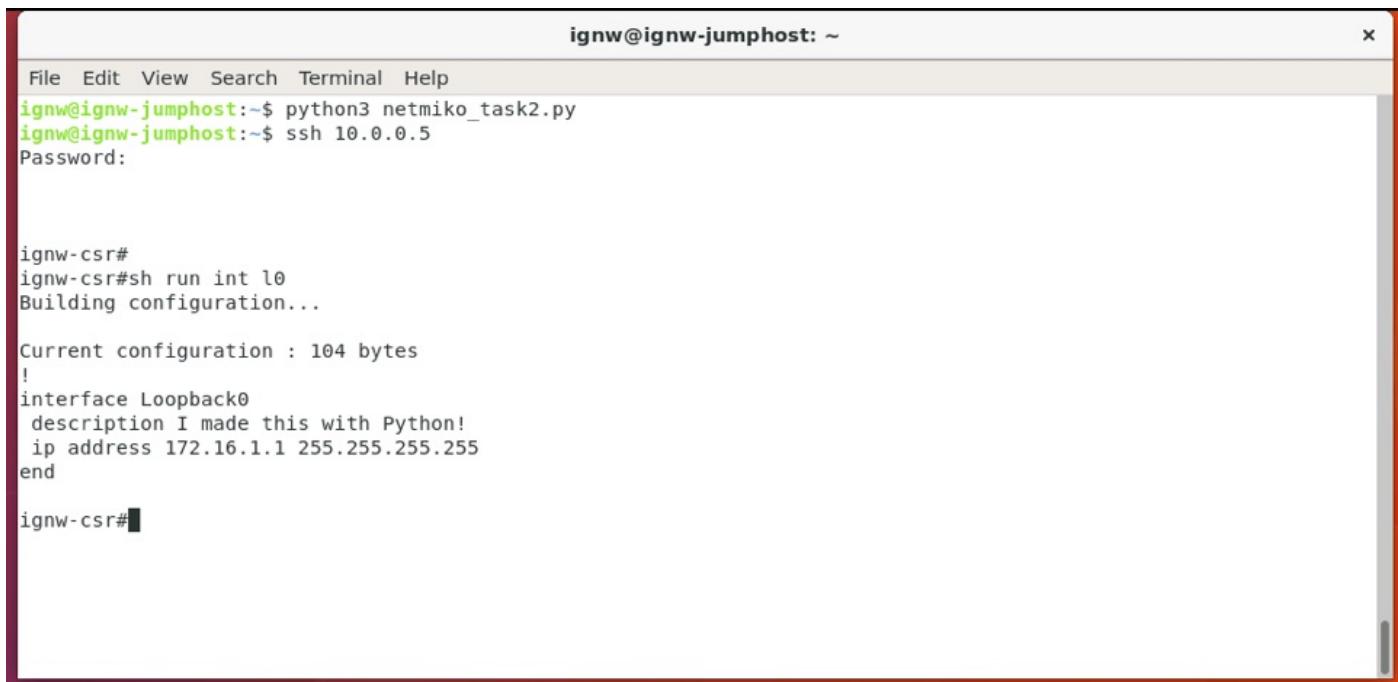
It's finally time to start making configuration changes with Python! In this task you will configure a new loopback interface, assign it an IP address and validate that it is in the "up/up" state.

### Configure a new loopback

Create a new text document called "netmiko\_task2.py" in the home directory we used previously, we'll be working out of this file for the task.

Use "loopback0" for this task, and an IP address of 172.16.1.1 255.255.255.255.

You're on your own for this task, you may want to search the Netmiko documentation for the "enable" and "send\_config\_set" methods as these will make your life easier!



The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~". The window has a menu bar with File, Edit, View, Search, Terminal, and Help. The terminal session starts with "ignw@ignw-jumphost:~\$ python3 netmiko\_task2.py" and "ignw@ignw-jumphost:~\$ ssh 10.0.0.5". It then prompts for a password. The configuration command "sh run int lo" is entered, followed by "Building configuration...". The output shows the configuration of a loopback interface "Loopback0" with the description "I made this with Python!", IP address "172.16.1.1", and subnet mask "255.255.255.255". The configuration is saved with "end". The final prompt is "ignw-csr#".

```
ignw@ignw-jumphost:~$ python3 netmiko_task2.py
ignw@ignw-jumphost:~$ ssh 10.0.0.5
Password:

ignw-csr#
ignw-csr#sh run int lo
Building configuration...

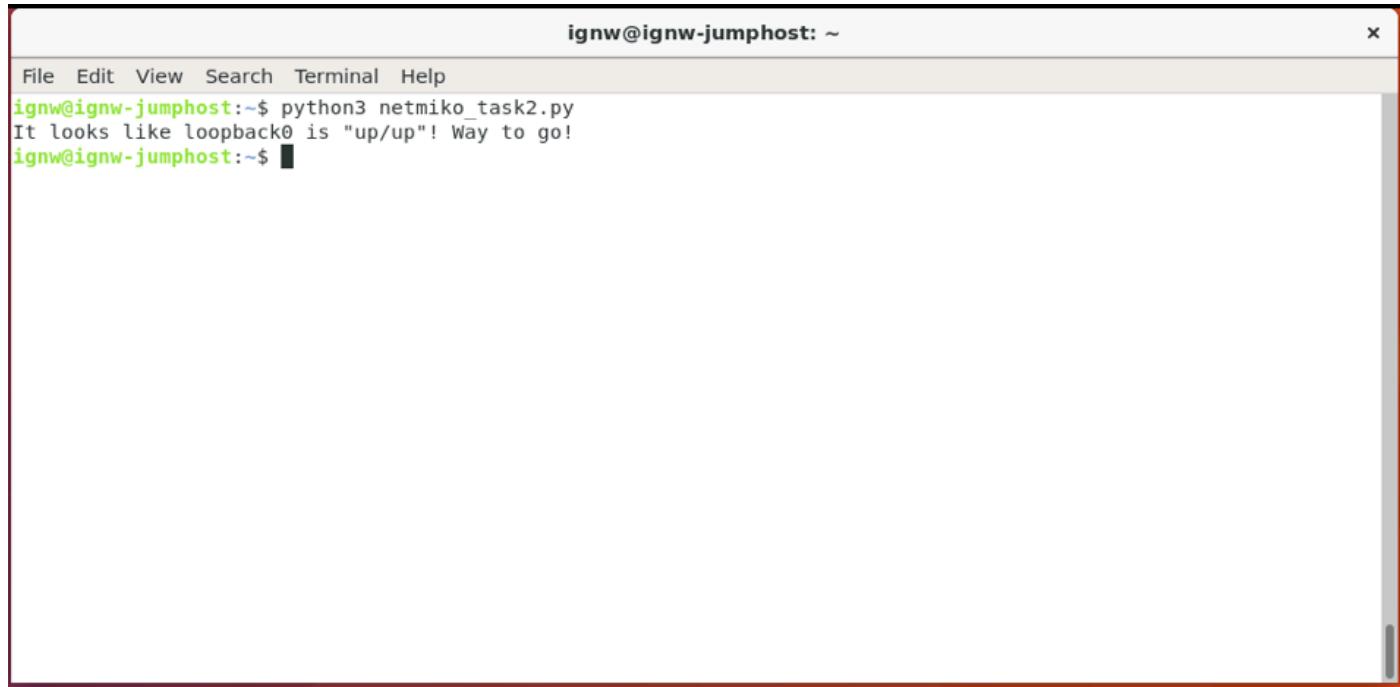
Current configuration : 104 bytes
!
interface Loopback0
    description I made this with Python!
    ip address 172.16.1.1 255.255.255.255
end

ignw-csr#
```

## Validate Interface is "up/up"

Now that you've successfully configured your loopback its time to validate that your configuration is successful and the loopback interface is in the "up/up" state.

How would you do this if you were manually running CLI commands? Can you take some of that same logic and apply it in Python? Can you leverage built in IOS tooling to help with this task as we've done before? There are many, many ways to go about this task, pick one and give it a go!

A screenshot of a terminal window titled "ignw@ignw-jumphost: ~". The window has a standard OS X-style title bar with icons for close, minimize, and zoom. The menu bar at the top includes "File", "Edit", "View", "Search", "Terminal", and "Help". Below the menu is a command-line interface. The user has run the command "python3 netmiko\_task2.py". The output of the script is displayed in green text:

```
ignw@ignw-jumphost:~$ python3 netmiko_task2.py
It looks like loopback0 is "up/up"! Way to go!
ignw@ignw-jumphost:~$
```

## Config and Validate (CSR, ASA, and NXOSv)

In this task we'll branch out and configure not only the CSR, but also an ASA and NXOSv device by configuring a new loopback interface, creating an ACL, some interfaces, and then validating your configurations with ICMP.

This task will require you to handle connecting to three devices with different credentials (enable password vs not) and different CLI syntax. This is obviously a very small example, but this task is representative of a real world work flow that you may have in your environment.

In our setup the CSR is representing an "edge" router -- with the soon to be created loopback acting as a mock Internet host. Our ASA is of course functioning as an "edge" firewall, and our NX-OSv device will serve as a DMZ or Core host. You'll need to provision the new loopback, interfaces (on the CSR, ASA, and NX-OSv), as well as some routing and an access-control list.

As we saw in the previous task validation is a critical component of automation and scripting -- in this task you will need to validate your configurations with a functional test to prove your script was successful, and that your internal device (NX-OSv) can get to the "Internet"!

You can re-use any of the code you've already created up to this point (Pexpect or Netmiko), or you can experiment with other libraries/tools if you're feeling up for a challenge (paramiko, Brigade (now nornir), ciscoconfigparse, etc -- there are many options and you have privileges to install whatever you need!)! Whatever you do -- be thinking about how the code you are creating can be made to be modular and re-usable!

Create a new text document called "netmiko\_task3.py" in the home directory we used previously. We'll use this file for all of the "Config and Validate" tasks.

### CSR Configurations

#### Create another Loopback

On the CSR, create a new loopback, "loopback1" with an IP address of 8.8.4.4 255.255.255.255. This task

should be very straightforward -- you can simply re-use and tweak the previous tasks code to accomplish this.

The image shows two terminal windows side-by-side. The top window has a red border and displays the command `python3 netmiko_task3.py` followed by the message "It looks like loopback1 is "up/up"! Way to go!". The bottom window also has a red border and shows the command `ssh 10.0.0.5` followed by a password prompt. It then displays the output of the `sh ip int brief` command on a Cisco CSR device. The table shows the following interfaces:

Interface	IP-Address	OK?	Method	Status	Protocol
GigabitEthernet1	10.0.0.5	YES	NVRAM	up	up
GigabitEthernet2	203.0.113.1	YES	manual	up	up
Loopback0	172.10.1.1	YES	manual	up	up
Loopback1	8.8.4.4	YES	manual	up	up

netmiko\_task3.py — — Atom

```
File Edit View Selection Find Packages Help
    netmiko_task3.py
1 import sys
2 import time
3 from netmiko import ConnectHandler
4
5
6 cisco_cloud_router = {'device_type': 'cisco_ios',
7                         'ip': '10.0.0.5',
8                         'username': 'ignw',
9                         'password': 'ignw'}
10
11 connection = ConnectHandler(**cisco_cloud_router)
12
13 commands = ['interface loopback1',
14             'description IGNW was here!',
15             'ip address 8.8.4.4 255.255.255.255',
16             'no shut']
17
18 connection.config_mode()
19
20 connection.send_config_set(commands)
21 time.sleep(2)
22
23 show_output = connection.send_command('show ip int loopback1 | i Loopback1')
24 if show_output.count('up') == 2:
25     print('It looks like loopback1 is "up/up"! Way to go!')
26 else:
27     print('Something went wrong... let\'s get outa here before we break something!')
28     sys.exit()
29
```

## Configure Interface

The next step is to configure the interface that connects to the "outside" interface of the ASA. Use an IP address of "203.0.113.1 255.255.255.192" on the GigabitEthernet2 interface of the CSR. This task is really more of the same as before.

The following screen shots will show a very simple way to attack this and the following tasks; though it may not be a very reusable or flexible, it will get you going! Try to make your code even better than the screen shots!

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 netmiko_task3.py
It looks like loopback1 is "up/up"! Way to go!
Moving on to GigabitEthernet...
It looks like GigabitEthernet2 is "up/up"! Keep going!
ignw@ignw-jumphost:~$ 
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw-csr#
ignw-csr#
ignw-csr#sh ip int brie
Interface          IP-Address      OK? Method Status      Protocol
GigabitEthernet1   10.0.0.5        YES NVRAM up           up
GigabitEthernet2   203.0.113.1    YES manual up         up
Loopback0          172.16.1.1     YES manual up         up
Loopback1          8.8.4.4        YES manual up         up
ignw-csr#
ignw-csr#sh run int g2
Building configuration...
Current configuration : 157 bytes
!
interface GigabitEthernet2
description This goes to the ASA
ip address 203.0.113.1 255.255.255.192
negotiation auto
no mop enabled
no mop sysid
end
ignw-csr#
```

netmiko\_task3.py — — Atom

```
File Edit View Selection Find Packages Help
  netmiko_task3.py
1 import sys
2 import time
3 from netmiko import ConnectHandler
4
5
6 cisco_cloud_router = {'device_type': 'cisco_ios',
7                         'ip': '10.0.0.5',
8                         'username': 'ignw',
9                         'password': 'ignw'}
10
11 connection = ConnectHandler(**cisco_cloud_router)
12
13 commands = ['interface loopback1',
14             'description IGNW was here!',
15             'ip address 8.8.4.4 255.255.255.255',
16             'no shut']
17
18 connection.config_mode()
19
20 connection.send_config_set(commands)
21 time.sleep(2)
22
23 show_output = connection.send_command('show ip int loopback1 | i Loopback1')
24 if show_output.count('up') == 2:
25     print('It looks like loopback1 is "up/up"! Way to go!')
26 else:
27     print('Something went wrong... let\'s get outa here before we break something!')
28     sys.exit()
29
30 print('Moving on to GigabitEthernet2...')
31 commands = ['interface GigabitEthernet2',
32             'description This goes to the ASA',
33             'ip address 203.0.113.1 255.255.255.192',
34             'no shut']
35
36 connection.send_config_set(commands)
37 time.sleep(2)
38
39 show_output = connection.send_command('show ip int GigabitEthernet2 | i GigabitEthernet2')
40 if show_output.count('up') == 2:
41     print('It looks like GigabitEthernet2 is "up/up"! Keep going!')
42 else:
43     print('Something went wrong... let\'s get outa here before we break something!')
44     sys.exit()
```

netmiko\_task3.py 15:50 LF UTF-8 Python 0 files

Why do you think we put that "time.sleep(2)" in there? What does it do? It does what it sounds like it should do! It simply "sleeps" the execution of the code for two seconds in this case. Why? This is some "cheap" (free really!) insurance for us. If you've ever configured an interface you know that sometimes it takes a few seconds for the router/switch to report that it is actually "up/up", so we can bake in a very simple task to sleep to ensure that the router has time to bring the interface up before checking on it.

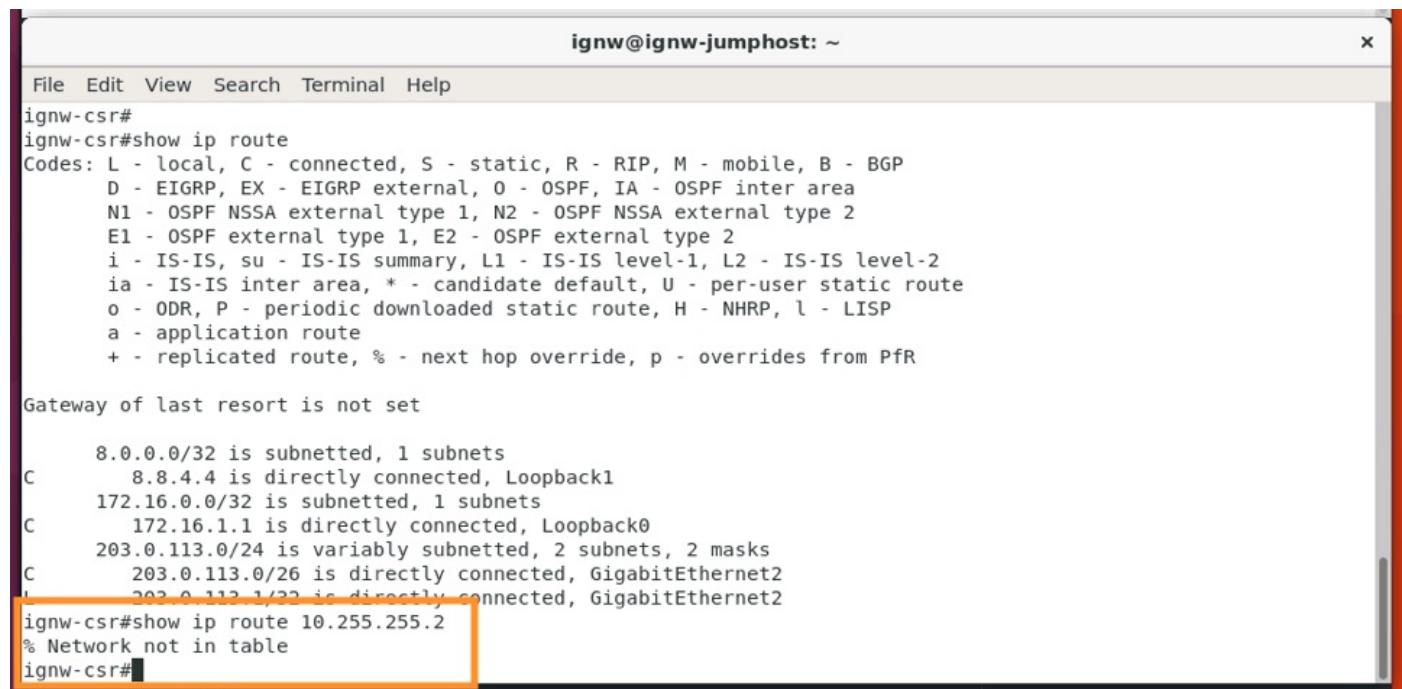
## Create a Host Route

Now that we have our interface that goes to the ASA v "up", we need to prepare a route that points to our internal node. The NXOSv's IP address will be 10.255.255.2. A host route pointing to this should look like the following:

```
ip route 10.255.255.2 255.255.255.255 203.0.113.2
```

The 203.0.113.2 IP address will be the interface on the ASA v that you will configure in the next step.

This task is again more of the same, however verification of it is a bit different. You'll probably want to check the output of "show ip route" to verify that your route gets installed in the routing table. As long as the route makes it into the table, we can assume (at this point at least) that we are good to go!



A terminal window titled "ignw@ignw-jumphost: ~" showing ASA configuration and route verification. The window includes a menu bar with File, Edit, View, Search, Terminal, Help. The terminal content shows:

```
ignw-csr#  
ignw-csr#show ip route  
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP  
      D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area  
      N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2  
      E1 - OSPF external type 1, E2 - OSPF external type 2  
      i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2  
      ia - IS-IS inter area, * - candidate default, U - per-user static route  
      o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP  
      a - application route  
      + - replicated route, % - next hop override, p - overrides from PfR  
  
Gateway of last resort is not set  
  
      8.0.0.0/32 is subnetted, 1 subnets  
C        8.8.4.4 is directly connected, Loopback1  
      172.16.0.0/32 is subnetted, 1 subnets  
C        172.16.1.1 is directly connected, Loopback0  
      203.0.113.0/24 is variably subnetted, 2 subnets, 2 masks  
C          203.0.113.0/26 is directly connected, GigabitEthernet2  
L          203.0.113.1/22 is directly connected, GigabitEthernet2  
ignw-csr#show ip route 10.255.255.2  
% Network not in table  
ignw-csr#
```

Huh, that "Network not in table" thing would probably be a good thing to trigger on to let us know if something went wrong, don't you think?

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 netmiko_task3.py
It looks like loopback1 is "up/up"! Way to go!
Moving on to GigabitEthernet2...
It looks like GigabitEthernet2 is "up/up"! Keep going!
Moving on to static routing...
It looks like the route made it into the table!
ignw@ignw-jumphost:~$
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw-csr#
ignw-csr#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
      D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
      N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
      E1 - OSPF external type 1, E2 - OSPF external type 2
      i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
      ia - IS-IS inter area, * - candidate default, U - per-user static route
      o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
      a - application route
      + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is not set

      8.0.0.0/32 is subnetted, 1 subnets
C          8.8.4.4 is directly connected, Loopback1
S          10.0.0.0/32 is subnetted, 1 subnets
          10.255.255.2 [1/0] via 203.0.113.2
          172.16.0.0/32 is subnetted, 1 subnets
C          172.16.1.1 is directly connected, Loopback0
          203.0.113.0/24 is variably subnetted, 2 subnets, 2 masks
C          203.0.113.0/26 is directly connected, GigabitEthernet2
L          203.0.113.1/32 is directly connected, GigabitEthernet2
ignw-csr#
```

```
netmiko_task3.py — — Atom
File Edit View Selection Find Packages Help
    netmiko_task3.py
18 connection.config_mode()
19
20 connection.send_config_set(commands)
21 time.sleep(2)
22
23 show_output = connection.send_command('show ip int loopback1 | i Loopback1')
24 if show_output.count('up') == 2:
25     print('It looks like loopback1 is "up/up"! Way to go!')
26 else:
27     print('Something went wrong... let\'s get outa here before we break something!')
28     sys.exit()
29
30 print('Moving on to GigabitEthernet2...')
31 commands = ['interface GigabitEthernet2',
32             'description This goes to the ASA',
33             'ip address 203.0.113.1 255.255.255.192',
34             'no shut']
35
36 connection.send_config_set(commands)
37 time.sleep(2)
38
39 show_output = connection.send_command('show ip int GigabitEthernet2 | i GigabitEthernet2')
40 if show_output.count('up') == 2:
41     print('It looks like GigabitEthernet2 is "up/up"! Keep going!')
42 else:
43     print('Something went wrong... let\'s get outa here before we break something!')
44     sys.exit()
45
46 print('Moving on to static routing...')
47
48 commands = ['ip route 10.255.255.2 255.255.255.255 203.0.113.2']
49
50 connection.send_config_set(commands)
51 time.sleep(2)
52
53 show_output = connection.send_command('show ip route 10.255.255.2')
54 if 'Network not in table' in show_output:
55     print('Something went wrong... let\'s get outa here before we break something!')
56     sys.exit()
57 else:
58     print('It looks like the route made it into the table!')
59
60 connection.send_command('wr')
61
```

## ASA Configuration

Up until now, we've only used Netmiko to interact with the CSR router. The CSR router runs a different operating system from the ASA. A quick check on the [Netmiko Github page](#) does seem to indicate that the ASA platform is supported by Netmiko. In fact it even says ASA is a "regularly tested" platform!

That seems like a good sign that we can continue to use Netmiko (if you want!), so let's keep going!

## Configure Interfaces

The ASA has two interfaces that we need to configure -- the first "pointing to" the CSR router, and the second to the NX-OSv. GigabitEthernet0/0 connects to the CSR, while GigabitEthernet0/1 connects to the Nexus. Their IP addresses should be "203.0.113.2 255.255.255.192" and "10.255.255.1 255.255.255.240" respectively.

The ASA also has a few other interface level configurations that we'll need. The first is a "nameif" -- this is a security zone on an ASA. Let's call our security zones "outside" and "dmz" to keep things simple. The ASA will also need a "security-level" associated to each interface. 100 is the most secure and is usually reserved for management (as in this case), so let's use "0" for the outside and "100" for the inside.

Complete interface configurations should look like this:

```
interface GigabitEthernet0/0
description Connected to CSR
nameif outside
security-level 0
ip address 203.0.113.2 255.255.255.192
```

And...

```
interface GigabitEthernet0/0
description Connected to NX-OSv
nameif inside
security-level 100
ip address 10.255.255.1 255.255.255.240
```

With all that in mind, let's try to keep on keeping on how we've been doing! We obviously will need to update some information -- IP address is of course different, also the password is different. We also can prep our configurations.

```

1 from netmiko import ConnectHandler
2
3 asav = {'device_type': 'cisco_ios',
4         'ip': '10.0.0.8',
5         'username': 'ignw',
6         'password': 'ignw'}
7 connection = ConnectHandler(**asav)
8
9 print('Starting on GigabitEthernet0/0...')
10
11 commands = ['interface GigabitEthernet0/0',
12             'description Connected to CSR',
13             'nameif outside',
14             'security-level 0',
15             'ip address 203.0.113.2 255.255.255.192',
16             'no shut']
17
18 connection.send_config_set(commands)
19

```

Did your script run?

```

ignw@ignw-jumphost:~/tmp$ Starting on GigabitEthernet0/0...
Traceback (most recent call last):
  File "tmp.py", line 19, in <module>
    connection.send_config_set(commands)
  File "/usr/local/lib/python3.6/dist-packages/netmiko/base_connection.py", line 1292, in send_config_set
    output = self.config_mode(*cfg_mode_args)
  File "/usr/local/lib/python3.6/dist-packages/netmiko/cisco_base_connection.py", line 42, in config_mode
    pattern=pattern)
  File "/usr/local/lib/python3.6/dist-packages/netmiko/base_connection.py", line 1210, in config_mode
    raise ValueError("Failed to enter configuration mode.")
ValueError: Failed to enter configuration mode.
ignw@ignw-jumphost:~$ 

```

It seems we are running into an issue! "Failed to enter configuration mode." -- well, that actually makes a lot of sense since we have an enable password on the ASA. Take a look at the Github documentation for Netmiko, do you see a way to ensure that our connection object knows about the enable password?

## Examples:

Create a dictionary representing the device.

Supported device\_types can be found [here](#), see CLASS\_MAPPER keys.

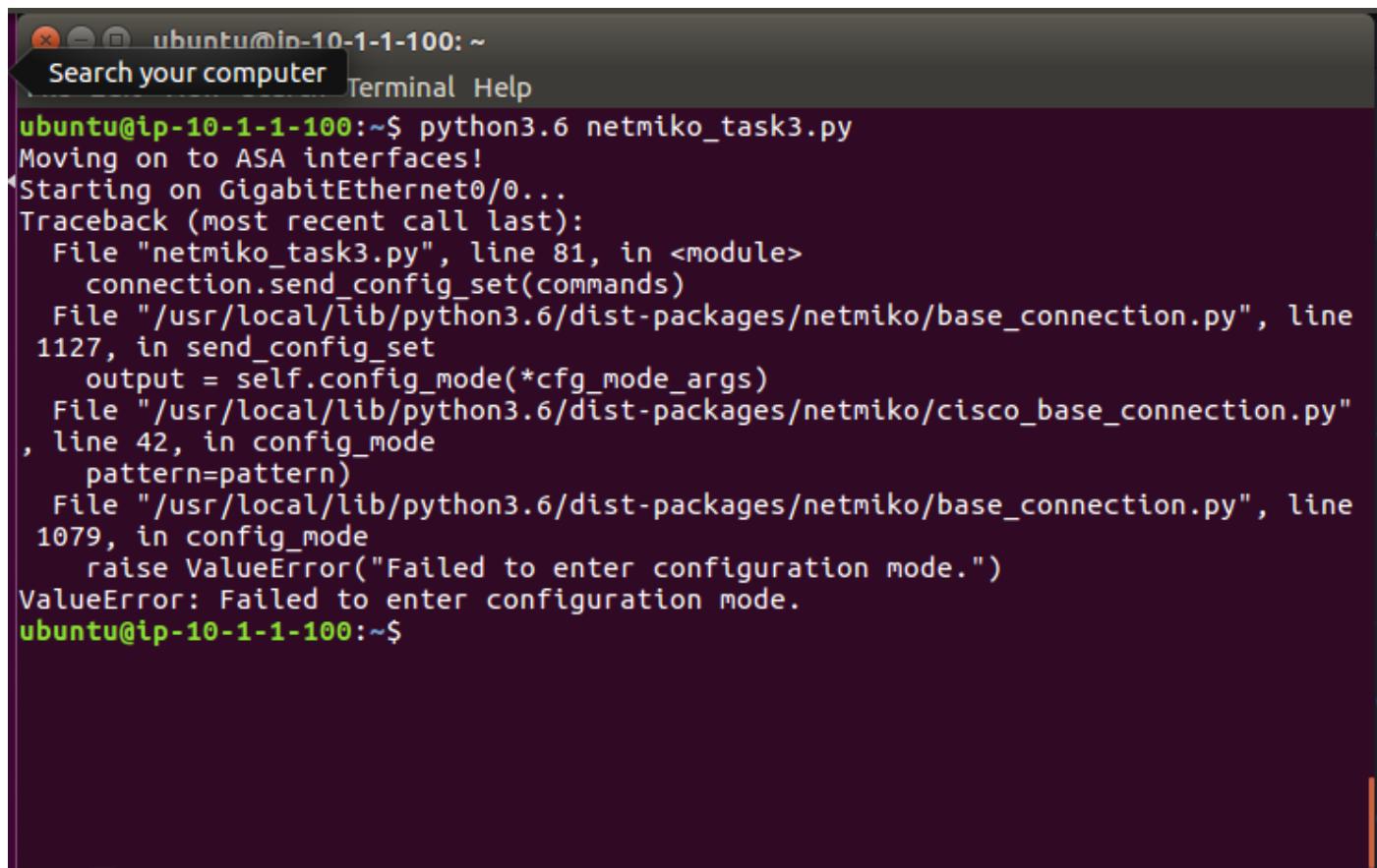
```
from netmiko import ConnectHandler

cisco_881 = {
    'device_type': 'cisco_ios',
    'ip': '10.10.10.10',
    'username': 'test',
    'password': 'password',
    'port': 8022,
    'secret': 'secret',          # optional, defaults to ''
    'verbose': False,           # optional, defaults to False
}
```

Try to update your dictionary to contain the "secret" key and enable password of "ignw" (strong password huh?).

```
1 from netmiko import ConnectHandler
2
3 asav = {'device_type': 'cisco_ios',
4         'ip': '10.0.0.8',
5         'username': 'ignw',
6         'password': 'ignw',
7         'secret': 'ignw'}
8 connection = ConnectHandler(**asav)
9
10 print('Starting on GigabitEthernet0/0...')
11
12 commands = ['interface GigabitEthernet0/0',
13             'description Connected to CSR',
14             'nameif outside',
15             'security-level 0',
16             'ip address 203.0.113.2 255.255.255.192',
17             'no shut']
18
19 connection.send_config_set(commands)
20 |
```

Does your script run now?



A screenshot of a terminal window titled "ubuntu@ip-10-1-1-100: ~". The window has a dark background with light-colored text. At the top, there's a menu bar with "Search your computer", "Terminal", and "Help". The main area of the terminal shows the following command-line session:

```
ubuntu@ip-10-1-1-100:~$ python3.6 netmiko_task3.py
Moving on to ASA interfaces!
Starting on GigabitEthernet0/0...
Traceback (most recent call last):
  File "netmiko_task3.py", line 81, in <module>
    connection.send_config_set(commands)
  File "/usr/local/lib/python3.6/dist-packages/netmiko/base_connection.py", line
1127, in send_config_set
    output = self.config_mode(*cfg_mode_args)
  File "/usr/local/lib/python3.6/dist-packages/netmiko/cisco_base_connection.py"
, line 42, in config_mode
    pattern=pattern)
  File "/usr/local/lib/python3.6/dist-packages/netmiko/base_connection.py", line
1079, in config_mode
    raise ValueError("Failed to enter configuration mode.")
ValueError: Failed to enter configuration mode.
ubuntu@ip-10-1-1-100:~$
```

Uhoh, still no dice! What about that "device\_type" value? We aren't working with "cisco\_ios" anymore, this is an ASA! While this may seem very specific to Cisco IOS and/or Cisco ASAs, or even Netmiko (and it is of course), it is very common to need to do a bit of digging into documentation to investigate things like this to get your code to work. Can you find (or even take a guess!) the device type we should be using?

One more thing to know for the ASA interface configuration -- the validation will be slightly different as well, you may want to use the "show interface ip brief" command; it is very similar to the "show ip interface brief" used previously.

```

ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 netmiko_task3.py
It looks like loopback1 is "up/up"! Way to go!
Moving on to GigabitEthernet2...
It looks like GigabitEthernet2 is "up/up"! Keep going!
Moving on to static routing...
It looks like the route made it into the table!
Moving on to ASA interfaces!
Starting on GigabitEthernet0/0...
Starting on GigabitEthernet0/1...
It looks like GigabitEthernet0/0 is "up/up"!
It looks like GigabitEthernet0/1 is "up/up"!
ignw@ignw-jumphost:~$ █

ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
description Connected to CSR
nameif outside
security-level 0
ip address 203.0.113.2 255.255.255.192
ignw-asav# sh run int g0/1
!
interface GigabitEthernet0/1
description Connected to NX-OSv
nameif inside
security-level 100
ip address 10.255.255.1 255.255.255.240
ignw-asav# sh int ip brie
Interface          IP-Address      OK? Method Status       Protocol
GigabitEthernet0/0  203.0.113.2    YES manual up        up
GigabitEthernet0/1  10.255.255.1   YES manual up        up
GigabitEthernet0/2  unassigned     YES unset administratively down up
GigabitEthernet0/3  unassigned     YES unset administratively down up
GigabitEthernet0/4  unassigned     YES unset administratively down up
GigabitEthernet0/5  unassigned     YES unset administratively down up
GigabitEthernet0/6  unassigned     YES unset administratively down up
GigabitEthernet0/7  unassigned     YES unset administratively down up
GigabitEthernet0/8  unassigned     YES unset administratively down up
Management0/0       10.0.0.8      YES CONFIG up       up
ignw-asav# █

```

netmiko\_task3.py — — Atom

```
File Edit View Selection Find Packages Help
    netmiko_task3.py

65 asav = {'device_type': 'cisco_asa',
66     'ip': '10.0.0.8',
67     'username': 'ignw',
68     'password': 'ignw',
69     'secret': 'ignw'}
70 connection = ConnectHandler(**asav)
71
72 print('Starting on GigabitEthernet0/0...')
73
74 commands = ['interface GigabitEthernet0/0',
75             'description Connected to CSR',
76             'nameif outside',
77             'security-level 0',
78             'ip address 203.0.113.2 255.255.255.192',
79             'no shut']
80
81 connection.send_config_set(commands)
82
83 print('Starting on GigabitEthernet0/1...')
84
85 commands = ['interface GigabitEthernet0/1',
86             'description Connected to NX-OSv',
87             'nameif inside',
88             'security-level 100',
89             'ip address 10.255.255.1 255.255.255.240',
90             'no shut']
91
92 connection.send_config_set(commands)
93 time.sleep(2)
94
95 show_output = connection.send_command('show int ip brief | i GigabitEthernet0/0')
96 if show_output.count('up') == 2:
97     print('It looks like GigabitEthernet0/0 is "up/up"!')
98 else:
99     print('Something went wrong... let\'s get outa here before we break something!')
100    sys.exit()
101 show_output = connection.send_command('show int ip brief | i GigabitEthernet0/1')
102 if show_output.count('up') == 2:
103     print('It looks like GigabitEthernet0/1 is "up/up"!')
104 else:
105     print('Something went wrong... let\'s get outa here before we break something!')
106    sys.exit()
107
108
netmiko_task3.py  1:1
```

LF UTF-8 Python 0 files

## Configure a Host Route

Now that the interfaces are "up/up" we need to get a route configured on the ASA v so that it has reachability to our previously created loopback. Our route should look like the following:

```
route outside 8.8.4.4 255.255.255.255 203.0.113.1
```

Verification is just a tiny bit different from IOS as well -- the "show route [prefix]" command takes place of the "show ip route [prefix]".

With that in mind, can you get the route setup on the ASA?

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 netmiko_task3.py
It looks like loopback1 is "up/up"! Way to go!
Moving on to GigabitEthernet2...
It looks like GigabitEthernet2 is "up/up"! Keep going!
Moving on to static routing...
It looks like the route made it into the table!
Moving on to ASA interfaces!
Starting on GigabitEthernet0/0...
Starting on GigabitEthernet0/1...
It looks like GigabitEthernet0/0 is "up/up"!
It looks like GigabitEthernet0/1 is "up/up"!
Moving on to static routing...
It looks like the route made it into the table!
ignw@ignw-jumphost:~$
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw-asav# sh run route
route outside 8.8.4.4 255.255.255.255 203.0.113.1 1
route management 10.10.0.0 255.255.0.0 10.0.0.1 1
ignw-asav# sh route

Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
      D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
      N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
      E1 - OSPF external type 1, E2 - OSPF external type 2, V - VPN
      i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
      ia - IS-IS inter area, * - candidate default, U - per-user static route
      o - ODR, P - periodic downloaded static route, + - replicated route
Gateway of last resort is not set

S     8.8.4.4 255.255.255.255 [1/0] via 203.0.113.1, outside
C     10.0.0.0 255.255.255.0 is directly connected, management
L     10.0.0.8 255.255.255.255 is directly connected, management
S     10.10.0.0 255.255.0.0 [1/0] via 10.0.0.1, management
C     10.255.255.0 255.255.255.240 is directly connected, inside
L     10.255.255.1 255.255.255.255 is directly connected, inside
C     203.0.113.0 255.255.255.192 is directly connected, outside
L     203.0.113.2 255.255.255.255 is directly connected, outside

ignw-asav#
```

netmiko\_task3.py — — Atom

File Edit View Selection Find Packages Help

```
netmiko_task3.py

    'ip address 203.0.113.2 255.255.255.192',
    'no shut']

80
81 connection.send_config_set(commands)
82
83 print('Starting on GigabitEthernet0/1...')
84
85 commands = ['interface GigabitEthernet0/1',
86             'description Connected to NX-OSv',
87             'nameif inside',
88             'security-level 100',
89             'ip address 10.255.255.1 255.255.255.240',
90             'no shut']
91
92 connection.send_config_set(commands)
93 time.sleep(2)
94
95 show_output = connection.send_command('show int ip brief | i GigabitEthernet0/0')
96 if show_output.count('up') == 2:
97     print('It looks like GigabitEthernet0/0 is "up/up"!')
98 else:
99     print('Something went wrong... let\'s get outa here before we break something!')
100    sys.exit()
101 show_output = connection.send_command('show int ip brief | i GigabitEthernet0/1')
102 if show_output.count('up') == 2:
103     print('It looks like GigabitEthernet0/1 is "up/up"!')
104 else:
105     print('Something went wrong... let\'s get outa here before we break something!')
106    sys.exit()
107
108 print('Moving on to static routing...')
109
110 commands = ['route outside 8.8.4.4 255.255.255.255 203.0.113.1']
111
112 connection.send_config_set(commands)
113 time.sleep(2)
114
115 show_output = connection.send_command('show route 8.8.4.4')
116 if 'Network not in table' in show_output:
117     print('Something went wrong... let\'s get outa here before we break something!')
118     sys.exit()
119 else:
120     print('It looks like the route made it into the table!')
```

netmiko\_task3.py 1:1 LF UTF-8 Python 0 files

## Configure and Apply ACL

Finally, configure an ACL on ASA to allow ICMP from your newly created loopback on the CSR to the host in the "DMZ" of your ASA visa versa. Your ACL should look something like the following:

```
access-list outside_in extended permit icmp any any
```

It is then applied to the interface:

```
access-group outside_in in interface outside
```

Instead of validating our configurations this time, we'll simply move on to a "functional" test -- actually pinging from our server.



netmiko\_task3.py — — Atom

File Edit View Selection Find Packages Help

```
netmiko_task3.py

87     'nameif inside',
88     'security-level 100',
89     'ip address 10.255.255.1 255.255.255.240',
90     'no shut']
91
92 connection.send_config_set(commands)
93 time.sleep(2)
94
95 show_output = connection.send_command('show int ip brief | i GigabitEthernet0/0')
96 if show_output.count('up') == 2:
97     print('It looks like GigabitEthernet0/0 is "up/up"!')
98 else:
99     print('Something went wrong... let\'s get outa here before we break something!')
100    sys.exit()
101 show_output = connection.send_command('show int ip brief | i GigabitEthernet0/1')
102 if show_output.count('up') == 2:
103     print('It looks like GigabitEthernet0/1 is "up/up"!')
104 else:
105     print('Something went wrong... let\'s get outa here before we break something!')
106    sys.exit()
107
108 print('Moving on to static routing...')
109
110 commands = ['route outside 8.8.4.4 255.255.255.255 203.0.113.1']
111
112 connection.send_config_set(commands)
113 time.sleep(2)
114
115 show_output = connection.send_command('show route 8.8.4.4')
116 if 'Network not in table' in show_output:
117     print('Something went wrong... let\'s get outa here before we break something!')
118     sys.exit()
119 else:
120     print('It looks like the route made it into the table!')
121
122 print('Moving on to access-list configuration...')
123
124 commands = ['access-list outside_in extended permit icmp any any',
125             'access-group outside_in in interface outside']
126
127 connection.send_config_set(commands)
128 connection.send_command('wr')
129
130
```

netmiko\_task3.py 1:1 LF UTF-8 Python 0 files

## Configuring NXOS Interface and Route

Before we can validate we need to quickly fire up an interface and a route on the NXOS device. It is pretty common to have a L2 segment between core and firewall devices (to facilitate HA pairs for example), so we'll

configure an SVI, and then a trunk with our VLAN as the native VLAN. Your configurations should look something like this:

```
vlan 1000
interface vlan1000
description To ASA
ip address 10.255.255.2 255.255.255.240
no shut
interface Ethernet1/2
switchport
switchport mode trunk
switchport trunk native vlan 1000
no shut
```

You can wrap this all up in one large "config\_set" with Netmiko. Just like when we switched to the ASA, make sure you check and set the "device\_type" properly! Also recall that syntax with NXOS is going to be a tick different than with the ASA or CSR, so be careful on your validations!

netmiko\_task3.py — — Atom

```
File Edit View Selection Find Packages Help
    netmiko_task3.py
120
127 connection.send_config_set(commands)
128 connection.send_command('wr')
129
130 print('Moving on to NXOSv Port')
131
132 nxosv = {'device_type': 'cisco_nxos',
133             'ip': '10.0.0.6',
134             'username': 'ignw',
135             'password': 'ignw'}
136 connection = ConnectHandler(**nxosv)
137
138 commands = ['feature interface-vlan',
139             'vlan 1000',
140             'interface vlan1000',
141             'description To ASAv',
142             'ip address 10.255.255.2 255.255.255.240',
143             'no shut',
144             'interface Ethernet1/2',
145             'switchport',
146             'no shut',
147             'switchport mode trunk',
148             'switchport trunk native vlan 1000',
149             'no shut']
150
151 connection.send_config_set(commands)
152 time.sleep(2)
153
154 show_output = connection.send_command('show ip int Vlan1000 | i Vlan1000')
155 if show_output.count('up') == 3:
156     print('It looks like Vlan1000 is "up/up"! Way to go!')
157 else:
158     print('Something went wrong... let\'s get outa here before we break something!')
159     sys.exit()
160 show_output = connection.send_command('show int status | i Eth1/2')
161 if show_output.count('connected') == 1:
162     print('It looks like Eth1/2 is "up/up"! Way to go!')
163 else:
164     print('Something went wrong... let\'s get outa here before we break something!')
165     sys.exit()
166
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 netmiko_task3.py
It looks like loopback1 is "up/up"! Way to go!
Moving on to GigabitEthernet2...
It looks like GigabitEthernet2 is "up/up"! Keep going!
Moving on to static routing...
It looks like the route made it into the table!
Moving on to ASA interfaces!
Starting on GigabitEthernet0/0...
Starting on GigabitEthernet0/1...
It looks like GigabitEthernet0/0 is "up/up"!
It looks like GigabitEthernet0/1 is "up/up"!
Moving on to static routing...
It looks like the route made it into the table!
Moving on to access-list configuration...
Moving on to NXOSv Port
It looks like Vlan1000 is "up/up"! Way to go!
It looks like Eth1/2 is "up/up"! Way to go!
```

## Validate Success!

Now we *should* have ICMP permitted from the loopback1 IP address on the CSR, through the ASA, and over on to the NXOS node. It is probably a good idea to check this manually before writing any code -- we want to make sure you're not troubleshooting your script when something else is the culprit!

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ ssh 10.0.0.5
Password:

ignw-csr#ping 10.255.255.2 sour
ignw-csr#ping 10.255.255.2 source 8.8.4.4
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.255.255.2, timeout is 2 seconds:
Packet sent with a source address of 8.8.4.4
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/3/5 ms
ignw-csr#
```

If you can't ping from the CSR, make sure you are sourcing your ping from the correct IP address -- that is the only route that the DMZ host has!

Assuming all is well, it's time to add some functional validation to our script.

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 netmiko_task3.py
It looks like loopback1 is "up/up"! Way to go!
Moving on to GigabitEthernet2...
It looks like GigabitEthernet2 is "up/up"! Keep going!
Moving on to static routing...
It looks like the route made it into the table!
Moving on to ASA interfaces!
Starting on GigabitEthernet0/0...
Starting on GigabitEthernet0/1...
It looks like GigabitEthernet0/0 is "up/up"!
It looks like GigabitEthernet0/1 is "up/up"!
Moving on to static routing...
It looks like the route made it into the table!
Moving on to access-list configuration...
Moving on to NXOSv Port
It looks like Vlan1000 is "up/up"! Way to go!
It looks like Eth1/2 is "up/up"! Way to go!
Testing connectivity from CSR
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.255.255.2, timeout is 2 seconds:
Packet sent with a source address of 8.8.4.4
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/3/7 ms
ignw@ignw-jumphost:~$
```

```
164     print('Something went wrong... let\'s get outa here before we break something!')
165     sys.exit()
166
167 print('Testing connectivity from CSR')
168
169 connection = ConnectHandler(**cisco_cloud_router)
170 show_output = connection.send_command('ping 10.255.255.2 source 8.8.4.4')
171 print(show_output)
172
```

netmiko\_task3.py 171:19

LF UTF-8 Python 0 files

## Bonus 1

Validate your script from the DMZ Node side -- you may use pexpect for this, or something completely different!

## Bonus 2

Validate your script by taking a capture on the ASA v -- use this to match quantity of ICMP packets the ASA v processes so you **REALLY** know it works!

In general validation on networking (and to a lesser extent) systems is difficult at the moment. Telemetry is all the buzz and vendors are rapidly adding streaming telemetry capabilities to devices. As these features are more and more prevalent Python and other languages will become more and more powerful a tool!

# Getting Started APIs with Postman

## Overview and Objectives

In this lab you will use Postman as a tool to learn about APIs.

Objectives:

- Gain a basic familiarity with Postman
- Get authenticated to an API
- Capture data from the API
- Make configuration changes via the API

## APIs and Authentication

Now that you've learned a little bit about APIs in general, and "REST" APIs specifically it's time to start interacting with them yourself! The first thing that you'll need to consider is how you authenticate to the services you would like to interact with. In much the same way that you may have different authentication methods or procedures for logging into your email, or to your bank accounts (simple authentication, two factor authentication, user/pass plus a pin, etc.), different APIs have different authentication requirements.

While there may be platform/service specific requirements for authentication, many APIs use one of the three following authentication/authorization (there is a difference!) mechanisms:

### HTTP Basic Authentication

This is the simplest method by far, as the name "Basic" would imply! In this approach a simple username and password is included in the HTTP header of each and every request to a system. By itself this is hugely insecure! HTTP is of course an unencrypted format, which leaves the username and password wide open in clear text! Typically the username/password is encoded in base64 (an encoding scheme) which does obfuscate the credentials (kind of), erroneously leading some to term this "encryption" -- it is not! Base64 encoded data can simply be decoded! Using HTTPS (TSL) does of course provide transport layer encryption, but does so at a cost of additional overhead.

In general, without TLS, HTTP Basic Authentication should be regarded as highly insecure and used only where appropriate! With TLS, HTTP Basic Authentication is clearly much more secure, however it is not the most efficient mechanism as the username and password must be sent in each and every request, this is likely perfectly acceptable for most applications, but at very high scale (of requests/users) this may be an issue.

### API Key / Token

An API Key or Token is a string that is assigned to a user upon successful authentication (through registering with a service such as Twitter, or authenticating with a service via an API request). Every subsequent API request includes the token indicating that the request is coming from a trusted/known source. This is a very simple approach that is fast, easy, and efficient.

There are potential security gaps with this approach in that a token could be intercepted and used by a malicious third party. To combat this some platforms using this type of authentication will set finite lifetimes for the validity of a token (10 minutes for example), forcing users to re-authenticate/re-generate a token to continue using the service.

### OAuth

OAuth is an open standard for authorization. One of the principle ideas of OAuth is to be able to authenticate to a service *without giving away your password*. If you've ever used GMail or Twitter or similar services and authorized third party applications to access your accounts you likely did so by "authorizing" them -- not by providing your credentials directly to the third party. All of this was likely OAuth!

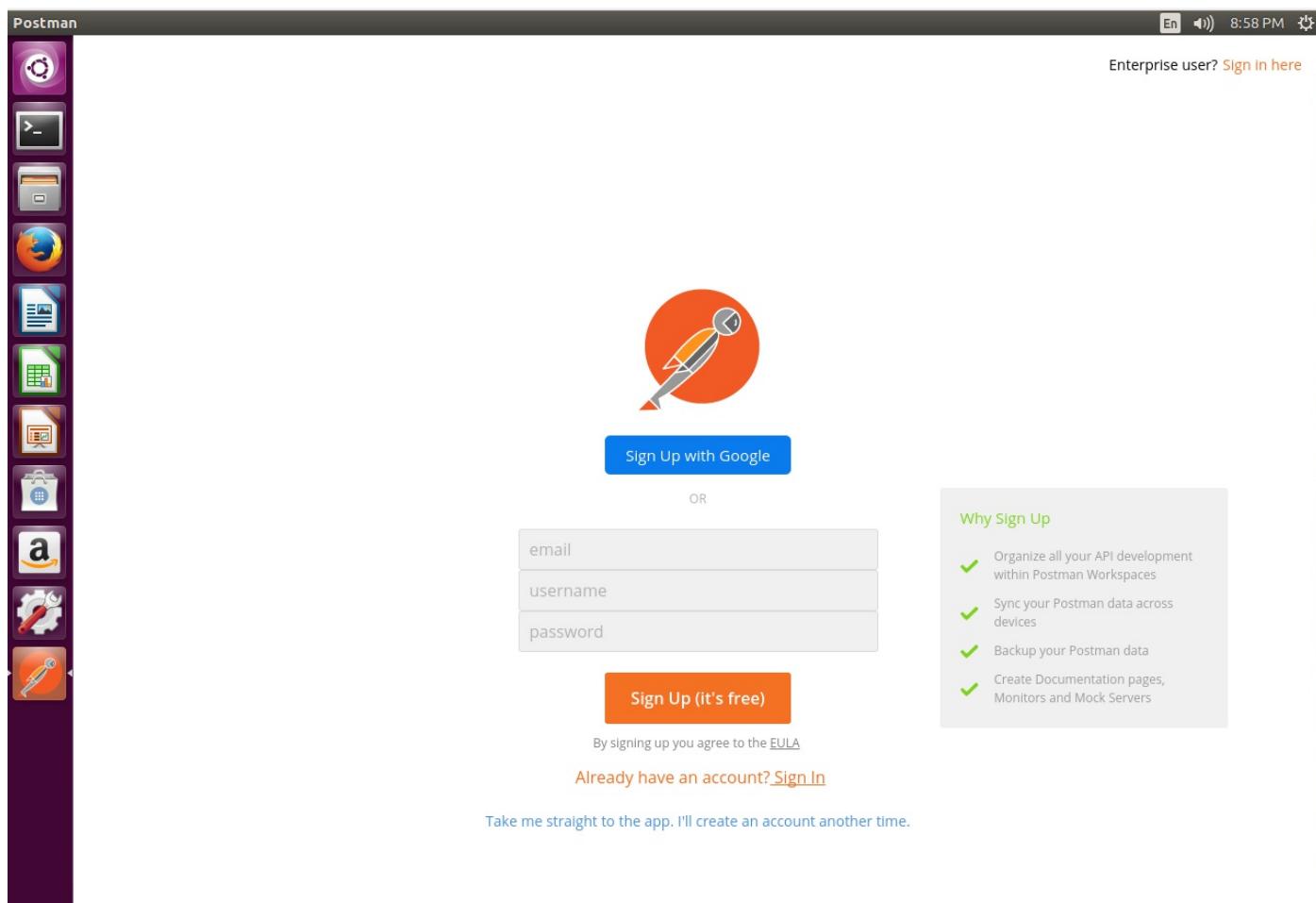
Sticking with the GMail/Twitter example -- you may indicate to a third party that you wish to have it integrated into your account. The third party will ask the service (GMail/Twitter) for a token which will then follow you to the service so that you can authenticate (with the token in your request) and allow the third party service access. Once you've approved that the third party can then get a new token that permits them access (as opposed to the original token just used for *requesting* access). While that is a gross oversimplification, that is the general idea!

In the following tasks you will work with HTTP Basic Authentication on the Cisco ASA v platform, however it is important to understand a little about other authentication methods in case you encounter them on different platforms.

## Getting Familiar with Postman

Postman is a GUI tool used for API development and testing, and is a great platform for beginning to learn how to interact with APIs.

Open Postman by clicking on the top icon on the dock. Type "postman" in the provided search field, and then click on the Atom icon to launch it.

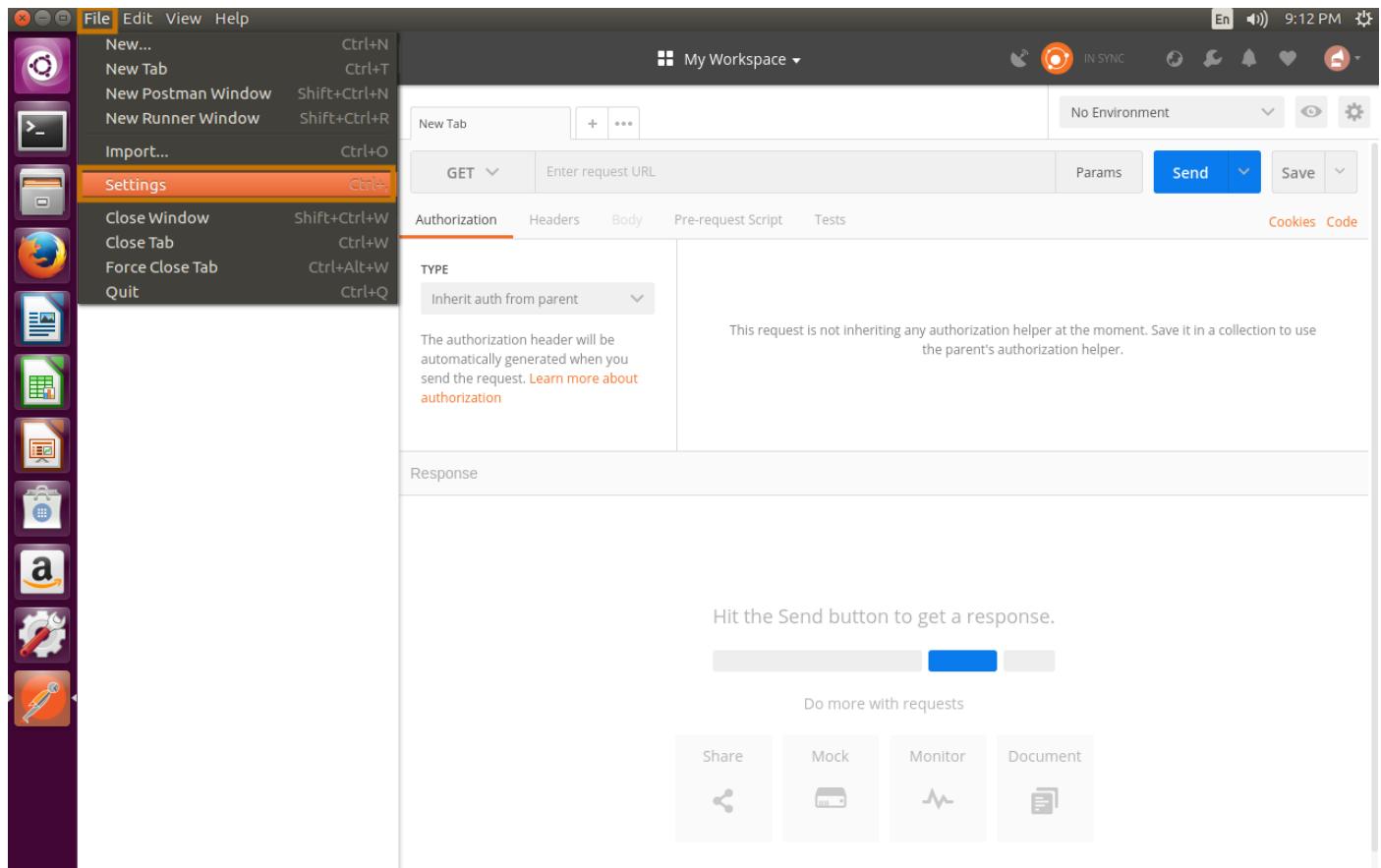


If you've used Postman before, feel free to sign in with your account (the AWS instance will be destroyed after the class; no credentials or other data will be saved). If you have not used Postman before, you can sign in with a Google account if you have one, otherwise enter an email, username, and password in the provided fields and click "Sign Up (it's free)".

Postman will send an email to verify your account; this can be ignored if you prefer.

The screenshot shows the Postman application interface. On the left is a vertical toolbar with icons for various functions like New, Import, Runner, and others. The main workspace has a header bar with 'My Workspace' and various status indicators. Below the header is a search bar and a 'New Tab' button. The main area is divided into sections: 'Authorization' (selected), 'Headers', 'Body', 'Pre-request Script', and 'Tests'. Under 'Authorization', it says 'TYPE' and 'Inherit auth from parent'. A note states: 'The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)'. To the right, there's a message: 'This request is not inheriting any authorization helper at the moment. Save it in a collection to use the parent's authorization helper.' At the bottom of the request editor, it says 'Hit the Send button to get a response.' Below the editor are buttons for 'Share', 'Mock', 'Monitor', and 'Document'.

Before we get started, we'll need to disable SSL Certificate verification as the ASA has a self-signed certificate which Postman does not like! Click on "File", then "Settings".



Ensure that the "SSL certificate verification" is set to "OFF".

SETTINGS X

General Themes Shortcuts Data Add-ons Sync Certificates Proxy Update About

**REQUEST**

Trim keys and values in request body  OFF

SSL certificate verification  OFF

Always open requests in new tab  OFF

Language detection

Request timeout in ms (0 for infinity)

**HEADERS**

Send no-cache header  ON

Send Postman Token header  ON

Retain headers when clicking on links  OFF

Automatically follow redirects  ON

Send anonymous usage data to Postman  ON

**USER INTERFACE**

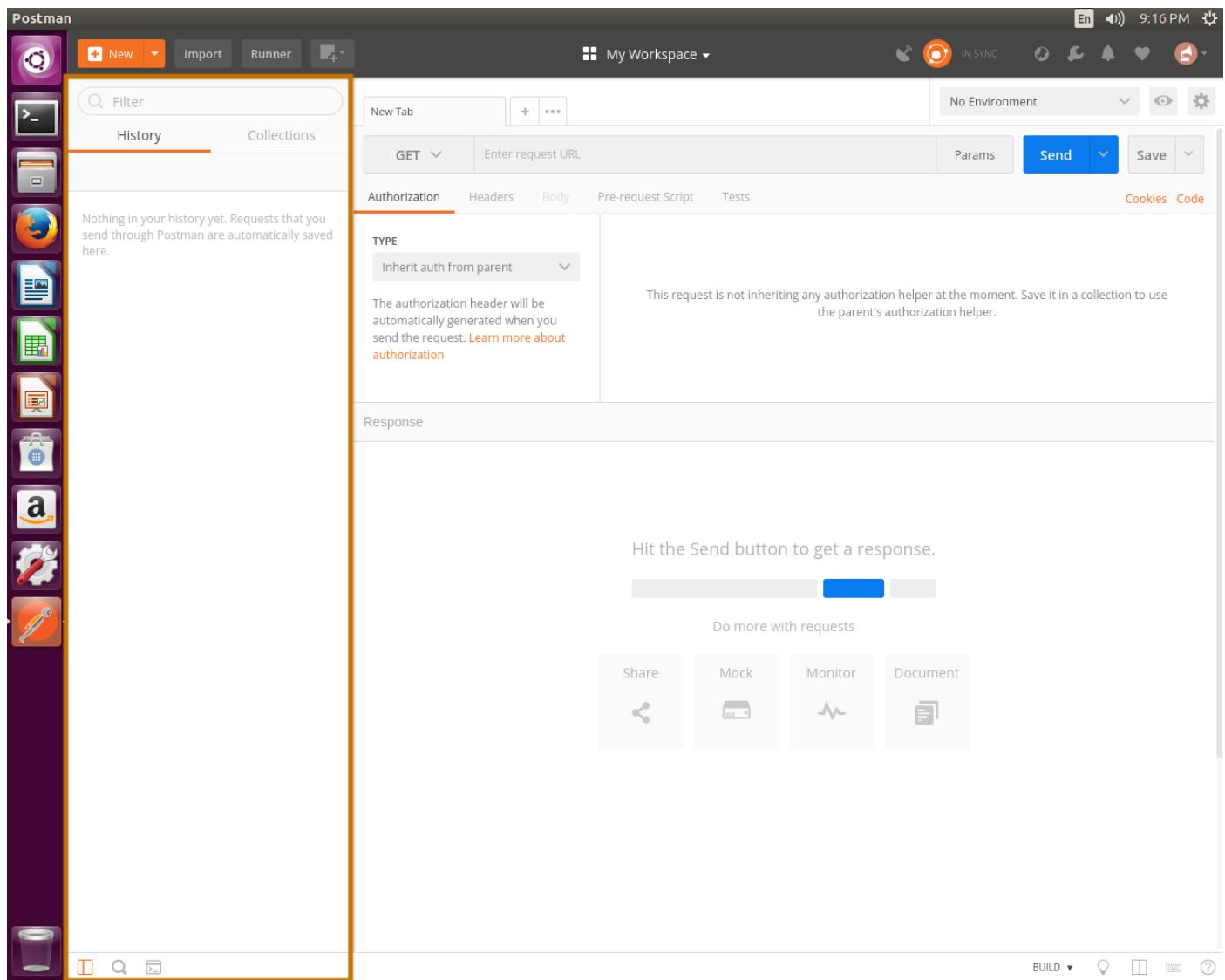
Editor Font Size (px)

Two-pane view (*beta*)  OFF

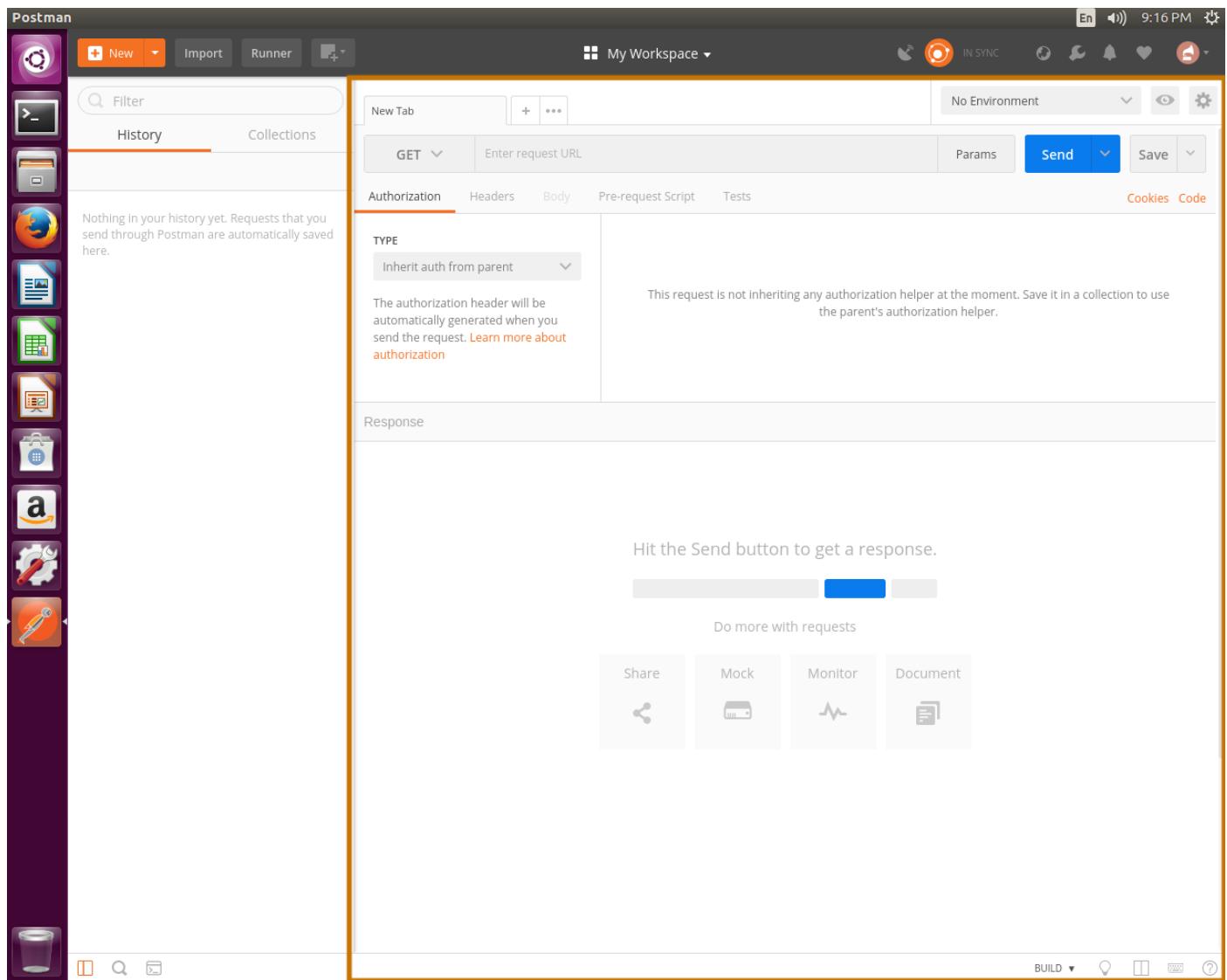
Variable autocomplete  ON

Once you've disabled SSL verification, close the Settings pane.

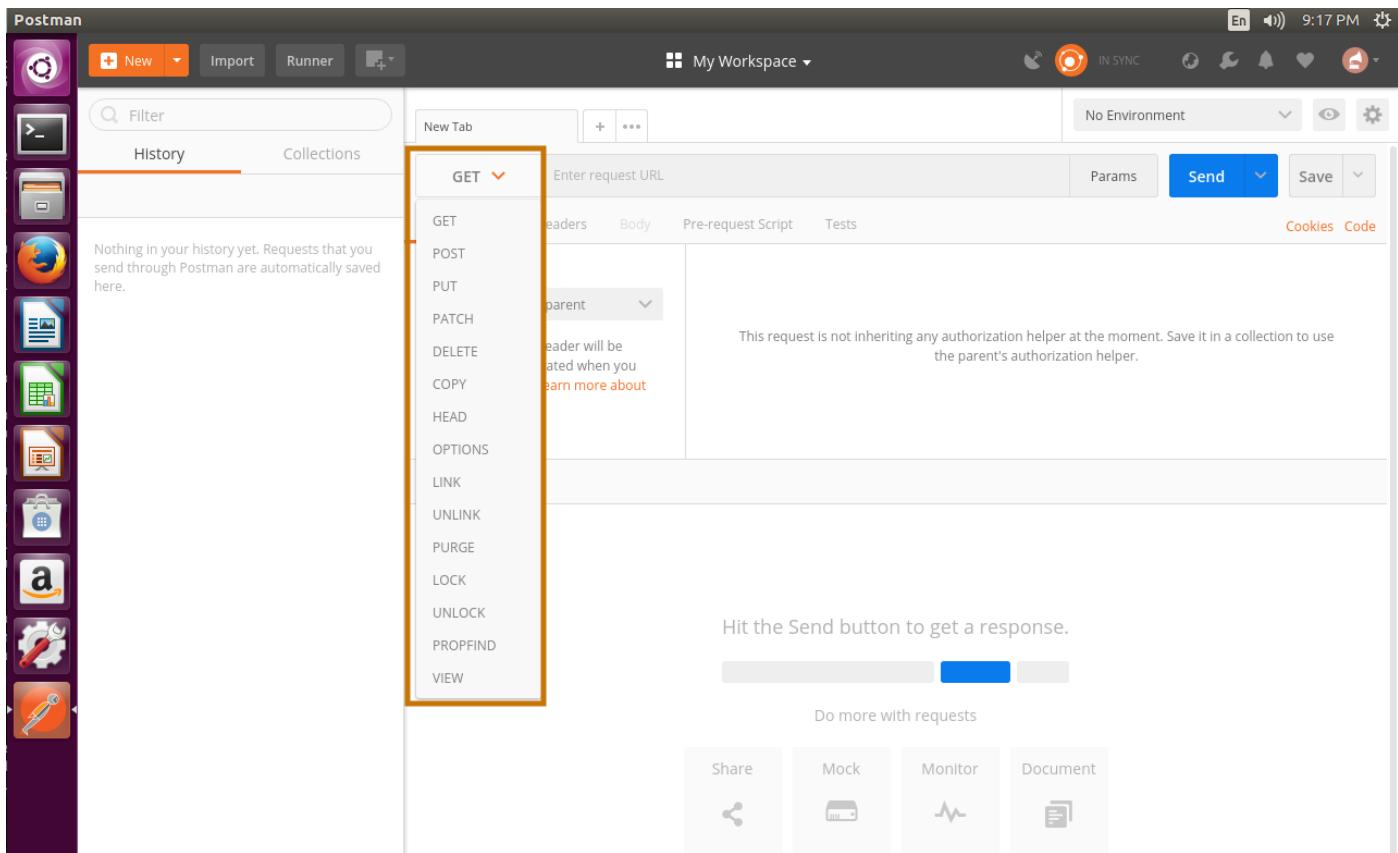
Your Postman client is made up of a few primary sections. The "navigation pane" on the left side of the screen contains a history and Collections tab. You'll use these to navigate through your requests.



The right pane is the "work pane" -- this is where we will craft our API calls.



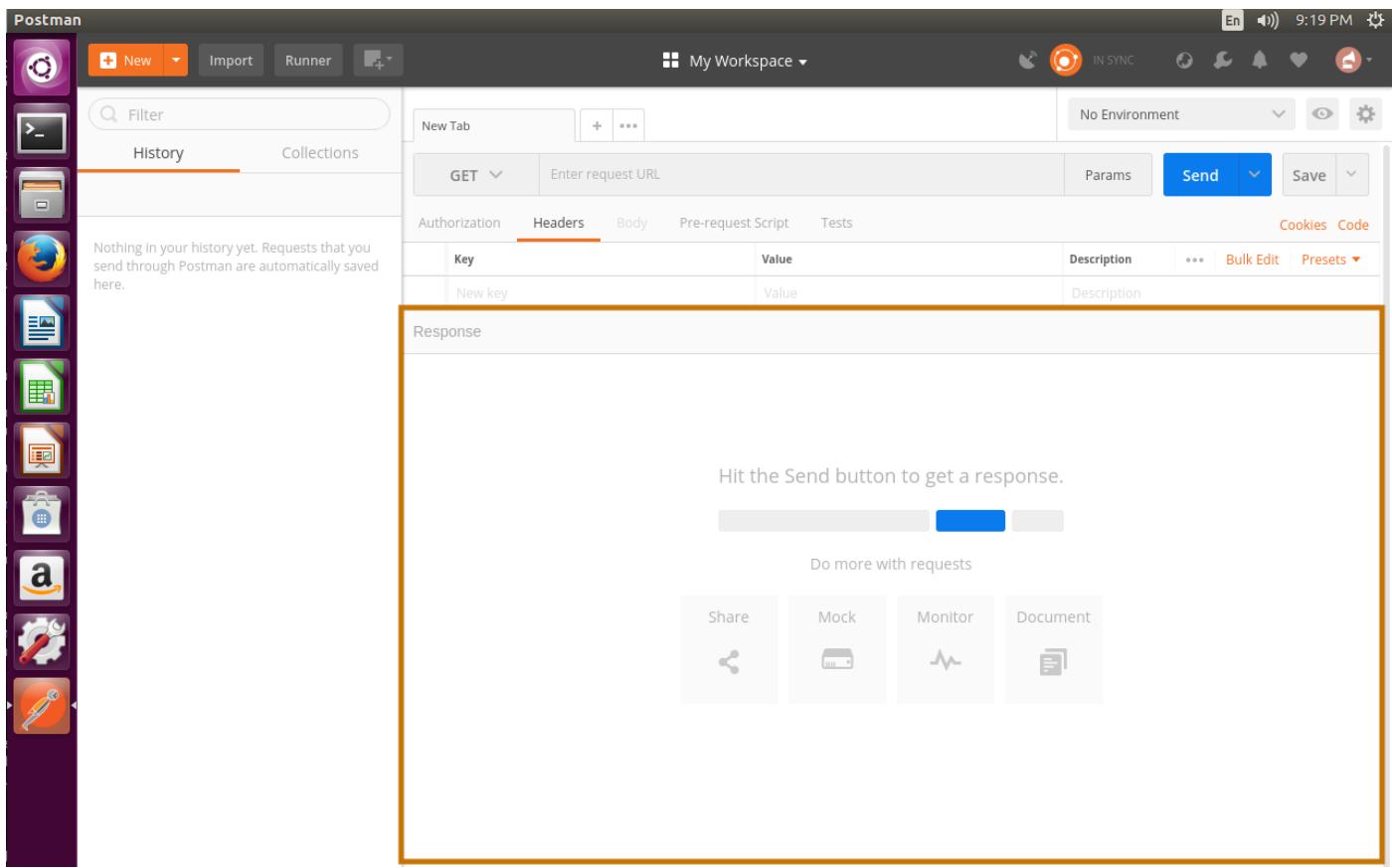
Within the work pane, there is a drop down menu that selects the *type* of HTTP request that you would like to make -- there are a lot of options, but for now we'll only need to worry about "GET".



Under the method drop down and URL field several tabs provide options for crafting our HTTP request -- Authorization, Headers, and Body are the ones we'll need to worry about -- note that "Body" is grayed out if the method is set to GET or another method that does not require a body.

The screenshot shows the Postman application interface. On the left is a sidebar with various icons for creating new requests, importing files, and running tests. The main workspace is titled "My Workspace" and contains a "History" section which is currently empty, stating "Nothing in your history yet. Requests that you send through Postman are automatically saved here." Below this is a request configuration area with tabs for "Authorization", "Headers" (which is selected and highlighted in orange), "Body", "Pre-request Script", and "Tests". The "Headers" tab displays a table with one row: "Key" (New key) and "Value" (Value). To the right of the table are buttons for "Description", "Bulk Edit", and "Presets". Below the headers is a large empty area labeled "Response" with the placeholder text "Hit the Send button to get a response." At the bottom of the workspace are four buttons: "Share" (with a user icon), "Mock" (with a server icon), "Monitor" (with a graph icon), and "Document" (with a document icon).

Finally, the bottom portion of the work pane is where the response will be displayed once we make our calls.



## Authenticating to the ASA

Now that you've got the lay of the land with Postman, it's time to build your first request!

The first thing we need to determine is how we authenticate to the ASA -- this will determine how we craft our request. Typically this first step is well documented as nobody would be able to use an API service if they can't authenticate! In this case we can refer to the Cisco [documentation](#) which indicates that we need to use "basic authentication" (HTTP Basic Authentication). We can also see that a URL is provided:

```
https://10.0.0.8/api/objects/networkobjects
```

Finally, a "content type" of "application/json" is provided for us. Hmm... no mention of what type of method, however the fact that there is no payload outlined should give an indication that this is going to be a GET method.

In your Postman client, set the HTTP method to GET, and enter the URL from above, with your ASA IP address of 10.0.0.8. At the very least the first part of this URL should be pretty self-explanatory -- "HTTPS" because we want to use TLS encryption, and the IP address of the ASA because of course that is the device we would like to interact with. Next we have "api/objects/networkobjects" -- the "api" part makes sense too since we are (hoping to be!) interacting with an API. The rest we'll cover in a bit, but for now its just an "endpoint" that we can poke to validate that we are properly authenticated and communicating with the ASA.

**Note:** Make sure you're using the correct IP address as outlined in the lab guides, the screen shots were captured using a different IP address on the ASA.

Postman

New Import Runner

My Workspace

No Environment

History Collections

GET https://10.1.2.102/api/objects/networkobjects

Params Send Save

Authorization Headers Body Pre-request Script Tests

Key Value Description

New key Value Description

Response

Hit the Send button to get a response.

Do more with requests

Share Mock Monitor Document

Next we need to figure out the Authorization; click on the Authorization section.

Postman

New Import Runner

My Workspace

No Environment

History Collections

GET https://10.1.2.102/api/objects/networkobjects

Params Send Save

Authorization Headers Body Pre-request Script Tests

TYPE

Inherit auth from parent

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

This request is not inheriting any authorization helper at the moment. Save it in a collection to use the parent's authorization helper.

Response

Hit the Send button to get a response.

Do more with requests

Share Mock Monitor Document

From the "Type" drop down menu select "Basic Auth".

The screenshot shows the Postman application interface. On the left is a sidebar with various icons. The main area has tabs for 'New', 'Import', 'Runner', and 'My Workspace'. A search bar labeled 'Filter' is present. Below it, there's a 'History' section stating 'Nothing in your history yet. Requests that you send through Postman are automatically saved here.' To the right, a request is defined: 'GET https://10.1.2.102/api/objects/networkobjects'. The 'Authorization' tab is selected, showing a dropdown menu titled 'TYPE' with options: 'Inherit auth from parent', 'No Auth', 'Bearer Token', 'Basic Auth' (which is highlighted with a red box), 'Digest Auth', 'OAuth 1.0', 'OAuth 2.0', 'Hawk Authentication', 'AWS Signature', and 'NTLM Authentication [Beta]'. A note to the right says: 'This request is not inheriting any authorization helper at the moment. Save it in a collection to use the parent's authorization helper.' Below the dropdown is a button 'Hit the Send button to get a response.' At the bottom are buttons for 'Share', 'Mock', 'Monitor', and 'Document'.

This provides a handy username and password field to the right -- enter in the credentials for your ASA (Username = "ignw", Password = "ignw").

The screenshot shows the Postman application interface. On the left is a sidebar with various icons for different tools and environments. The main workspace has a header bar with 'Postman' and various status indicators like 'IN SYNC'. Below the header, there's a search bar and tabs for 'History' and 'Collections'. A message in the history section says 'Nothing in your history yet. Requests that you send through Postman are automatically saved here.' The main content area shows a 'GET' request to the URL 'https://10.1.2.102/api/objects/networkobjects'. The 'Authorization' tab is active, displaying 'Basic Auth' settings. A note in the authorization section cautions about sensitive data and recommends using variables. The 'Send' button is at the top right, and a 'Preview Request' button is below it. The 'Response' section is currently empty, with a placeholder message 'Hit the Send button to get a response.' At the bottom are buttons for 'Share', 'Mock', 'Monitor', and 'Document'.

The last piece of information we got from the documentation was "content-type" this is a standard HTTP header field, Postman is smart enough to take care of this for us, but for sake of thoroughness let's configure it manually.

Click on the "Headers" tab. In the "key" column, under Authorization, enter a new key of "Content-Type" with a value of "application/json" just as the documentation identified.

The screenshot shows the Postman application interface. On the left is a sidebar with various icons for different tools and environments. The main workspace is titled "My Workspace". A tab for "New Tab" is open, showing a GET request to the URL <https://10.1.2.102/api/objects/networkobjects>. The "Headers" tab is selected, displaying one header entry: "Content-Type" with the value "application/json". The "Send" button is highlighted with a blue border. Below the request area, there's a message saying "Hit the Send button to get a response." and a "Do more with requests" section with "Share", "Mock", "Monitor", and "Document" buttons.

That covers all of the items that we found in the documentation, click the "Send" button to see what happens.

This screenshot is identical to the one above, but the "Send" button is now highlighted with a blue border, indicating it has been clicked. The rest of the interface remains the same, showing the request details and the "Do more with requests" options.

If all went well you should have a response very similar to that shown below. While kind of uneventful, this is a great first step! Note the "Status" field on the top right of the response section -- 200 is the HTTP status code for "OK" ([Helpful HTTP Status Code Reminder!](#)), if you see anything other than 200 there at this point something went wrong.

The screenshot shows the Postman application interface. On the left is a sidebar with various icons for different tools and environments. The main area has a header bar with tabs for 'New', 'Import', 'Runner', and 'My Workspace'. Below the header is a 'History' section with a 'Clear all' button. A 'Collections' section is also present. The central workspace shows a 'GET' request to 'https://10.1.2.102/api/objects/networkobjects'. The 'Headers' tab is selected, showing two entries: 'Authorization' with value 'Basic aWdudzpdXBlcnNlY3VyZQ==' and 'Content-Type' with value 'application/json'. The 'Body' tab is selected, displaying a JSON response with a 'Pretty' view. The JSON output is:

```
1 [ { 2   "selfLink": "https://10.1.2.102/api/objects/networkobjects", 3   "rangeInfo": { 4     "offset": 0, 5     "limit": 0, 6     "total": 0 7   }, 8   "items": [] 9 } ]
```

The status bar at the bottom indicates 'Status: 200 OK', 'Time: 13580 ms', and 'Size: 357 B'.

If you take a peak at the "Headers" section once more you'll notice that there is now a field there for "Authorization" -- take a look at the "value" for this field. That doesn't look anything like our username and password!! What's that all about? Recall that HTTP Basic Authentication is encoded in base64 -- Postman knows this, and auto-magically encoded our credentials for us. Pretty nice of Postman!

The screenshot shows the Postman application interface. On the left is a sidebar with various icons for different tools and collections. The main workspace has a header bar with 'Postman' and various status indicators like 'IN SYNC'. Below the header is a search bar and tabs for 'History' and 'Collections'. A 'Clear all' button is visible. The central area shows a 'New Tab' button, a 'GET' method selected, and the URL 'https://10.1.2.102/api/objects/networkobjects'. To the right of the URL are buttons for 'Params', 'Send', and 'Save'. The 'Send' button is highlighted in blue. Below the URL, the 'Headers' tab is selected, showing two entries: 'Authorization' with value 'Basic aWdudzpdXBcnNlY3VyZQ==', and 'Content-Type' with value 'application/json'. There is also a 'New key' row for adding more headers. Below the headers, there are tabs for 'Body', 'Cookies', 'Headers (6)', and 'Test Results'. The 'Body' tab is selected, showing a JSON response with a status of '200 OK', time '13580 ms', and size '357 B'. The JSON response is displayed in a code editor-like view:

```
1 [ {  
2   "selfLink": "https://10.1.2.102/api/objects/networkobjects",  
3   "rangeInfo": {  
4     "offset": 0,  
5     "limit": 0,  
6     "total": 0  
7   },  
8   "items": []  
9 } ]
```

One last task before we move on -- we need to save our request. Click the small drop down icon next to the "Save" button on the top right of the work pane, then click "Save As..."

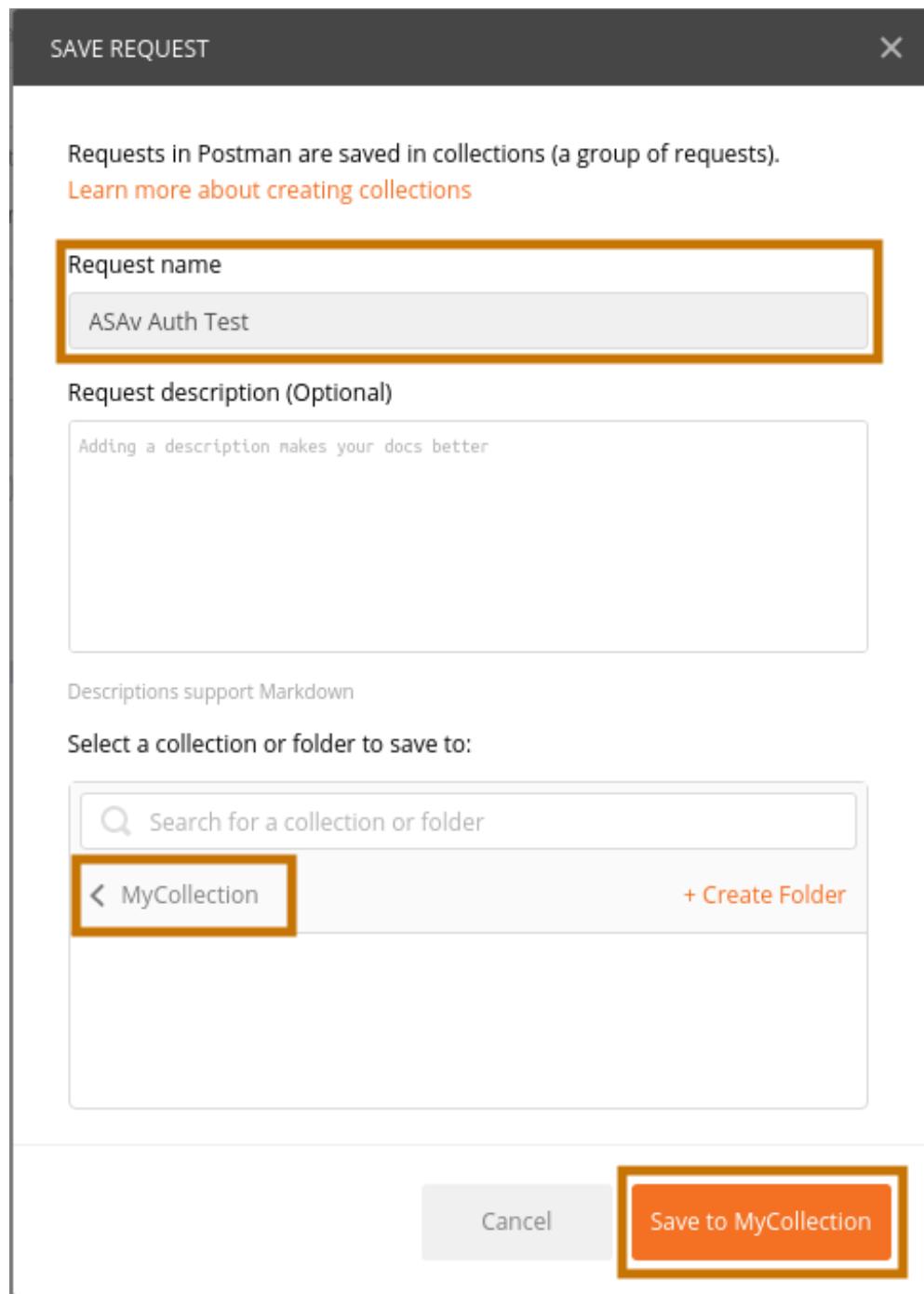
The screenshot shows the Postman application interface. On the left is a sidebar with various icons for different tools and collections. The main area has a header bar with 'Postman' and various status indicators. Below the header is a navigation bar with 'New', 'Import', 'Runner', and a 'My Workspace' dropdown set to 'My Workspace'. To the right of the navigation is a toolbar with 'Send', 'Save', and other options. The main workspace shows a 'History' tab selected, displaying a recent request. The request details are as follows:

- Method:** GET
- URL:** <https://10.1.2.102/api/objects/networkobjects>
- Headers:** (2)
  - Authorization: Basic aWdudzpzdBlcnNIY3VQ==
  - Content-Type: application/json
- Body:** (Pretty, Raw, Preview, JSON)
  - Pretty JSON output:

```
1 [{}  
2   "selfLink": "https://10.1.2.102/api/objects/networkobjects",  
3   "rangeInfo": {  
4     "offset": 0,  
5     "limit": 0,  
6     "total": 0  
7   },  
8   "items": []  
9 }
```
- Test Results:** Status: 200 OK, Time: 13580 ms, Size: 357 B

Enter a request name ("ASAAuth Test" for example), and click the "+ Create Collection" button. A Collection is exactly what it sounds like, a collection of requests -- kind of like a folder where we can store things in. Create a collection to store all the requests for the subsequent Postman tasks. Once you create your Collection click on the collection to store the request in the folder.

Finally, click "Save".



You should now see a folder in the navigation pane, and your request saved under the folder.

The screenshot shows the Postman application interface. On the left is a sidebar with various icons for different tools and environments. The main area shows a 'Collections' tab selected, displaying a list of collections. One collection, 'MyCollection', is highlighted with an orange border. Inside 'MyCollection', there is one request named 'ASAv Auth Test'. The central workspace shows a 'GET' request to 'https://10.1.2.102/api/objects/networkobjects'. The 'Headers' tab is active, showing two headers: 'Authorization' (set to 'Basic aWdudzpdXBlcnNlY3VzZQ==') and 'Content-Type' (set to 'application/json'). Below the headers, the 'Body' tab is selected, showing a JSON response. The response is a single object with a 'selfLink' key pointing to the API endpoint, a 'rangeIndex' object with 'offset', 'limit', and 'total' fields all set to 0, and an 'items' array which is currently empty. At the bottom right of the workspace, there are buttons for 'Pretty', 'Raw', 'Preview', and 'JSON' (which is currently selected), along with a 'Save Response' button.

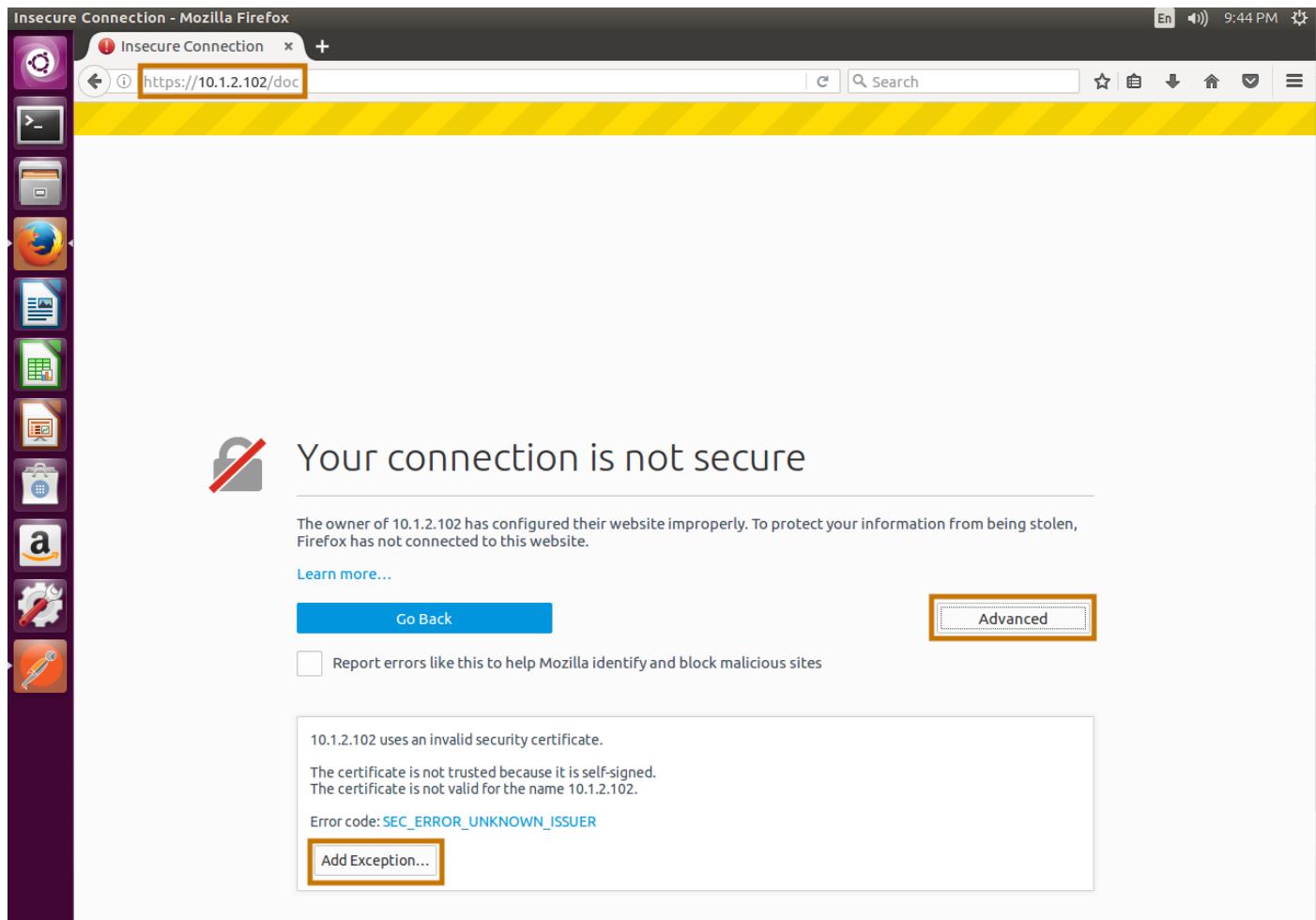
## Learning how to Navigate the API

Now that we know we can get authenticated to our ASAv, we probably need to figure out what we can actually do with it! Lucky for us the ASA itself contains the API documentation! Open Firefox on your student Desktop and navigate to the following URL or click [here](#):

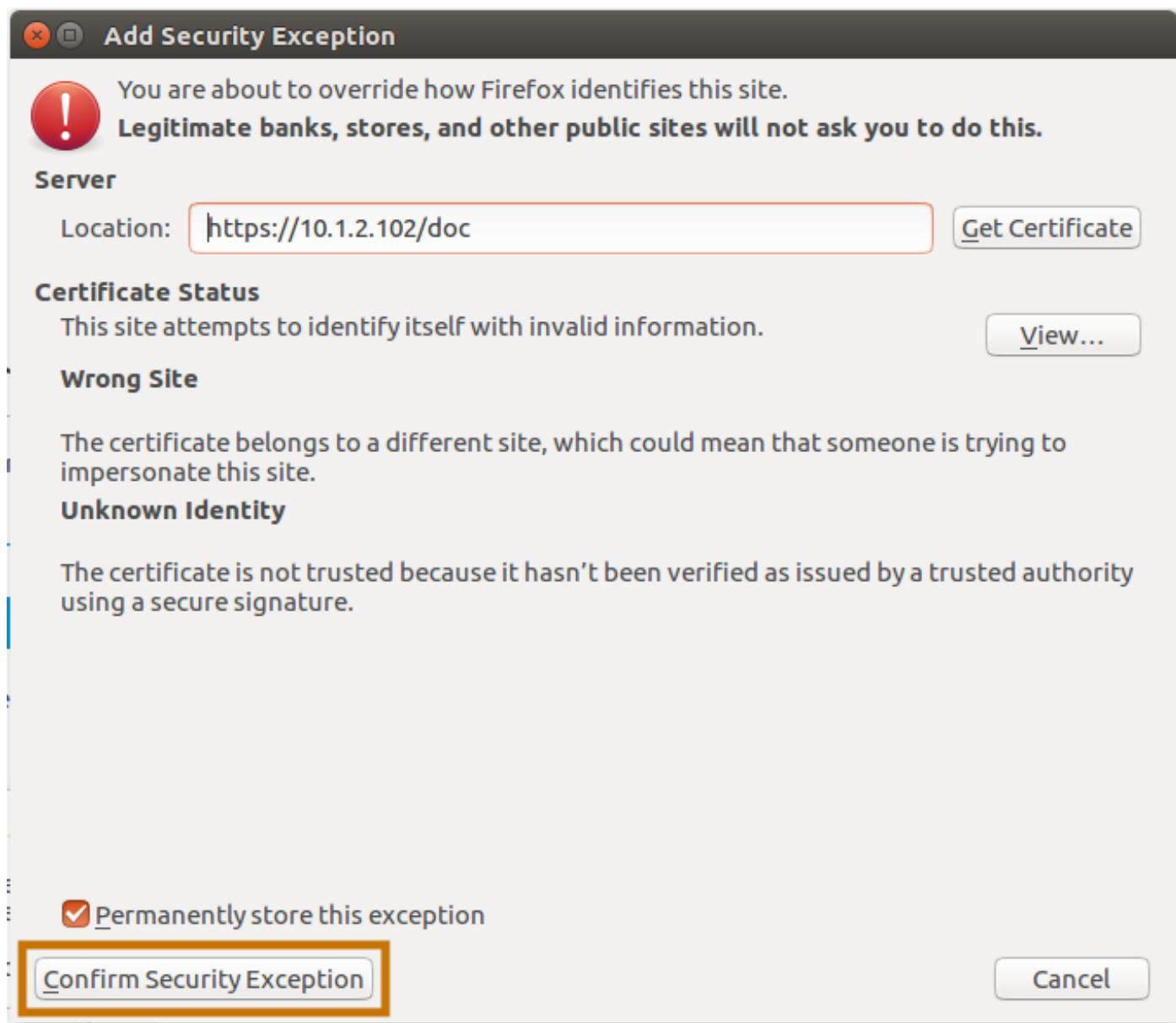
```
https://10.0.0.8/doc/
```

**Note:** Make sure you include the trailing "/"!

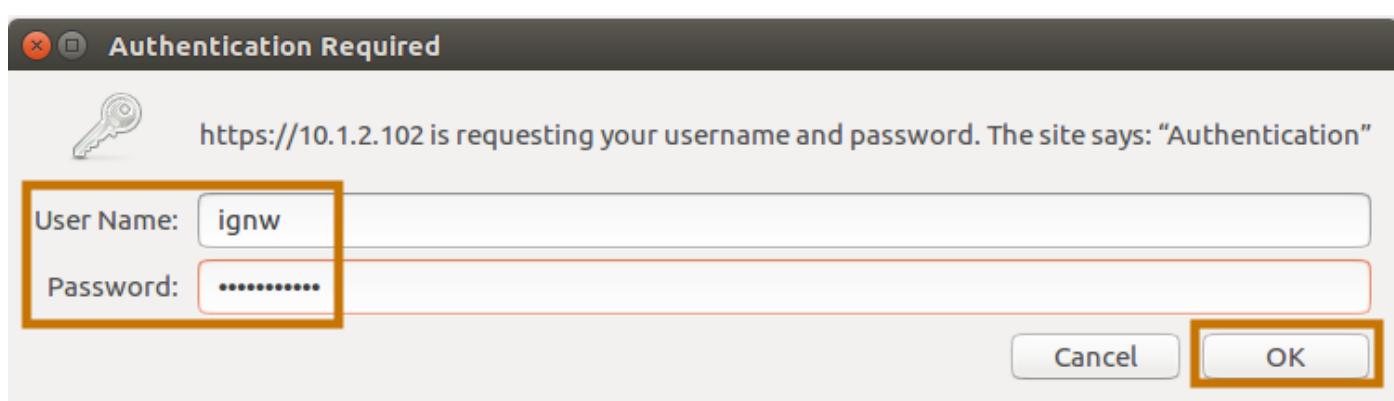
Firefox will prompt you that the site is not secure (again due to the self-signed certificate). Click "Advanced" button, and then the "Add Exception..." button to ignore the self signed certificate.



In the "Add Security Exception" pop-up box, click "Confirm Security Exception".



Enter your username and password at the authentication prompt, then click "OK".



Eventually, you'll make it to the ASA REST API Documentation & Console page!

The screenshot shows the ASA REST API Documentation & Console interface. The left sidebar contains a navigation tree with various icons and sections: API INFO (ASA Version: 9.9(1)2), AAA, Access, Bulk, CLI, Certificate, Context Management, DHCP Server/Relay, DNS Client/Dynamic DNS, Device Setup, Failover, Firewall, Full Backup, Full Restore, Hardware bypass, Interfaces, Interfaces (System), Licensing, Logging, and Management access. The main content area has a blue header "ASA REST API Documentation & Console". Below the header, there are two tabs: "Services" and "Methods". A message says "Select a feature from the left-hand panel to view its APIs." To the right is the "API CONSOLE" section, which includes tabs for "Response Text", "Response Info" (which is selected), and "Request Info". A large text area labeled "Response Information" is currently empty. At the bottom right of the console area is a button "Export operation in...".

This is the ASA REST API Documentation and & Console -- we can use it to learn what API "endpoints" we can query or use for configuration purposes. An "endpoint" is basically a URL (URI if you want to be pedantic!) that represents an object class or type in a device. Take a look at the navigation pane on the left side of the screen -- if you're familiar with the Cisco ASA platform or firewalls or network devices in general, many of the sections likely make sense -- Interfaces, Licensing, Logging, NAT, Routing, etc..

Select the "Interfaces" option. This will display all of the types of interfaces and the URI for each type. It also displays the HTTP methods that are available for each endpoint.

**ASA REST API Documentation & Console**

**API INFO**  
ASA Version: 9.9(1)2

- AAA
- Access
- Bulk
- CLI
- Certificate
- Context Management
- DHCP Server/Relay
- DNS Client/Dynamic DNS
- Device Setup
- Failover
- Firewall
- Full Backup
- Full Restore
- Hardware bypass
- Interfaces**
- Interfaces (System)
- Licensing
- Logging
- Management access
- Monitoring

### Interfaces Services

Interface configuration

<a href="#"><b>/api/interfaces/bvi</b></a>	GET DELETE POST PATCH PUT
API operations on Bridge-group Virtual Interfaces (BVLs).	
<a href="#"><b>/api/interfaces/ethernet</b></a>	GET DELETE POST PATCH PUT
API operations on logical Ethernet interfaces.	
<a href="#"><b>/api/interfaces/firepower/93xx/portchannel</b></a>	GET DELETE POST PATCH PUT
API operations on port-channel interfaces for firepower 93xx devices.	
<a href="#"><b>/api/interfaces/physical</b></a>	GET PATCH PUT
API operations on individual physical interface objects.	
<a href="#"><b>/api/interfaces/portchannel</b></a>	GET DELETE POST PATCH PUT
API operations on port-channel interfaces.	
<a href="#"><b>/api/interfaces/redundant</b></a>	GET DELETE POST PATCH PUT
API operations on logical "redundant" interfaces.	
<a href="#"><b>/api/interfaces/setup</b></a>	PATCH PUT GET
API operations for global interface setup.	
<a href="#"><b>/api/interfaces/vlan</b></a>	GET DELETE POST PATCH PUT
API operations on VLAN interfaces.	

API CONSOLE

Response Text    Response Info    Request Info

Response Information

Export operation in.. ▾

Click on the "GET" option on the "physical" interfaces section.

The screenshot shows the ASA REST API Documentation & Console. On the left, a sidebar lists various API categories like AAA, Access, Bulk, CLI, Certificate, Context Management, DHCP Server/Relay, DNS Client/Dynamic DNS, Device Setup, Failover, Firewall, Full Backup, Full Restore, Hardware bypass, Interfaces (selected), Interfaces (System), Licensing, Logging, Management access, Monitoring, NAT, Objects, Routing, Save, Service policy, Token Services, and VPN.

The main content area displays the details for the `/api/interfaces/physical` endpoint. It includes:

- Implementation Notes:** Fetch a physical interface definition.
- Parameters:**

Parameter	Required	Description	Type	Data Type
<code>objectId</code>	false	Fetch a physical interface definition using its object ID (hardware port ID).	path	string
<code>limit</code>	false	Number of items to return; maximum of 100	query	integer
<code>offset</code>	false	Index of first item to return	query	integer
- Response:**
  - Response Content Type: application/json
  - Response Object: `MgmtInterface`
- MgmtInterface Model:**

Field	Value	Description	Constraints
<code>standByMacAddress</code>	string	Standby MAC Address.	None
<code>kind</code> *	string	The kind of this resource object.	None
<code>name</code>	string	To provide a name for an interface.	None
<code>objectId</code>	string	Unique ID of this resource object	None
<code>securityLevel</code>	int...	The interface security level: enter a value between 0 (lowest) and 100 (highest).	None
<code>activeMacAddress</code>	string	Active MAC Address.	None

The right side features an **API CONSOLE** section for the `/api/interfaces/physical` endpoint. It includes a text input for `objectId`, a note about fetching by object ID, a `+ query parameter` link, a `GET` button, and tabs for Response Text, Response Info, Request Info, and an `Export operation in..` button.

This shows us more detail about the endpoint. The "Parameters" section shows us optional parameters that can help us build out queries; in this case, we have two query options -- limit and offset, and a path option. We'll play with these in the next task.

This documentation also shows us information about the response data that we should expect to receive from the device. In this case the "MgmtInterface" model defines all fields that the ASA would send in a response to this URI.

On the right hand side of the screen there is an "API Console" section. This section allows us to test query functionality right from the browser. Notice that the URI is already selected for you based on which sections you've clicked on. Click the "GET" button to see what happens:

**ASA REST API Documentation & Console**

**API INFO**  
 ASA Version: 9.9(1)2

**/api/interfaces/physical**
GET PATCH PUT

API operations on individual physical interface objects.

**Implementation Notes**  
 Fetch a physical interface definition.

**Parameters**

Parameter	Required	Description	Type	Data Type
objectId	false	Fetch a physical interface definition using its object ID (hardware port ID).	path	string
limit	false	Number of items to return; maximum of 100	query	integer
offset	false	Index of first item to return	query	integer

**Response**

Response Content Type application/json

Response Object MgmtInterface

**MgmtInterface Model**

Field	Value	Description	Constraints
standByMacAddress	string	Standby MAC Address.	None
kind *	string	The kind of this resource object.	None
name	string	To provide a name for an interface.	None
objectId	string	Unique ID of this resource object	None
securityLevel	int...	The interface security level: enter a value between 0 (lowest) and 100 (highest).	None
activeMacAddress	string	Active MAC Address.	None

**API CONSOLE**

**/api/interfaces/physical**

**objectId**

Fetch a physical interface definition using its object ID (hardware port ID).

+ query parameter

GET Success!

Response Text
Response Info
Request Info

```
{
  "kind": "collection#MgmtInterface",
  "selfLink": "https://10.1.2.102/api/interfaces/physical",
  "rangeInfo": {
    "offset": 0,
    "limit": 1,
    "total": 1
  },
  "items": [
    {
      "name": "GigabitEthernet0/0/0"
    }
  ]
}
```

[Export operation in.. ▾](#)

Click on the "Request Info" tab to see what the request you sent looks like -- pretty simple huh!

ASA REST API Documentation & Console

**API INFO**  
ASA Version: 9.9(1)2

**/api/interfaces/physical** GET PATCH PUT

API operations on individual physical interface objects.

**Implementation Notes** Examples  
Fetch a physical interface definition.

**Parameters**

Parameter	Required	Description	Type	Data Type
objectId	false	Fetch a physical interface definition using its object ID (hardware port ID).	path	string
limit	false	Number of items to return; maximum of 100	query	integer
offset	false	Index of first item to return	query	integer

**Response**

Response Content Type application/json

Response Object MgmtInterface

**MgmtInterface Model**

**API CONSOLE**

**/api/interfaces/physical**

objectId

Fetch a physical interface definition using its object ID (hardware port ID).  
+ query parameter

GET Success!

**Response Text** **Response Info** Request Info

```
{"url": "/api/interfaces/physical"}
```

Export operation in.. ▾

Recall the previous task and the URL you used to log into the ASA, this URI is not very different from before!

Click on the "Response Info" tab. This provides some of the same data that we saw earlier with Postman. The most important piece of data here is again the HTTP status code -- 200 in this case. We also see that the content-type matches the "application/json" content-type we used before.

Finally, click on the "Response Text" section. You can expand the window to make it a little easier to see.

ASA REST API Documentation & Console

**API INFO**  
ASA Version: 9.9(1)2

**/api/interfaces/physical** GET PATCH PUT

API operations on individual physical interface objects.

**Implementation Notes** Examples  
Fetch a physical interface definition.

**Parameters**

Parameter	Required	Description	Type	Data Type
objectId	false	Fetch a physical interface definition using its object ID (hardware port ID).	path	string
limit	false	Number of items to return; maximum of 100	query	integer
offset	false	Index of first item to return	query	integer

**Response**

Response Content Type application/json

Response Object MgmtInterface

**MgmtInterface Model**

**API CONSOLE**

**/api/interfaces/physical**

objectId

Fetch a physical interface definition using its object ID (hardware port ID).  
+ query parameter

GET Success!

**Response Text** Response Info **Request Info**

```
Status Code: 200
Content-Type: application/json; charset=UTF-8
Content-Length: 1257
Server: CiscoASARestApiServer
Accept-Ranges: bytes
Vary: Accept-Charset, Accept-Encoding, Accept-Language, Accept
Date: Mon, 19 Mar 2018 16:40:53 GMT
```

Export operation in.. ▾

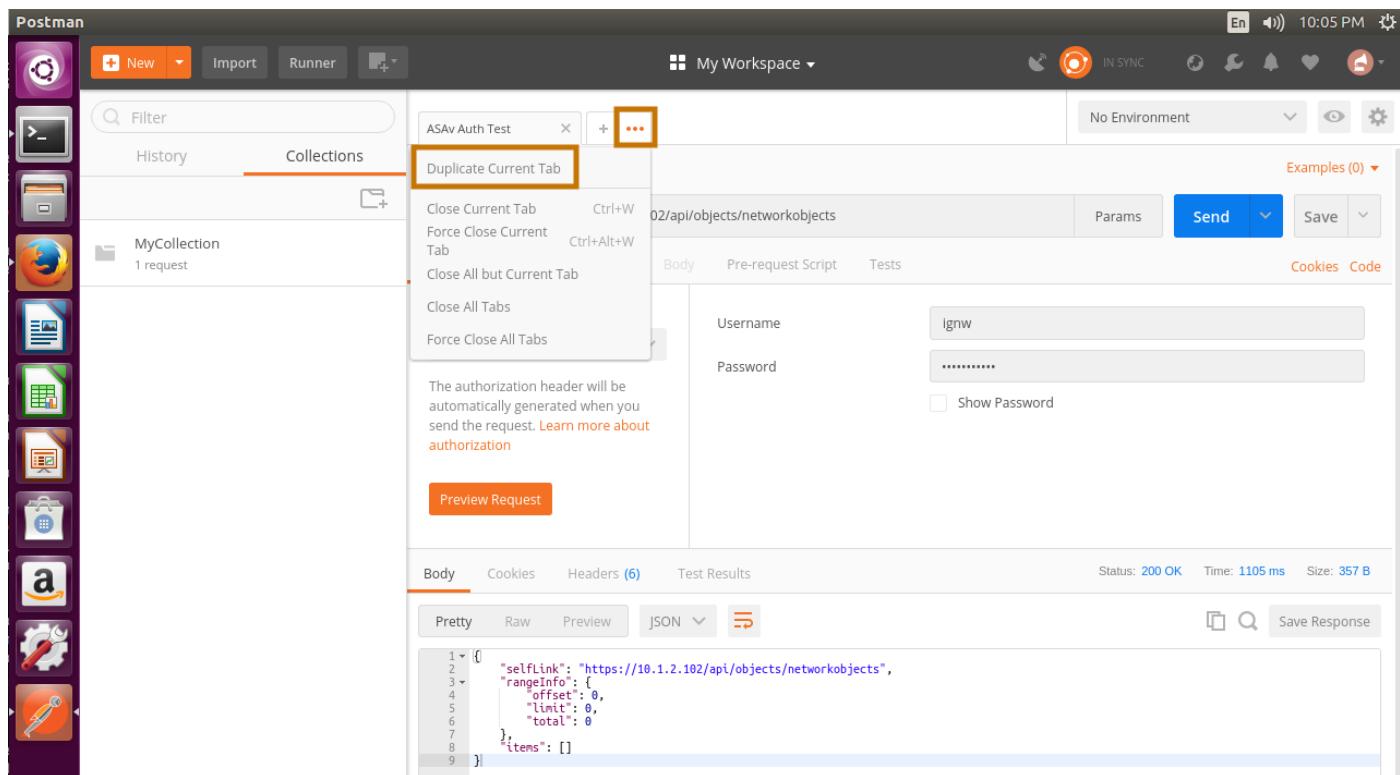
There is a lot of data to unpack here that we'll cover in the following section, but for now simply note that the ASA has replied to our request with a large amount of JSON data describing the endpoint we queried. If you scroll through this JSON blob you'll notice that much of the data corresponds to values displayed in the "MgmtInterface Model".

Every platform will of course have its own documentation that will invariably differ from this, however these are some of key components that you would need to look for -- what are the endpoints (URIs), what data can you send to them, how can you filter your queries, and what is the overall structure of your request and response.

## GETting some (useful?) Data

Now that we've gone through a bit of the documentation we should be able to use Postman to "GET" some data from our ASA.

Create a new request in Postman by clicking the "..." next to your first request, and clicking "Duplicate Current Tab".



The screenshot shows the Postman application interface. On the left is a sidebar with various icons. In the center, there's a list of collections: 'History' and 'MyCollection'. A red arrow points from the 'Collections' tab to a context menu that is open over the 'ASAv Auth Test' collection. The menu is titled 'Duplicate Current Tab' and includes options like 'Close Current Tab', 'Force Close Current Tab', 'Close All but Current Tab', 'Close All Tabs', and 'Force Close All Tabs'. Below the menu, a note says: 'The authorization header will be automatically generated when you send the request. Learn more about authorization'. At the bottom of the interface, there's a preview of the JSON response, which is partially visible.

Mouse near the name of your new request, click the pencil icon to rename it. Name your new request "Get Interface Information".

The screenshot shows the Postman application interface. On the left is a sidebar with various icons for tools like terminal, browser, and file management. The main area is titled "My Workspace" and contains a collection named "ASAv Auth Test". Inside this collection, there is a single request labeled "ASAv Auth Test". The request is set to "GET" and points to the URL "https://10.1.2.102/api/objects/networkobjects". The "Authorization" tab is selected, showing "Basic Auth" configured with "Username: ignw" and "Password: .....". There is a "Preview Request" button and a note about automatic header generation. Below the request, a large button says "Hit the Send button to get a response.".

Let's try to get information about interfaces just like we did in the browser interface. You'll need to change the URL to reflect the endpoint for the interfaces.

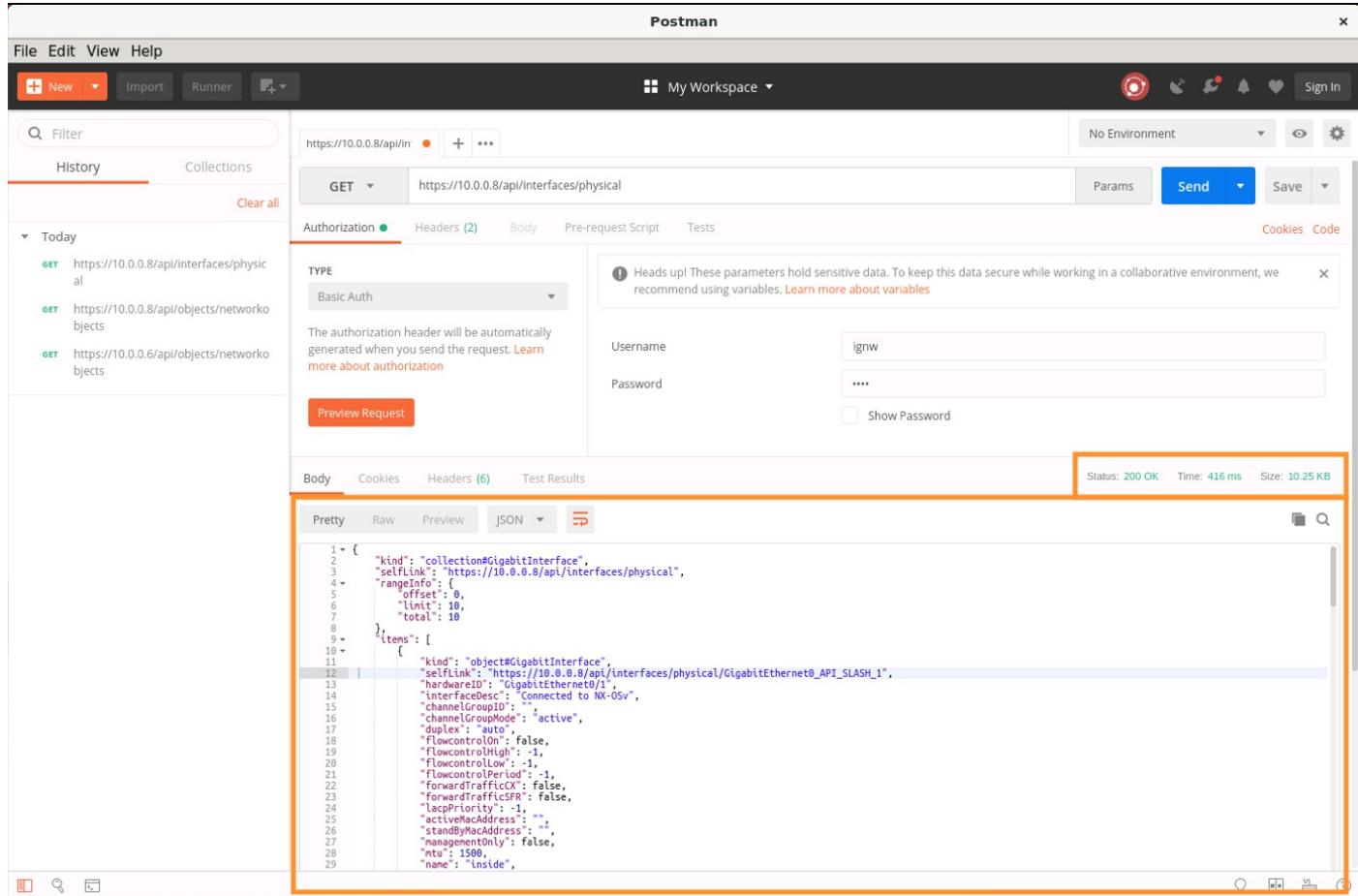
This screenshot shows the Postman interface again. A new collection named "Get Interface Information" has been created. Inside it, there is a request for "Get Interface Information" with the URL "https://10.1.2.102/api/interfaces/physical". The "Authorization" tab is selected, showing "Basic Auth" with "Username: ignw" and "Password: .....". A note about automatic header generation is present. A "Preview Request" button is available, and a message at the bottom says "Hit the Send button to get a response.".

At this point, that should be all you need to do since you copied the original request that was already setup for

the authentication and content-type.

Click "Send" to see if your request is successful. What status code did you get back? If you got a "200 OK" then you are good to go!

You should also have received a payload back from the ASA that Postman has been nice enough to "pretty print" for you as shown below:



The screenshot shows the Postman application interface. In the top navigation bar, 'File Edit View Help' are visible, along with 'New', 'Import', 'Runner', and 'My Workspace'. The workspace shows a single request: 'GET https://10.0.0.8/api/interfaces/physical'. The 'Authorization' tab is selected, showing 'Basic Auth' with fields for 'Username' (ignw) and 'Password' (omitted). The 'Body' tab is selected, displaying the JSON response. The JSON output is as follows:

```
1 {  
2   "kind": "collection#GigabitInterface",  
3   "selfLink": "https://10.0.0.8/api/interfaces/physical",  
4   "rangeInfo": {  
5     "offset": 0,  
6     "limit": 10,  
7     "total": 10  
9   },  
10  "items": [  
11    {  
12      "kind": "object#GigabitInterface",  
13      "selfLink": "https://10.0.0.8/api/interfaces/physical/GigabitEthernet0_API_SLASH_1",  
14      "hardwareID": "GigabitEthernet0/1",  
15      "interfaceDesc": "Connected to NX-OSv",  
16      "channelGroupId": "",  
17      "channelGroupMode": "active",  
18      "duplex": "full",  
19      "flowcontrolOn": false,  
20      "flowcontrolHigh": -1,  
21      "flowcontrolLow": -1,  
22      "flowcontrolPeriod": -1,  
23      "forwardTrafficCC": false,  
24      "frameRelay": false,  
25      "laciPriority": 1,  
26      "activeMacAddress": "",  
27      "standbyMacAddress": "",  
28      "managementOnly": false,  
29      "mtu": 1500,  
      "name": "Inside",  
    }  
  ]  
}
```

The status bar at the bottom right indicates 'Status: 200 OK', 'Time: 416 ms', and 'Size: 10.25 KB'.

This data should look awfully similar to the data you saw in the browser tool. Let's dig into it a bit more now. The JSON payload we received is very much like the dictionaries we worked with in Python (not 100% the same, but pretty close!), so let's take a look at the key/value pairs that we see in the first section of the payload.

```

1  {
2      "kind": "collection#GigabitInterface",
3      "selflink": "https://10.0.0.8/api/interfaces/physical",
4      "rangeInfo": [
5          {
6              "offset": 0,
7              "limit": 10,
8              "total": 10
9          }
10         "items": [
11             {
12                 "kind": "object#GigabitInterface",
13                 "selflink": "https://10.0.0.8/api/interfaces/physical/GigabitEthernet0_API_SLASH_1",
14                 "hardwareID": "GigabitEthernet0/1",
15                 "interfaceDesc": "Connected to NX-OS",
16                 "channelGroupID": ,
17                 "channelGroupMode": "active",
18                 "duplex": "auto",
19                 "flowcontrolOn": false,
20                 "flowcontrolHigh": -1,
21                 "flowcontrolLow": -1,
22                 "flowcontrolPeriod": -1,
23                 "forwardTrafficCX": false,
24                 "forwardTrafficSFR": false,
25                 "lacpPriority": -1,
26                 "activeMacAddress": ,
27                 "standbyMacAddress": ,
28                 "managementOnly": false,
29                 "mtu": 1500,
30                 "name": "inside"
31             }
32         ]
33     }
34 
```

Our first key is "kind", with a value of "MgmtInterface". This is the "Response Object" kind that we saw in the documentation. We obviously know what type of data we were expecting to get since we just crafted this request, but if we were dealing with exported data from the ASA in a flat JSON file this would probably be very good information to have!

The "selfLink" object is clearly the same as the URL we just queried -- again, not super useful since we just built our query out and we of course know this URL, but if this was in a flat file that we were parsing (with Python perhaps!) this would be very handy data that would allow us to re-query the device quickly.

Finally, we have a nested dictionary/hash called "rangeInfo" -- this contains three values that appear to be telling us about the length of the data set that we've received from the ASA.

Below the rangeInfo section is the meat of the response, contained within the "items" section which is a list object (remember, square brackets!). Based on the data from the rangeInfo section, we can make an educated guess that there are three entries. If you SSH directly into the ASA, does that number make sense?

The terminal window shows the following session:

```

ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ ssh 10.0.0.8
ignw@10.0.0.8's password:
User ignw logged in to ignw-asav
Logins over the last 1 days: 2. Last login: 22:19:43 UTC May 23 2018 from 10.10.0.254
Failed logins since the last login: 0.
Type help or '?' for a list of available commands.
ignw-asav> en
Password: ****
ignw-asav# sh int ip brief
Interface          IP-Address      OK? Method Status      Protocol
GigabitEthernet0/0  203.0.113.2    YES CONFIG up        up
GigabitEthernet0/1  10.255.255.1   YES CONFIG up        up
GigabitEthernet0/2  unassigned     YES unset administratively down up
GigabitEthernet0/3  unassigned     YES unset administratively down up
GigabitEthernet0/4  unassigned     YES unset administratively down up
GigabitEthernet0/5  unassigned     YES unset administratively down up
GigabitEthernet0/6  unassigned     YES unset administratively down up
GigabitEthernet0/7  unassigned     YES unset administratively down up
GigabitEthernet0/8  unassigned     YES unset administratively down up
Management0/0       10.0.0.8      YES CONFIG up        up
ignw-asav#

```

The JSON API response in Postman is:

```

1 ▾ {
2   "kind": "collection#GigabitInterface",
3   "selfLink": "https://10.0.0.8/api/interfaces/physical",
4   "rangeInfo": {
5     "offset": 0,
6     "limit": 10,
7     "total": 10
8   },
9   "items": [
10  {
11    "kind": "object#GigabitInterface",
12    "selfLink": "https://10.0.0.8/api/interfaces/physical/GigabitEthernet0_API_SLASH_1",
13    "hardwareID": "GigabitEthernet0/1",
14    "interfaceDesc": "Connected to NX-OSv",
15    "channelGroupID": "",
16    "channelGroupMode": "active",
17    "duplex": "auto",
18    "flowcontrol0": false,
19    "flowcontrolHigh": -1,
20    "flowcontrolLow": -1,
21    "flowcontrolPeriod": -1,
22    "forwardTrafficCX": false,
23    "forwardTrafficSFR": false,
24    "lacpPriority": -1,
25    "activeMacAddress": "",
26    "standByMacAddress": "",
27    "managementOnly": false,
28    "mtu": 1500,
29    "name": "inside",

```

Looking back at the data in Postman, what kind of data did we receive? It looks like we have all sorts of good stuff: mtu, security level, IP address information, interface description, and the list goes on!

Now what if we wanted to filter this data down a bit? Let's say we only want to get data for a single interface, how do you think we could go about doing that? Take a look back in the API documentation for the physical interfaces and see if you can figure out how to get the same output as shown below:

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

**Preview Request**

Body Cookies Headers (6) Test Results Status: 200 OK Time: 776 ms Size: 1.47 KB

Pretty Raw Preview JSON ↗ Save Response

```

1  {
2     "kind": "collection#MgmtInterface",
3     "selflink": "https://10.1.2.102/api/interfaces/physical?limit=1",
4     "rangeInfo": {
5         "offset": 0,
6         "limit": 1,
7         "total": 1
8     },
9     "items": [
10        {
11            "kind": "object#MgmtInterface",
12            "selflink": "https://10.1.2.102/api/interfaces/physical/Management0_API_SLASH_0",
13            "hardwareID": "Management0/0",
14            "interfaceDesc": "Management Int (Don't change me!)",
15            "channelGroupID": null,
16            "channelGroupMode": "active",
17            "duplex": "auto",
18            "flowcontrolOn": false,
19            "flowcontrolHigh": -1,
20            "flowcontrolLow": -1,
21            "flowcontrolPeriod": -1,
22            "forwardTrafficCX": false,
23            "forwardTrafficSFR": false,
24            "lacpPriority": -1,
25            "activeMacAddress": null,
26            "standbyMacAddress": null,
27            "managementOnly": false,
28            "mtu": 1500,
29            "name": "management",
30            "securityLevel": 100,
31            "shutdown": false,
32            "speed": "auto",
33            "ipAddress": {
34                "kind": "StaticIP",
35                "ip": {
36                    "kind": "IPv4Address",
37                    "value": "10.1.2.102"
38                },
39                "netMask": {
40                    "kind": "IPv4NetMask",
41                    "value": "255.255.255.0"
42                }
43            },
44            "ipv6Info": {
45                "enabled": false
46            }
47        }
48    ]
49 }
50 
```

If you're stuck, make sure to pay extra attention to the "Parameters" section in the documentation, you can then use the API Console in the browser, filter your query, and hit the "Request Info" tab to see the URL to use!

Try to query the ASA using Postman to capture the management access information.

## Configuration Time!

It's time to use Postman to configure the ASA! We'll try to configure GigabitEthernet0/8 since it is currently not

in use -- that gives us the flexibility to do whatever we want to it without worrying about breaking anything!

Now would be a good time to open the ASA API Documentation back up. Find the Interface section, and select "physical" interface. What HTTP methods are available for this endpoint? It looks like we have "GET", "PATCH", and "PUT". Now is probably a good time to discuss these a bit further...

## GET

You've already worked with the "GET" method, and as you would expect it is used to request or "GET" data via HTTP. You can check out the Mozilla Developer doc for GET [here](#).

## PUT

The PUT method allows us to "create a new resource" or replace an existing resource. What does this mean to us and our ASA? PUT will be used to configure something initially, or to *replace* an existing configuration. PUT is also idempotent - just like some of the Ansible Playbooks created previously. You can check out the Mozilla Developer doc for PUT [here](#).

## PATCH

Finally, PATCH is similar to PUT except it can be used to make "partial modifications to a resource." In other words -- with PUT we have to send the entire configuration for the object we would like to configure, existing configurations will be replaced with whatever we send, whereas with PATCH we can modify subset(s) of the object we wish to configure. You can check out the Mozilla Developer doc for PATCH [here](#).

## POST

POST is also similar to PUT in that it is used to send data to an endpoint, however it is **not** idempotent. You can check out the Mozilla Developer doc for POST [here](#).

**Note:** The above are definitions from Mozilla, though the exact implementation of each of them methods does vary from API to API -- some may use POST for *everything*. Some use POST to create, and PATCH/PUT to modify some the exact opposite. You'll need to check out the documentation for the API of the device you are working with -- or just try it out to see what you need to do.

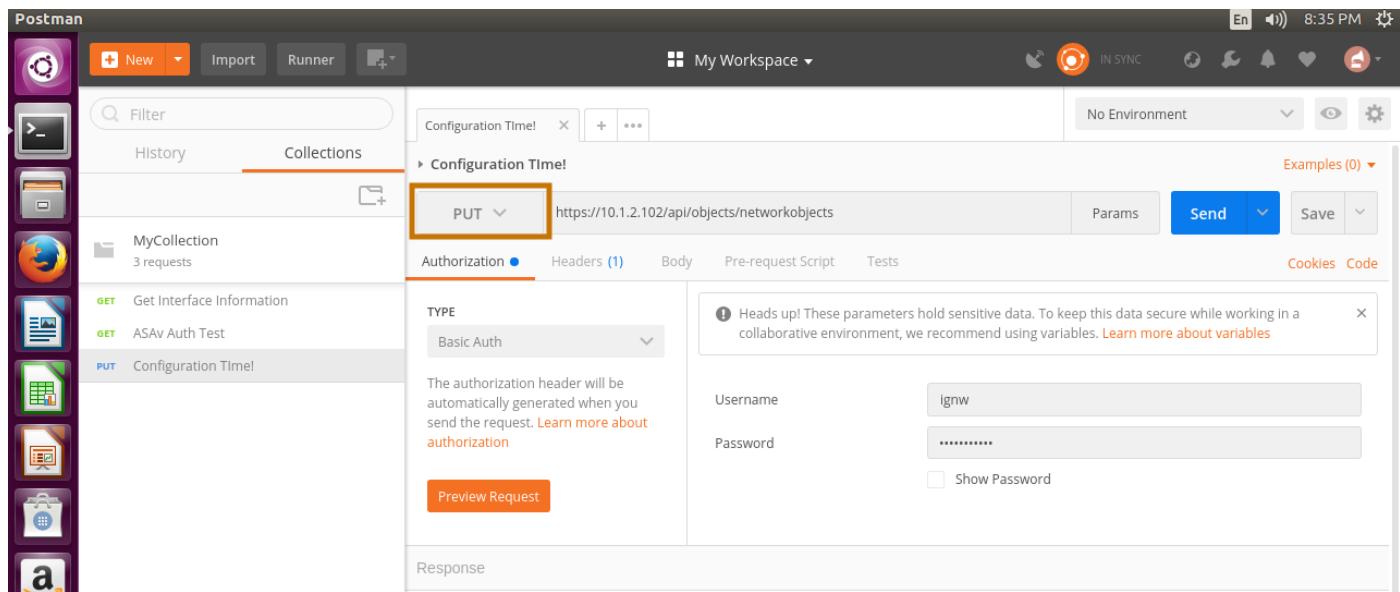
Depending on the API you are working with, you may end up using any combination of these or other methods we haven't covered.

Now that we know a bit about the HTTP methods listed on our physical interface endpoint, which do you suppose we should use? We're probably OK using the PUT method, but we can certainly try both PATCH and PUT, and we already know we don't need to use GET since that of course won't configure anything for us!

Before we get too far, let's duplicate our last Postman request, and create and save a new one for this configuration task.

The screenshot shows the Postman application interface. On the left is a sidebar with various icons for different tools and environments. The main workspace is titled "My Workspace" and shows a collection named "MyCollection" which contains three requests: "Get Interface Information", "ASAv Auth Test", and "Configuration Time!". The "Configuration Time!" request is currently selected, highlighted with a yellow box. The request details pane shows the method as "GET", the URL as "https://10.1.2.102/api/objects/networkobjects", and the "Authorization" tab selected. The "Basic Auth" type is chosen, and the "Username" field is filled with "ignw". The "Send" button is visible at the top right of the request details pane. Below the request details, there's a "Response" section with a placeholder message "Hit the Send button to get a response." and a "Do more with requests" section containing "Share", "Mock", "Monitor", and "Document" buttons.

Change the HTTP method to "PUT".



Now we need to know "where" to PUT this -- what endpoint/URL do we send our configuration to? We know from before that we can use the "api/interfaces/physical" URI for at least query, but can we send data to that? In general with a RESTful API the answer is "no" (or perhaps more realistically, "maybe") -- we are trying to configure a specific item and the "api/interfaces/physical" URI is an endpoint for that *class* of item not the specific item (i.e. GigabitEthernet0/8).

Try clicking on the "examples" button in the documentation to see if that gives us any more detail.

ASA REST API - Mozilla Firefox

ASA REST API

https://10.1.2.102/doc/#feature/interfaces/physical/{objectId}\_PUT

ASA REST API Documentation & Console

**API INFO**  
ASA Version: 9.9(1)2

**AAA**  
Access  
Bulk  
CLI  
Certificate  
Context Management  
DHCP Server/Relay  
DNS Client/Dynamic DNS  
Device Setup  
Failover  
Firewall  
Full Backup  
Full Restore  
Hardware bypass

**Interfaces**  
Interfaces (System)  
Licensing  
Logging  
Management access  
Monitoring  
NAT  
Objects  
Routing  
Save  
Service policy  
Token Services

**Implementation Notes**  
Update an existing physical interface definition using the provided parameters.

**Parameters**

Parameter	Required	Description	Type	Data Type
objectId	true	Update the existing physical interface definition specified by the supplied object ID (hardware port ID).	path	string
body	true	Physical interface definition which is to replace the existing interface object.	body	MgmtInterface

**Response**  
Response Content Type application/json

**MgmtInterface Model**

Field	Value	Description	Constraints
-------	-------	-------------	-------------

Physical interface definition which is to replace the existing interface object.

objectID

Update the existing physical interface definition specified by the supplied object ID (hardware port ID).

body

Physical interface definition which is to replace the existing interface object.

PUT Error

Response Text Response Info Request Info

```
{
  "messages": [
    {
      "level": "Error",
      "code": "RESOURCE-NOT-FOUND",
      "details": "RESOURCE-NOT-FOUND"
    }
  ]
}
```

Export operation in..

Choose What I Share

Firefox automatically sends some data to Mozilla so that we can improve your experience.

ASA REST API - Mozilla Firefox

ASA REST API

https://10.1.2.102/doc/#feature/interfaces/physical/{objectId}\_PUT

ASA REST API Documentation & Console

API INFO  
ASA Version: 9.9(1)2

Interfaces Services  
Interface configuration

AAA /api/interfaces/bvi GET DELETE POST PATCH PUT

Access

Bulk

CLI

Certificate

Context Manager

DHCP Server

DNS Client/DNS

Device Setup

Failover

Firewall

Full Backup

Full Restore

Hardware by

Interfaces

Interfaces (S)

Licensing

Logging

Management

Monitoring

NAT

Objects

Routing

Save

Service policies

Token Services

Example Request(s)

PUT a single GigabitEthernet0/0 interface

PUT /api/interfaces/physical/GigabitEthernet0\_API\_SLASH\_0

Request Data

```
{
    "securityLevel": 0,
    "kind": "object#GigabitInterface",
    "channelGroupMode": "active",
    "flowcontrolLow": -1,
    "name": "gig0",
    "duplex": "auto",
    "forwardTrafficSFR": false,
    "hardwareID": "GigabitEthernet0/0",
    "mtu": 1500,
    "lacpPriority": -1,
    "flowcontrolHigh": -1,
    "ipAddress": {
        "ip": {
            "kind": "IPv4Address",
            "value": "192.168.1.2"
        },
        "kind": "StaticIP",
        "netMask": {
            "kind": "IPv4NetMask",
            "value": "255.255.255.0"
        }
    },
    "flowcontrolOn": false,
    "shutdown": true,
    "interfaceDesc": "Add description test",
    "managementOnly": false,
    "channelGroupID": "",
    "speed": "auto",
    "forwardTrafficCX": false,
    "flowcontrolPeriod": -1
}
```

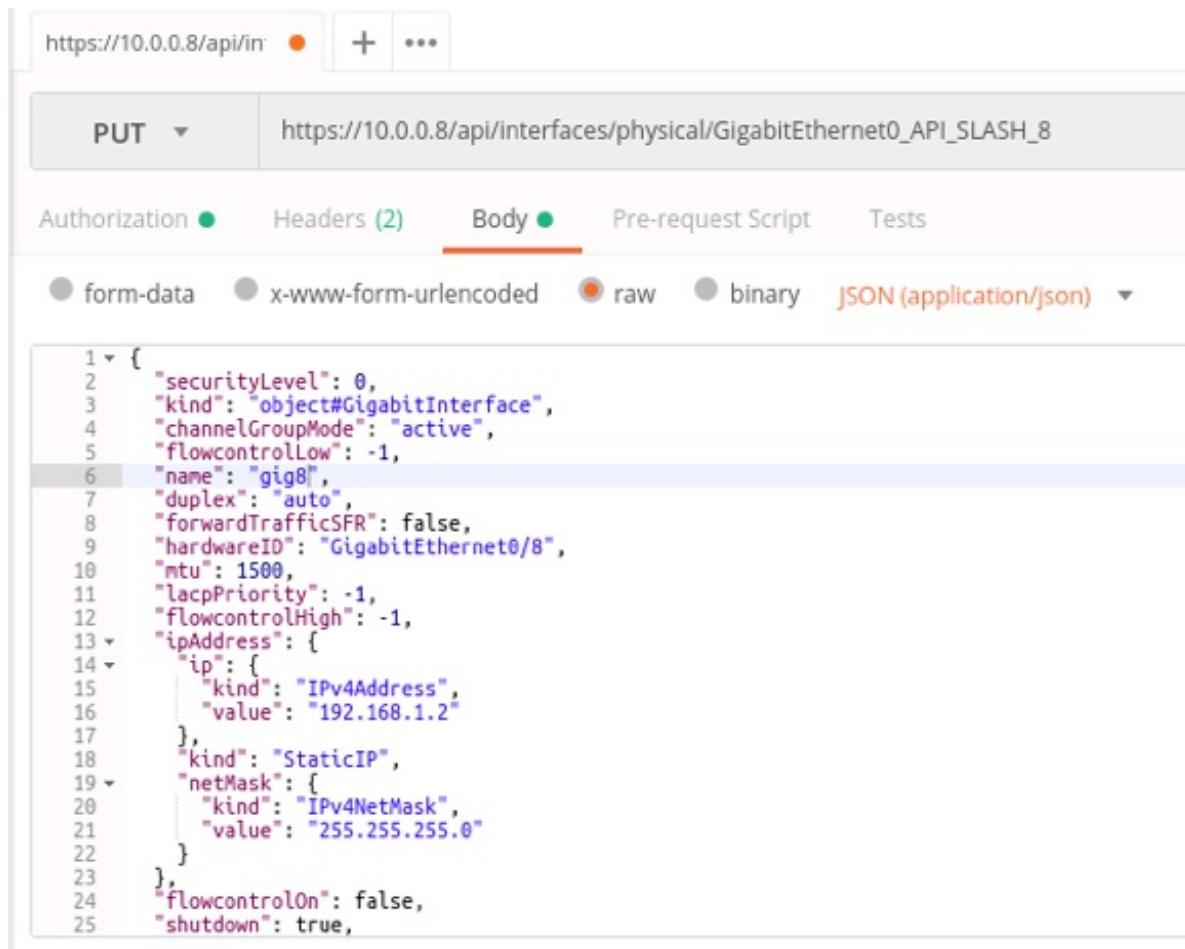
Response Status Code 204

Firefox automatically sends some data to Mozilla so that we can improve your experience.

Ahh! That's more like it! Interesting though that the URI specifically calls out "GigabitEthernet0\_API\_SLASH\_0" -- we will need to change that to "GigabitEthernet0\_API\_SLASH\_8" since we want to modify

GigabitEthernet0/8 not GigabitEthernet0/0. Modify the URL in Postman based on this new information.

Next we need to craft the "body" or the payload of our request. Thankfully the documentation even provides us this! Copy the example data and paste it into the "body" section of Postman (ensure you select the "raw" radio button).



The screenshot shows the Postman interface with a PUT request to the URL `https://10.0.0.8/api/interfaces/physical/GigabitEthernet0_API_SLASH_8`. The 'Body' tab is selected, and the 'raw' radio button is chosen. The JSON payload is as follows:

```
1 ▾ {  
2   "securityLevel": 0,  
3   "kind": "object#GigabitInterface",  
4   "channelGroupMode": "active",  
5   "flowcontrolLow": -1,  
6   "name": "gig8",  
7   "duplex": "auto",  
8   "forwardTrafficSFR": false,  
9   "hardwareID": "GigabitEthernet0/8",  
10  "mtu": 1500,  
11  "lacpPriority": -1,  
12  "flowcontrolHigh": -1,  
13  "ipAddress": {  
14    "ip": {  
15      "kind": "IPv4Address",  
16      "value": "192.168.1.2"  
17    },  
18    "kind": "StaticIP",  
19    "netMask": {  
20      "kind": "IPv4NetMask",  
21      "value": "255.255.255.0"  
22    }  
23  },  
24  "flowcontrolOn": false,  
25  "shutdown": true,
```

Given that the example is for GigabitEthernet0/0, it's a safe bet that we need to modify the payload as well! Look through the example JSON data and see which sections you think you need to change.

It kind of looks like there may be two places: "name", and "hardwareID". In this case we actually only *need* to modify the "hardwareID" -- "name" is referring to the "nameif" (security zone) and we can use whatever we would like for that! Change the "hardwareID" to "GigabitEthernet0/1", and the "name" to whatever you would like (just so it isn't confusingly named gig0!).

If you think your request looks ready to send, click the "Send" button!

The screenshot shows the Postman application interface. In the top navigation bar, there are tabs for File, Edit, View, Help, and a workspace dropdown labeled "My Workspace". Below the navigation is a search bar with a "Filter" button and a "History" tab selected. A sidebar on the left lists recent API calls with their URLs and methods: PUT https://10.0.0.8/api/interfaces/physical/GigabitEthernet0\_API\_SLASH\_8, GET https://10.0.0.8/api/interfaces/physical, GET https://10.0.0.8/api/objects/networkobjects, and GET https://10.0.0.6/api/objects/networkobjects.

The main workspace shows a "PUT" request to "https://10.0.0.8/api/interfaces/physical/GigabitEthernet0\_API\_SLASH\_8". The "Body" tab is active, showing a JSON payload:

```
1 ~ {  
2   "securityLevel": 0,  
3   "kind": "objectGigabitInterface",  
4   "channelGroupMode": "active",  
5   "flowcontrolOn": -1,  
6   "name": "gig0",  
7   "duplex": "auto",  
8   "forwardTrafficSFR": false,  
9   "hardwareID": "GigabitEthernet0/0",  
10  "mtu": 1500,  
11  "linkPriority": -1,  
12  "flowcontrolHigh": -1,  
13  "ipAddress": {  
14    "ip": {  
15      "kind": "IPv4Address",  
16      "value": "192.168.1.2",  
17    },  
18    "kind": "StaticIP",  
19    "netMask": {  
20      "kind": "IPv4NetMask",  
21      "value": "255.255.255.0"  
22    }  
23  },  
24  "flowcontrolOn": false,  
25  "shutdown": true,  
}
```

The "Body" tab also includes "Cookies", "Headers (6)", and "Test Results" sub-tabs. The "Test Results" section displays the response status: "Status: 204 No Content Time: 1548 ms Size: 281 B".

Well that was uneventful! Did nothing happen? What status code did you receive? You should have received a "204 No Content" as shown in the screen shot. Any 2xx status codes are *almost always* a good sign. Many APIs will provide a "200 OK" and include a payload that indicates success, but the ASA V simply gives us a status code with no response payload. SSH to your ASA V to see if your request was as successful as we think/hope it was!

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw-asav> en
Password: ****
ignw-asav# sh int ip brie
Interface IP-Address OK? Method Status Prot
ocol
GigabitEthernet0/0 203.0.113.2 YES CONFIG up
GigabitEthernet0/1 10.255.255.1 YES CONFIG up
GigabitEthernet0/2 unassigned YES unset administratively down up
GigabitEthernet0/3 unassigned YES unset administratively down up
GigabitEthernet0/4 unassigned YES unset administratively down up
GigabitEthernet0/5 unassigned YES unset administratively down up
GigabitEthernet0/6 unassigned YES unset administratively down up
GigabitEthernet0/7 unassigned YES unset administratively down up
GigabitEthernet0/8 192.168.1.2 YES manual administratively down up
Management0/0 10.0.0.8 YES CONFIG up
ignw-asav# sh run int g0/8
!
interface GigabitEthernet0/8
description Add description test
shutdown
nameif newsecurityzone
security-level 0
ip address 192.168.1.2 255.255.255.0
ignw-asav#
```

OK, so we are getting somewhere now! Look through the JSON that you sent to the ASA V and modify the IP address back to "10.1.254.5 255.255.255.0", and the interface description to be "Hey, I did this with Postman!", then re-run your PUT to see what happens.

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
GigabitEthernet0/5      unassigned      YES unset administratively down up
GigabitEthernet0/6      unassigned      YES unset administratively down up
GigabitEthernet0/7      unassigned      YES unset administratively down up
GigabitEthernet0/8      192.168.1.2    YES manual administratively down up
Management0/0           10.0.0.8       YES CONFIG up
ignw-asav# sh run int g0/8
!
interface GigabitEthernet0/8
description Add description test
shutdown
nameif newsecurityzone
security-level 0
ip address 192.168.1.2 255.255.255.0
ignw-asav#
ignw-asav#
ignw-asav# sh run int g0/8
!
interface GigabitEthernet0/8
description Hey, I did this with Postman!
shutdown
nameif newsecurityzone
security-level 0
ip address 192.168.1.2 255.255.255.0
ignw-asav#
```

What if we only wanted to make a minor change -- changing only the description perhaps? Based on what you learned about the different HTTP methods earlier, do you think you could remove all of the configuration options other than description and still have your request work?

Create another duplicate of your PUT request so that you can test it out.

The screenshot shows the Postman application interface. The top navigation bar includes File, Edit, View, Help, New, Import, Runner, and Sign In. The main workspace is titled "My Workspace" and shows a "PUT" request to the URL `https://10.0.0.8/api/interfaces/physical/GigabitEthernet0_API_SLASH_8`. The request body contains the following JSON payload:

```
1 - {  
2   "securityLevel": 0,  
3   "kind": "object#GigabitInterface",  
4   "channelGroupMode": "active",  
5   "flowcontrolLow": -1,  
6   "name": "SecurityZone",  
7   "duplex": "auto",  
8   "forwardTrafficSFR": false,  
9   "hardwareID": "GigabitEthernet0/0",  
10  "lacpPriority": -1,  
11  "flowcontrolHigh": -1,  
12  "ipConfig": [  
13    {  
14      "ip": {  
15        "kind": "IPv4Address",  
16        "value": "192.168.1.2"  
17      },  
18      "kind": "StaticIP",  
19      "netMask": {  
20        "kind": "IPv4NetMask",  
21        "value": "255.255.255.0"  
22      }  
23    },  
24    {"flowcontrolOn": false,  
25    "shutdown": true,  
26    "interfaceDesc": "Hey, I did this with Postman!"  
27  }  
28}
```

The status bar at the bottom indicates Status: 204 No Content, Time: 2137 ms, and Size: 281 B. The "Body" tab is selected, showing Pretty, Raw, Preview, Text, and a Save Response button.

Try to delete some of the interface configurations from the JSON payload -- "mtu" for example -- and run your new request.

The screenshot shows the Postman application interface. In the top navigation bar, there are tabs for File, Edit, View, Help, New, Import, Runner, and a workspace dropdown set to "My Workspace". On the right side of the header are icons for profile, notifications, and sign-in.

The main area displays a "PUT" request to the URL [https://10.0.0.8/api/interfaces/physical/GigabitEthernet0\\_API\\_SLASH\\_8](https://10.0.0.8/api/interfaces/physical/GigabitEthernet0_API_SLASH_8). The request body is a JSON object containing various interface configuration parameters. The response status is 400 Bad Request, with a time of 2428 ms and a size of 391 B. The response body is a JSON object with an "error" message:

```

1 ~ [
  2   "messages": [
  3     {
  4       "level": "Error",
  5       "code": "INTERFACE-MTU-INVALID",
  6       "context": "mtu",
  7       "details": "MTU must be an integer value between 64 and 9000."
  8     }
  9   ]
10 ]

```

Well.... that didn't go according to plan, if we can't remove one configuration component we certainly can't remove all but one configuration component! Why do you think this request failed? Think back to the different HTTP methods and their intended purpose, what do you recall about the PUT method?

The PUT method creates an object, or completely replaces an object... it would be pretty difficult to completely create any object if you don't have all of the required components. Perhaps the "PATCH" method would be better suited to changing only certain attributes of our interface. Change the method for your new request to "PATCH" (leaving the "mtu" field deleted) and try to run your request once more.

The screenshot shows the Postman application interface. At the top, there's a menu bar with File, Edit, View, Help. Below the menu is a toolbar with New, Import, Runner, and other icons. The main area has tabs for History and Collections, with Collections selected. A search bar labeled 'Filter' is present. In the center, there's a configuration section with a 'PATCH' dropdown and a URL field containing 'https://10.0.0.8/api/interfaces/physical/GigabitEthernet0\_API\_SLASH\_8'. To the right of the URL are buttons for Params, Send, and Save. Below the URL is a large text area showing a JSON payload with 25 lines of code. The payload includes fields like 'securityLevel', 'kind', 'channelGroupMode', 'flowcontrolOn', 'shutdown', and 'interfaceDesc'. The 'interfaceDesc' field contains the value 'Hey, I did this with Postman!'. At the bottom of the main window, there are tabs for Body, Cookies, Headers (6), and Test Results. The Body tab is active and shows the JSON payload. The Test Results tab shows a status bar with 'Status: 204 No Content', 'Time: 188 ms', and 'Size: 281 B'. The bottom right corner of the main window has a 'Save Response' button.

Now what if you wanted to change *only* the description on the interface? Take a look at the API documentation page once more to see if anything sticks out to you.

ASA REST API - Mozilla Firefox

ASA REST API

https://10.1.2.102/doc/#feature/interfaces/physical/{objectId}\_PUT

Search

ASA REST API Documentation & Console

parameters.

Parameter	Required	Description	Type	Data Type
objectId	true	Update the existing physical interface definition specified by the supplied object ID (hardware port ID).	path	string
body	true	Physical interface definition which is to replace the existing interface object.	body	MgmtInterface

**Response**

Response Content Type application/json

**MgmtInterface Model**

Field	Value	Description	Constraints
standByMacAddress	string	Standby MAC Address.	None
kind *	string	The kind of this resource object.	None
name	string	To provide a name for an interface.	None
objectId	string	Unique ID of this resource object	None
securityLevel	int...	The interface security level: enter a value between 0 (lowest) and 100 (highest).	None
activeMacAddress	string	Active MAC Address.	None
hardwareID	string	Physical interface ID (hardware ID).	None
forwardTraffic	bool...	Enable traffic forwarding to CX module.	None

objectID

Update the existing physical interface definition specified by the supplied object ID (hardware port ID).

body

Physical interface definition which is to replace the existing interface object.

+ query parameter

PUT Error

Response Text Response Info Request Info

```
{
  "messages": [
    {
      "level": "Error",
      "code": "RESOURCE-NOT-FOUND",
      "details": "RESOURCE-NOT-FOUND"
    }
  ]
}
```

Export operation in.. ▾

Choose What I Share x

Firefox automatically sends some data to Mozilla so that we can improve your experience.



## API INFO

ASA Version: 9.9(1)2

AAA
Access
Bulk
CLI
Certificate
Context Management
DHCP Server/Relay
DNS Client/Dynamic DNS
Device Setup
Failover
Firewall
Full Backup
Full Restore
Hardware bypass
Interfaces

		Supported in Management and VLAN Management in transparent mode)	
mtu	integer	Specify MTU; 300 to 65535 bytes.	None
ipAddress	object	IP Address option for the interface. (Supported in routed firewall mode. Supported in Management, VLAN Management and Bridge Group interfaces in transparent mode. PPPoE option is not supported in transparent mode)	One of: DHCP, PPPoE, StaticIP
selfLink	string	Unique URI for this resource object	None
bridgeGroupID	object	To assign an interface to a bridge group in transparent firewall mode. (Supported in transparent mode, except Management, VLAN Management and Bridge Group interface)	None

\* required attribute

It appears as though we *must* have the attribute of "kind" -- that makes sense, the payload should describe what type of object it is trying to manipulate, and the URI/endpoint describes the specific object. Try to remove all of the JSON data except for the "kind" and "interfaceDesc", change your description, then try to send your request.

Postman

File Edit View Help

New Import Runner + My Workspace No Environment Sign In

Filter History Collections

Configuration Time - 1 + ...

PATCH https://10.0.0.8/api/interfaces/physical/GigabitEthernet0\_API\_SLASH\_8 Params Send Save

MyCollection 1 request

1- { "kind": "object#GigabitInterface",  
2- "interfaceDesc": "Is it lunch yet?",  
3- }  
4 }

Body Cookies Headers (6) Test Results Status: 204 No Content Time: 188 ms Size: 281 B

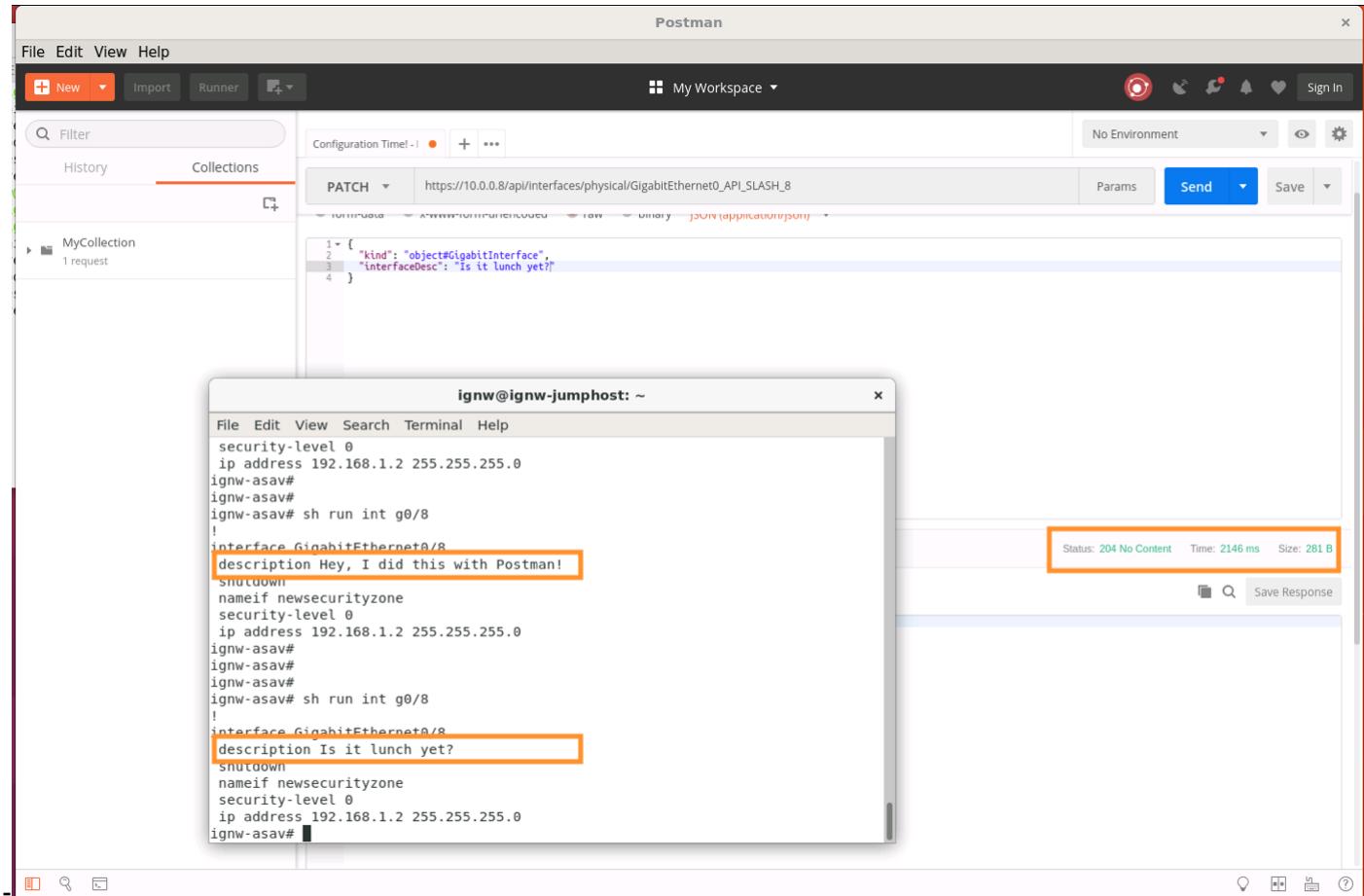
Pretty Raw Preview Text Save Response

1 |

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'File', 'Edit', 'View', 'Help', 'New', 'Import', 'Runner', 'My Workspace' (which is currently selected), 'No Environment', 'Sign In', and various icons. Below the navigation is a search bar labeled 'Filter' and tabs for 'History' and 'Collections' (the latter is highlighted). A configuration section shows 'Configuration Time - 1' with a red dot, a '+' button, and an '...' button. The main workspace shows a 'PATCH' request to the URL 'https://10.0.0.8/api/interfaces/physical/GigabitEthernet0\_API\_SLASH\_8'. The request body is displayed in a code editor with syntax highlighting, showing the following JSON:

```
1- { "kind": "object#GigabitInterface",  
2- "interfaceDesc": "Is it lunch yet?",  
3- }  
4 }
```

Below the request, there's a preview area with tabs for 'Body', 'Cookies', 'Headers (6)', and 'Test Results'. The 'Body' tab is selected, showing the JSON. The 'Test Results' tab indicates a successful response with 'Status: 204 No Content', 'Time: 188 ms', and 'Size: 281 B'. At the bottom of the preview area are buttons for 'Pretty', 'Raw', 'Preview', 'Text', and 'Save Response'.

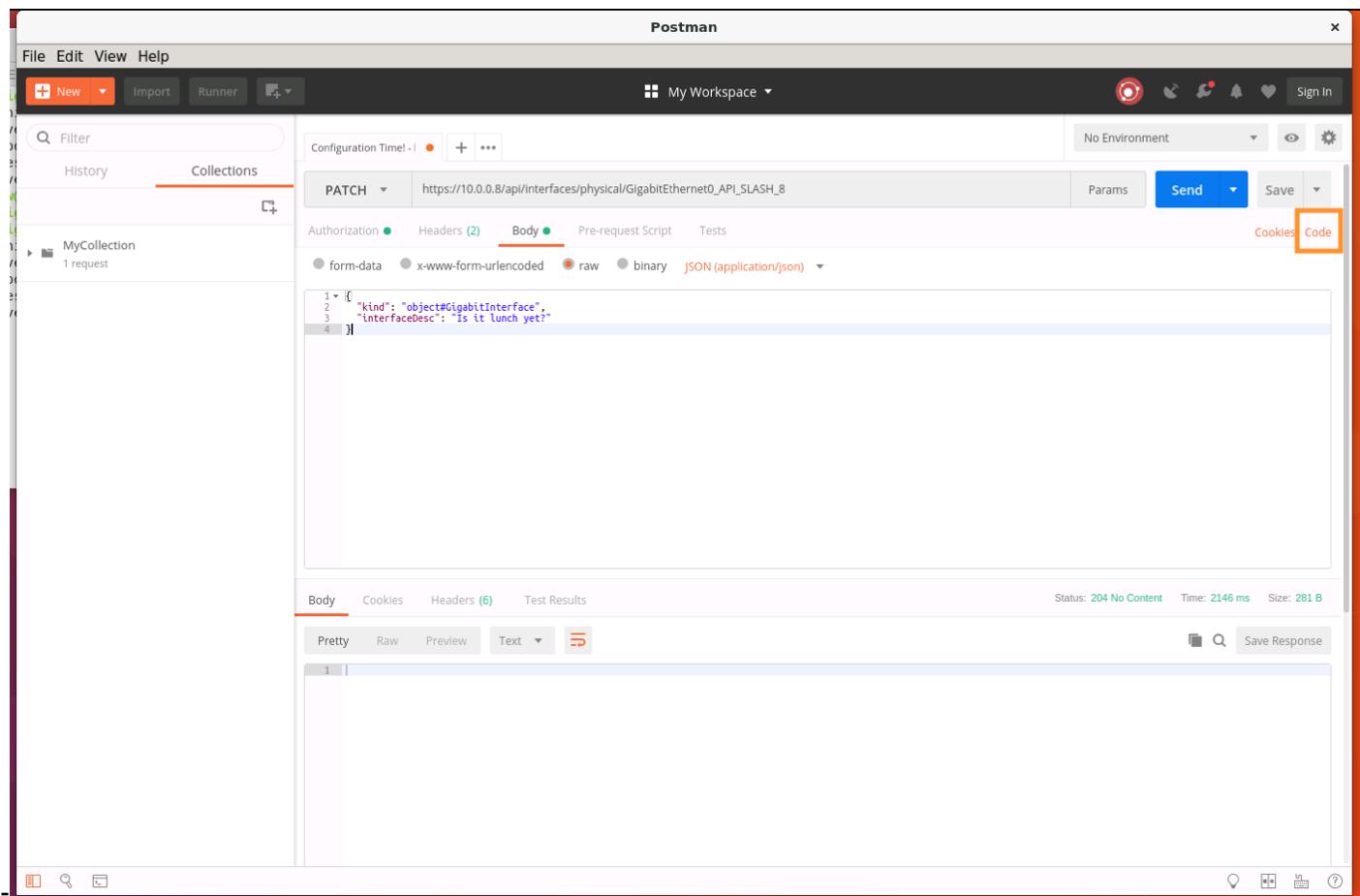


**Note:** The ASA API has some silly quirks that would permit configurations being made to G0/1 (for example) using the URI for G0/0 assuming the payload is "correct" - this would not be the best practice way to accomplish this so this guide is written to assume that the URI must match the object being configured in the payload, not just any object of the *type/class* of the payload.

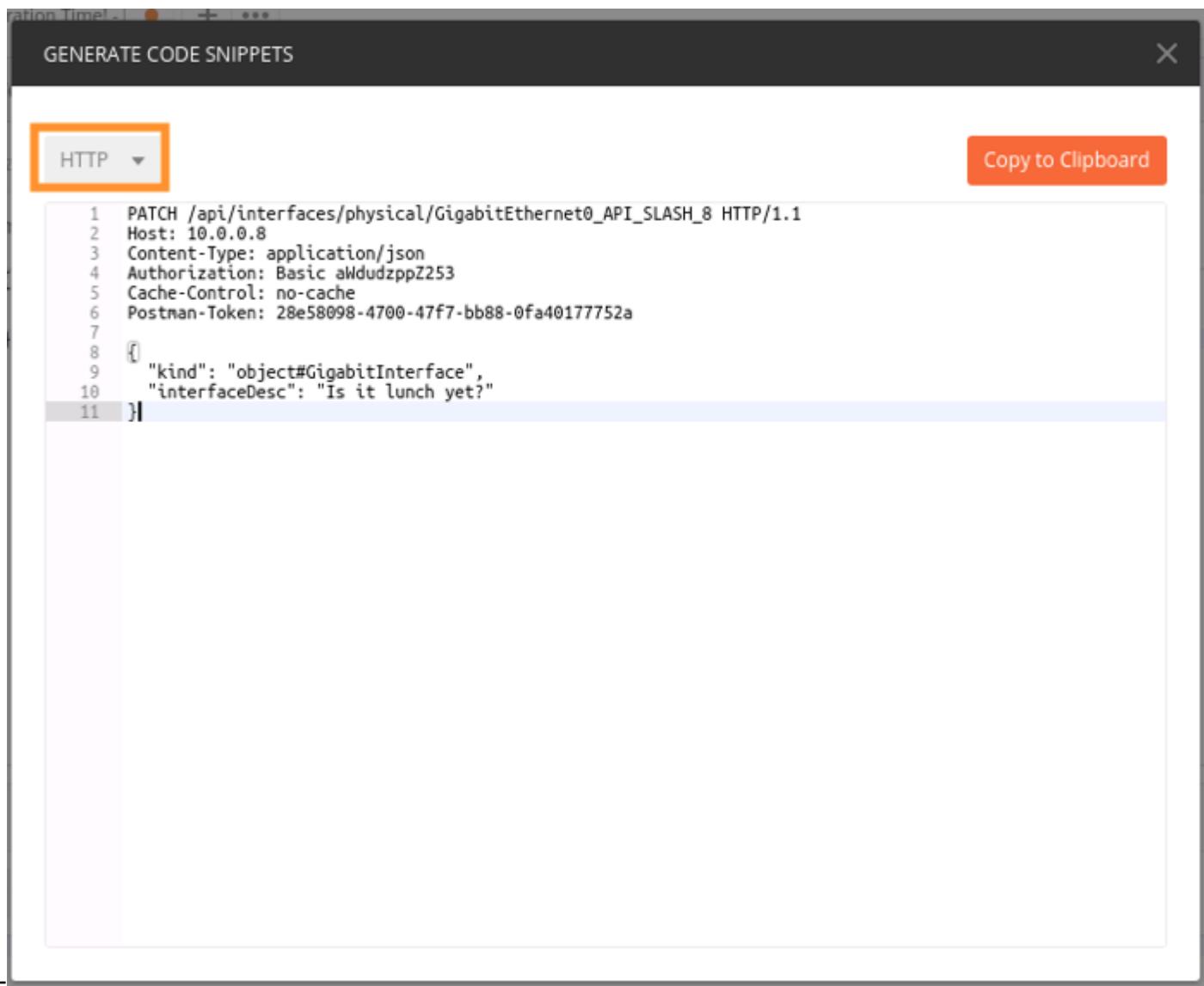
## Magical cURL/Python Output

Postman has another neat feature that may end up being helpful to you as you learn to do some of these same tasks with Python (or another language of your choice) -- it has the ability to export the requests that you've created in the Postman UI to one of many language outputs.

Select any of the requests you previously saved. In the top right side of the screen just below the "Send" button you'll see another link titled "Code". Click on the "Code" link.



By default when you open this page you will be greeted with what is essentially the "raw" HTTP data that represents the request that you selected. This by itself is a pretty neat way to take a peak under the covers at the actual data that your request is forming, but what is even cooler is the drop down box on the top left of this page.



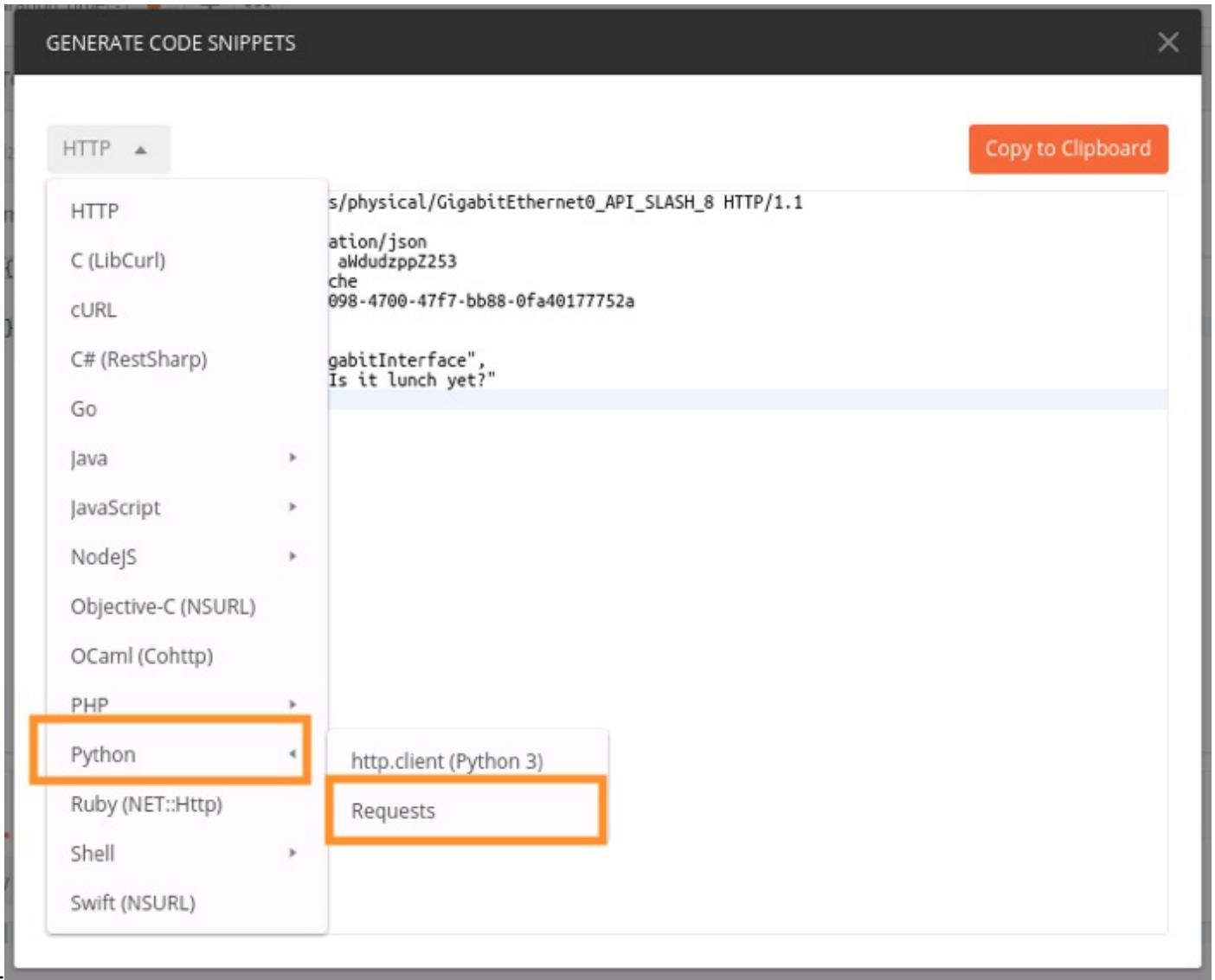
GENERATE CODE SNIPPETS X

HTTP ▼

Copy to Clipboard

```
1 PATCH /api/interfaces/physical/GigabitEthernet0_API_SLASH_8 HTTP/1.1
2 Host: 10.0.0.8
3 Content-Type: application/json
4 Authorization: Basic aWdudzppZ253
5 Cache-Control: no-cache
6 Postman-Token: 28e58098-4700-47f7-bb88-0fa40177752a
7
8 {
9   "kind": "object#GigabitInterface",
10  "interfaceDesc": "Is it lunch yet?"
11 }
```

Click on the drop down, and from the list highlight "Python", then select "Requests".



GENERATE CODE SNIPPETS X

Python Requests ▾

Copy to Clipboard

```
1 import requests
2
3 url = "https://10.0.0.8/api/interfaces/physical/GigabitEthernet0_API_SLASH_8"
4
5 payload = "{\n    \"kind\": \"object#GigabitInterface\", \n    \"interfaceDesc\": \"Is it lunch yet?\\n\""
6 headers = {
7     'Content-Type': "application/json",
8     'Authorization': "Basic aHdudzppZ253",
9     'Cache-Control': "no-cache",
10    'Postman-Token': "69d9bdbbe-4c44-4c56-a670-cc72d473af08"
11 }
12
13 response = requests.request("PATCH", url, data=payload, headers=headers)
14
15 print(response.text)
```

Wow! That is super handy!! This may not be in a style or format of code that you would write, but if you took this block of code and tried to run it in the Python interpreter or created a script from this it would indeed run!

Feel free to change the code style so you can check out what your request would look like in different languages. Do any of them make more sense than others to you? Python is generally lauded for its readable, but this is a great chance to compare it apples to apples to some other languages to see what works best in your brain!

# Python and APIs

## Overview and Objectives

In this lab you will expand on the understanding of APIs you gained from using Postman, and begin to interact with APIs via Python.

Objectives:

- Learn about the Python requests module
- Use the ncclient module to interact with a NETCONF API
- Interact with a RESTCONF API

## Introduction

Postman or other similar tools are a great way to begin to gain some familiarity with APIs, they may even prove useful for some deployment or query related tasks -- especially with tools like Postman "Runner", however these tools are more generally used for testing and development.

Working with APIs with a scripting or programming language opens up tons of opportunity to make interesting things happen! Integrating some scripting to query an API into an existing virtual machine deployment workflow for example may allow for additional configuration of non-virtual machine resources. Perhaps your firewall has an API and part of the work-flow should be to allow specific ports or protocols through to the new VM, or perhaps new network infrastructure needs to be provisioned to support the new workload.

While configuring additional resources is great, perhaps even more important is the ability to query API endpoints to validate that a deployment is a success, or to gather streaming telemetry from critical devices, parse it, and then make intelligent decisions about configurations or alerting.

APIs, as you've learned, enable all of this and more, and Python is a fantastic tool for interacting with APIs of all flavors!

## Introducing the Requests Module

The requests module is all about interacting with HTTP endpoints, and is the single most popular (per PyPi - the Python Package Index) Python module out there, and with good reason. Requests is widely regarded to have a very powerful and usable interface as you will soon find out!

You've already used Postman to interact with HTTP -- the REST API of the ASA. Now you'll get a chance to perform some of the same tasks, and some more, against the ASA API with Python!

## Authenticating to the ASA

Create a new text document called "requests\_task1.py" in the ubuntu (home) directory, we'll be working out of this file for the task.

As with before, it is a good idea to take a look at the documentation for any new modules before jumping straight in. Take a second to look at the very small example on the [requests home page](#).



**Requests**  
http for humans

Star 31,217

Requests is an elegant and simple HTTP library for Python, built for human beings.

Sponsored by [Linode](#) and other wonderful organizations.



Transactional Email Delivery. Start sending for Free with a 5-min Integration.

# Requests: HTTP for Humans

Release v2.18.4. ([Installation](#))

[license Apache 2.0](#) [wheel yes](#) [python 2.6, 2.7, 3.4, 3.5, 3.6](#) [codecov 90%](#) [Say Thanks! 🍻](#)

Requests is the only *Non-GMO* HTTP library for Python, safe for human consumption.

## Note:

The use of **Python 3** is *highly* preferred over Python 2. Consider upgrading your applications and infrastructure if you find yourself *still* using Python 2 in production today. If you are using Python 3, congratulations — you are indeed a person of excellent taste.  
—Kenneth Reitz

## Behold, the power of Requests:

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
u'{"type": "User", ...}'
>>> r.json()
{u'private_gists': 419, u'total_private_repos': 77, ...}
```

Well that doesn't look too complicated does it? It hopefully even looks a little bit familiar at this point!

We can clearly see the URL -- same as before, we also see some "auth" stuff happening -- probably pretty similar to before as well. It looks like there is a status code that is being printed out, and some stuff about the headers, encoding, text, and JSON response data.

Let's take the same basic data that we used with Postman and try to put it into requests form -- you may even want to pull up the Python code that Postman created for us before to use as a reference.

First things first, as requests is not part of the standard library, we will need to import the module at the top of our script (it has already been installed for you).

Next it would probably be a good plan to build out some variables that we will probably reuse a few times. Create a variable for the base URL of the ASA ("https://10.0.0.8/api/"), and for the username and password ("ignw", "ignw").

The screenshot shows a code editor window with a dark theme. The menu bar includes 'File', 'Edit', 'View', 'Selection', 'Find', 'Packages', and 'Help'. A file named 'requests\_task1.py' is open, indicated by a blue dot next to its name in the tab bar. The code itself is as follows:

```
1 import requests
2
3 url = 'https://10.0.0.8/api/'
4 username = 'ignw'
5 password = 'ignw'
6
7
```

With that out of the way we can take a look at the API documentation as well as the example that Postman created for us to figure out our next step. It looks like we're going to need to know the full URL we want to query -- let's "GET" the same one we started with before:

```
https://10.0.0.8/api/objects/networkobjects
```

We can craft our first request like so:

```
requests.get(f'{url}objects/networkobjects', auth=(username, password))
```

From the line above we can kind of see what is going on -- we are using the "requests" class, and the "get" method (function within the requests class -- that also happens to correspond to the type of HTTP method we are trying to run), and we are passing some arguments to that "get" method.

To craft the complete URL ("<https://10.1.2.102/api/objects/networkobjects>") we are using string interpolation to combine our "base url" with the rest of the endpoint we are trying to hit. Finally we are passing in some data in the "auth" variable (our username and password). Let's hope it all works!

Try to run your script.

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
    self.do_handshake()
File "/usr/lib/python3.6/ssl.py", line 1068, in do_handshake
    self._sslobj.do_handshake()
File "/usr/lib/python3.6/ssl.py", line 689, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:777)

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/requests/adapters.py", line 440, in send
    timeout=timeout
  File "/usr/lib/python3/dist-packages/urllib3/connectionpool.py", line 630, in urlopen
    raise SSLError(e)
urllib3.exceptions.SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:777)

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "requests_task1.py", line 7, in <module>
    resp = requests.get(f'{url}objects/networkobjects', auth=(username, password))
  File "/usr/lib/python3/dist-packages/requests/api.py", line 72, in get
    return request('get', url, params=params, **kwargs)
  File "/usr/lib/python3/dist-packages/requests/api.py", line 58, in request
    return session.request(method=method, url=url, **kwargs)
  File "/usr/lib/python3/dist-packages/requests/sessions.py", line 502, in request
    resp = self.send(prep, **send_kwargs)
  File "/usr/lib/python3/dist-packages/requests/sessions.py", line 612, in send
    r = adapter.send(request, **kwargs)
  File "/usr/lib/python3/dist-packages/requests/adapters.py", line 514, in send
    raise SSLError(e, request=request)
requests.exceptions.SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:777)
ignw@ignw-jumphost:~$
```

```
requests_task1.py — — Atom
File Edit View Selection Find Packages Help
1 requests_task1.py •
2
3 url = 'https://10.0.0.8/api/'
4 username = 'ignw'
5 password = 'ignw'
6
7 requests.get(f'{url}objects/networkobjects', auth=(username, password))
8
9
10
11
```

Uhh! That doesn't exactly look like a successful run! The good news is that the error does seem fairly obvious -- we have failed to verify the SSL certificate. Recall that we had to disable SSL certificate verification in Postman as well. While we know it is a poor security practice to do so, let's see if we can figure out how to disable certificate verification with requests.

A bit of searching yields the following from the requests documentation:

## SSL Cert Verification

Requests verifies SSL certificates for HTTPS requests, just like a web browser. By default, SSL verification is enabled, and Requests will throw a `SSLError` if it's unable to verify the certificate:

```
>>> requests.get('https://requestb.in')
requests.exceptions.SSLError: hostname 'requestb.in' doesn't match either of '*'
```

I don't have SSL setup on this domain, so it throws an exception. Excellent. GitHub does though:

```
>>> requests.get('https://github.com')
<Response [200]>
```

You can pass `verify` the path to a `CA_BUNDLE` file or directory with certificates of trusted CAs:

```
>>> requests.get('https://github.com', verify='/path/to/certfile')
```

or persistent:

```
s = requests.Session()
s.verify = '/path/to/certfile'
```

### Note:

If `verify` is set to a path to a directory, the directory must have been processed using the `c_rehash` utility supplied with OpenSSL.

This list of trusted CAs can also be specified through the `REQUESTS_CA_BUNDLE` environment variable.

Requests can also ignore verifying the SSL certificate if you set `verify` to `False`:

```
>>> requests.get('https://kennethreitz.org', verify=False)
<Response [200]>
```

By default, `verify` is set to `True`. Option `verify` only applies to host certs.

Modify your script to set "verify=False" in your request and try to run it once more.

The screenshot shows a terminal window at the top and an Atom code editor below it. The terminal window has a title bar "ignw@ignw-jumphost: ~" and a menu bar "File Edit View Search Terminal Help". The command `python3 requests\_task1.py` is run, resulting in a warning message about unverified HTTPS requests. The code editor shows a file named "requests\_task1.py" with the following content:

```
File Edit View Selection Find Packages Help
  requests_task1.py
3 url = 'https://10.0.0.8/api/'
4 username = 'ignw'
5 password = 'ignw'
6
7 requests.get(f'{url}objects/networkobjects', auth=(username, password), verify=False)
8
9
```

That was pretty uneventful, but at least there was no exception raised! Requests is nice enough to warn us that what we are doing is not exactly the greatest idea in the world from a security perspective. If we don't do something about it requests will show us this message every single time we make a request in our lab since all the certificates are self signed.

Paste the following into your code to hide these warnings:

```
from requests.packages.urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
```

Now that we are getting somewhere, how do we know if our request was successful? Look back to the example from the requests website. It looks like the example is assigning the value of the request they made to a variable called "r". In this case, "r" is short for response. Short variables are nice, but let's be a bit more verbose and call our variable "resp". Let's also print out "resp" and, just like the example, "resp.status\_code" to see what is happening.

```
resp = requests.get(f'{url}objects/networkobjects', auth=(username, password),
                    verify=False)
print(resp)
print(resp.status_code)
```

The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~" running on a Linux system. The terminal displays the execution of a Python script named "requests\_task1.py". The script uses the "requests" library to make a GET request to a local API endpoint (`https://10.0.0.8/api/`) and prints the response status code (200). The terminal also shows a warning message from the "urllib3" library about SSL certificate verification.

```
File Edit View Selection Find Packages Help
File Edit View Selection Find Packages Help
1 import requests
2 from requests.packages.urllib3.exceptions import InsecureRequestWarning
3 import json
4
5
6 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
7
8 url = 'https://10.0.0.8/api/'
9 username = 'ignw'
10 password = 'ignw'
11
12 resp = requests.get(f'{url}objects/networkobjects', auth=(username, password),
13                      verify=False)
14 print(resp)
15 print(resp.status_code)

ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 requests_task1.py
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:854: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
InsecureRequestWarning)
ignw@ignw-jumphost:~$ python3 requests_task1.py
<Response [200]>
200
ignw@ignw-jumphost:~$
```

That still isn't a ton of information back, but we know that a "200" status code is a good sign! What else do you think the "resp" object holds? Based on the example on the requests website what other data do you think we can get out of this object?

One way to find out would be to use "dir". "dir", short for "directory" is a built in Python function which, [per the documentation](#), "return the list of names in the current local scope". Try to print out the "dir" of your resp object.

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 requests_task1.py
/usr/lib/python3/dist-packages/urllib3/connectionpool.py:854: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning)
ignw@ignw-jumphost:~$ python3 requests_task1.py
<Response [200]>
200
ignw@ignw-jumphost:~$ python3 requests_task1.py
<Response [200]>
200
['__attrs__', '__bool__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__enter__', '__eq__', '__exit__', '__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__nonzero__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_content', '_content_consumed', '_next', 'apparent_encoding', 'close', 'connection', 'content', 'cookies', 'elapsed', 'encoding', 'headers', 'history', 'is_permanent_redirect', 'is_redirect', 'iter_content', 'iter_lines', 'json', 'links', 'next', 'ok', 'raise_for_status', 'raw', 'reason', 'request', 'status_code', 'text', 'url']
ignw@ignw-jumphost:~$
```

requests\_task1.py — — Atom

File Edit View Selection Find Packages Help

```
1 import requests
2 from requests.packages.urllib3.exceptions import InsecureRequestWarning
3 import json
4
5
6 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
7
8 url = 'https://10.0.0.8/api/'
9 username = 'ignw'
10 password = 'ignw'
11
12 resp = requests.get(f'{url}objects/networkobjects', auth=(username, password),
13                      verify=False)
14 print(resp)
15 print(resp.status_code)
16 print(dir(resp))
```

Based on the output above, you can see that "status\_code" is one of the attributes that the "resp" object contains. We could also get the "text" output from our response.

Try to query all of the physical interfaces of our ASA just like you did with Postman.

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
LASH_4'}, {'kind': 'object#GigabitInterface', 'selfLink': 'https://10.0.0.8/api/interfaces/physical/GigabitEthernet0_API_SLASH_5', 'hardwareID': 'GigabitEthernet0/5', 'interfaceDesc': '', 'channelGroupID': '', 'channelGroupMode': 'active', 'duplex': 'auto', 'flowcontrolOn': False, 'flowcontrolHigh': -1, 'flowcontrolLow': -1, 'flowcontrolPeriod': -1, 'forwardTrafficCX': False, 'forwardTrafficSFR': False, 'lacpPriority': -1, 'activeMacAddress': '', 'standByMacAddress': '', 'managementOnly': False, 'mtu': 1500, 'name': '', 'securityLevel': -1, 'shutdown': True, 'speed': 'auto', 'ipAddress': 'NoneSelected', 'ipv6Info': {'enabled': False, 'autoConfig': False, 'enforceEUI64': False, 'managedAddressConfig': False, 'nsInterval': 1000, 'dadAttempts': 1, 'nDiscoveryPrefixList': []}, 'otherStatefulConfig': False, 'routerAdvertInterval': 200, 'routerAdvertIntervalUnit': 'sec', 'routerAdvertLifetime': 1800, 'suppressRouterAdvert': False, 'reachableTime': 0, 'ipv6Addresses': [], 'kind': 'object#Ipv6InterfaceInfo'}, 'objectId': 'GigabitEthernet0_API_SLASH_5'}, {'kind': 'object#GigabitInterface', 'selfLink': 'https://10.0.0.8/api/interfaces/physical/GigabitEthernet0 API SLASH 6', 'hardwareID': 'GigabitEthernet0/6', 'interfaceDesc': '', 'channelGroupID': '', 'channelGroupMode': 'active', 'duplex': 'auto', 'flowcontrolOn': False, 'flowcontrolHigh': -1, 'flowcontrolLow': -1, 'flowcontrolPeriod': -1, 'forwardTrafficCX': False, 'forwardTrafficSFR': False, 'lacpPriority': -1, 'activeMacAddress': '', 'standByMacAddress': '', 'managementOnly': False, 'mtu': 1500, 'name': '', 'securityLevel': -1, 'shutdown': True, 'speed': 'auto', 'ipAddress': 'NoneSelected', 'ipv6Info': {'enabled': False, 'autoConfig': False, 'enforceEUI64': False, 'managedAddressConfig': False, 'nsInterval': 1000, 'dadAttempts': 1, 'nDiscoveryPrefixList': []}, 'otherStatefulConfig': False, 'routerAdvertInterval': 200, 'routerAdvertIntervalUnit': 'sec', 'routerAdvertLifetime': 1800, 'suppressRouterAdvert': False, 'reachableTime': 0, 'ipv6Addresses': [], 'kind': 'object#Ipv6InterfaceInfo'}, 'objectId': 'GigabitEthernet0_API_SLASH_6'}, {'kind': 'object#GigabitInterface', 'selfLink': 'https://10.0.0.8/api/interfaces/physical/GigabitEthernet0_API_SLASH_7', 'hardwareID': 'GigabitEthernet0/7', 'interfaceDesc': '', 'channelGroupID': '', 'channelGroupMode': 'active', 'duplex': 'auto', 'flowcontrolOn': False, 'flowcontrolHigh': -1, 'flowcontrolLow': -1, 'flowcontrolPeriod': -1, 'forwardTrafficCX': False, 'forwardTrafficSFR': False, 'lacpPriority': -1, 'activeMacAddress': '', 'standByMacAddress': '', 'managementOnly': False, 'mtu': 1500, 'name': '', 'securityLevel': -1, 'shutdown': True, 'speed': 'auto', 'ipAddress': 'NoneSelected', 'ipv6Info': {'enabled': False, 'autoConfig': False, 'enforceEUI64': False, 'managedAddressConfig': False, 'nsInterval': 1000, 'dadAttempts': 1, 'nDiscoveryPrefixList': []}, 'otherStatefulConfig': False, 'routerAdvertInterval': 200, 'routerAdvertIntervalUnit': 'sec', 'routerAdvertLifetime': 1800, 'suppressRouterAdvert': False, 'reachableTime': 0, 'ipv6Addresses': [], 'kind': 'object#Ipv6InterfaceInfo'}, 'objectId': 'GigabitEthernet0_API_SLASH_7'}]}
ignw@ignw-jumphost:~$
```

Are you able to get an output like that shown above? Recall that the URL you will want to "GET" is:

```
https://10.0.0.8/api/interfaces/physical
```

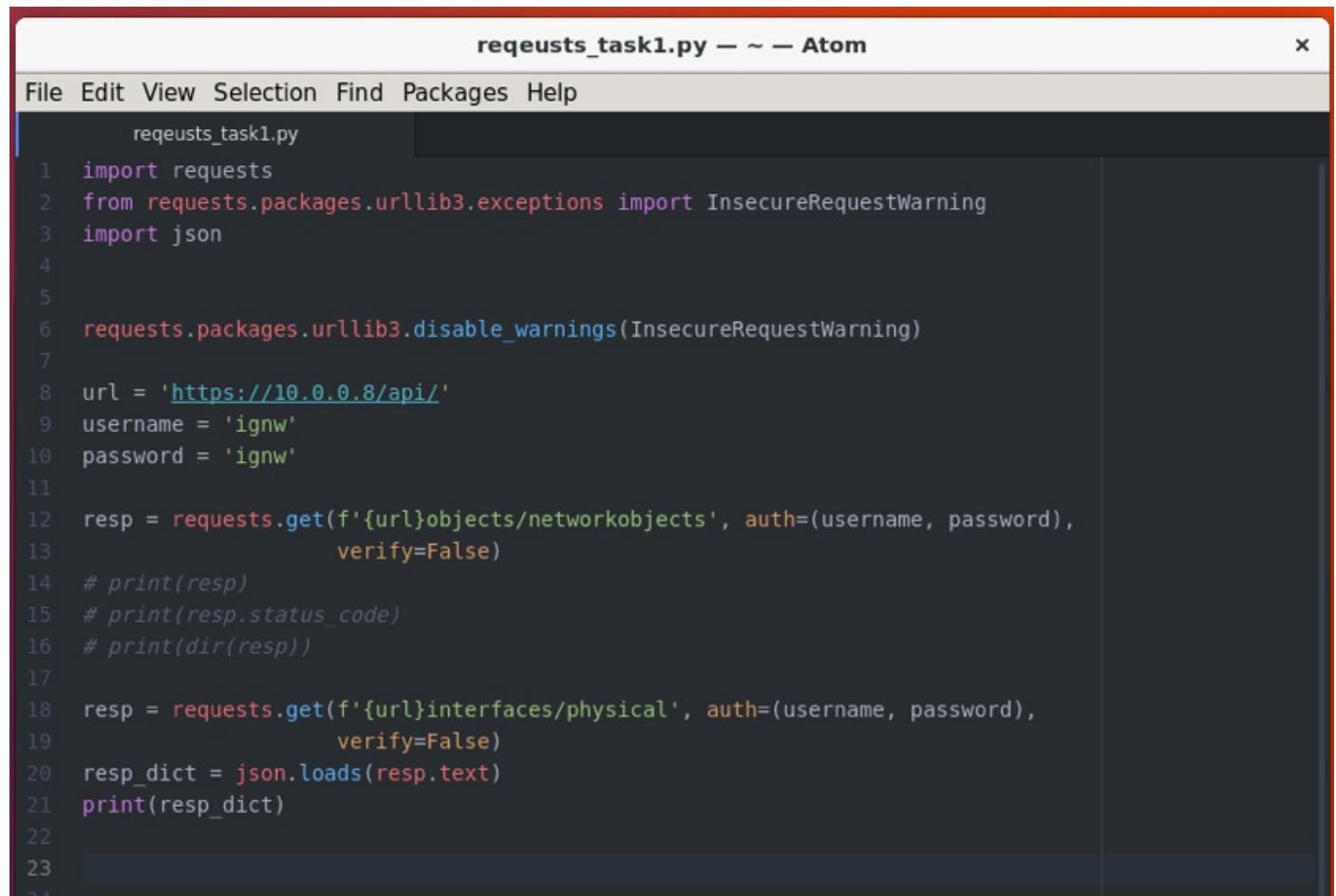
You'll then want to print out the "text" data in your response object.

Now that we're getting somewhere, how about we turn this into something useful. With Postman we got the JSON data "pretty printed" for us, which we can do with Python as well, but, it might be even more useful to take the response data and turn it into a Python dictionary. How do you think we can go about doing that?

Recall that JSON is quite similar (though not exactly the same!) as a Python dictionary... so if we can get a "blob" of JSON data back in text form from the API (which we know we can) maybe there is an easy way to convert that to a dictionary?

There is! The "json" module is part of the standard library and has a method called "loads" (load string) that can convert a JSON-like string into a dictionary. You can check out the documentation [here](#).

Create a new variable called "resp\_dict" that is a dictionary representation of the response, and then print it.



The screenshot shows a code editor window titled "requests\_task1.py" in the Atom text editor. The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. The code itself is as follows:

```
File Edit View Selection Find Packages Help
  requests_task1.py — ~ — Atom
1 import requests
2 from requests.packages.urllib3.exceptions import InsecureRequestWarning
3 import json
4
5
6 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
7
8 url = 'https://10.0.0.8/api/'
9 username = 'ignw'
10 password = 'ignw'
11
12 resp = requests.get(f'{url}objects/networkobjects', auth=(username, password),
13                      verify=False)
14 # print(resp)
15 # print(resp.status_code)
16 # print(dir(resp))
17
18 resp = requests.get(f'{url}interfaces/physical', auth=(username, password),
19                      verify=False)
20 resp_dict = json.loads(resp.text)
21 print(resp_dict)
22
23
24
```

**Tip:** Make sure you don't forget to import the json module!

With what you've learned about dictionaries, can you use this data to replicate the following output:

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 requests_task1.py
ignw@ignw-jumphost:~$ python3 requests_task1.py
The ASA has 10 interfaces. Named as follows:
['GigabitEthernet0/1', 'Management0/0', 'GigabitEthernet0/0', 'GigabitEthernet0/8', 'GigabitEth
ernet0/2', 'GigabitEthernet0/3', 'GigabitEthernet0/4', 'GigabitEthernet0/5', 'GigabitEthernet0/
6', 'GigabitEthernet0/7']
ignw@ignw-jumphost:~$
```

Using Python we can much more easily capture data that is relevant to us. Robots (computers) are much, much, better at parsing (structured) data than we are as humans, so we may as well put them to work!

requests\_task1.py — — Atom

```
File Edit View Selection Find Packages Help
rqeuests_task1.py

1 import requests
2 from requests.packages.urllib3.exceptions import InsecureRequestWarning
3 import json
4
5
6 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
7
8 url = 'https://10.0.0.8/api/'
9 username = 'ignw'
10 password = 'ignw'
11
12 resp = requests.get(f'{url}objects/networkobjects', auth=(username, password),
13                      verify=False)
14 # print(resp)
15 # print(resp.status_code)
16 # print(dir(resp))
17
18 resp = requests.get(f'{url}interfaces/physical', auth=(username, password),
19                      verify=False)
20 resp_dict = json.loads(resp.text)
21 # print(resp_dict)
22
23 ints_qty = resp_dict['rangeInfo']['total']
24 ints_names = []
25
26 for i in resp_dict['items']:
27     ints_names.append(i['hardwareID'])
28
29 print(f'The ASA has {ints_qty} interfaces. Named as follows: ')
30 print(ints_names)
31 |
```

rqeuests\_task1.py 31:1 LF UTF-8 Python 0 files

## Requests PUT? POST? PATCH?

Create a new text document called "requests\_task2.py" in the ubuntu (home) directory, we'll be working out of this file for the task.

Copy from your previous script the variables you hard coded and the imports, you'll need most/all of these again! Just like we did with Postman, let's try to PUT some configuration on the ASA.

How do you think we can go about configuring some network objects on the ASA? Hop back into the ASA REST API Documentation and Console that you accessed before.

Navigate to the "Objects" section, then "networkobjects", click on "PUT" and check out the examples:

The screenshot shows the ASA REST API Documentation & Console interface. On the left, there's a sidebar with various icons and menu items like API INFO, AAA, Access, Bulk, CLI, Certificate, Context Manager, DHCP Server, DNS Client/DNS, Device Setup, Failover, Firewall, Full Backup, Full Restore, Hardware, Interfaces, Interfaces (System), Licensing, Logging, Management access, Monitoring, NAT, Objects (which is selected and highlighted with a yellow box), Routing, Save, Service policy, and Token Services.

The main content area has a blue header "ASA REST API Documentation & Console". Below it, a navigation bar shows the URL [https://10.1.2.102/doc/#feature/objects/networkobjects/{objectId}\\_PUT](https://10.1.2.102/doc/#feature/objects/networkobjects/{objectId}_PUT) and includes links for GET, DELETE, POST, PATCH, and PUT (the latter is highlighted with a yellow box). A sub-header says "/api/objects/networkobjects API operations on network objects." and "Implementation Notes Update an existing network object." There are also "Examples" and "Implementation Notes" sections.

A modal window titled "Example Request(s)" is open, showing a "Test PUT of network object". It contains a "Request Data" section with the following JSON payload:

```

PUT /api/objects/networkobjects/newASA_Demo_NObj_1190
{
    "host": {
        "kind": "IPv4Address",
        "value": "1.10.8.20"
    },
    "kind": "object#NetworkObj",
    "name": "ASA_Demo_NObj_1111",
    "objectId": "newASA_Demo_NObj_1190"
}
  
```

The "Response Status Code 204" is shown below the request data. To the right of the modal, the "API CONSOLE" section shows the URL </api/objects/networkobjects>. It has input fields for "objectId" (containing "objectID") and "body". A note says "Parameters for the network object to be replaced." and a "PUT" button is present. Below the console are tabs for "Response Text", "Response Info", and "Request Info".

With this information you should be able to craft a payload to send a network object on the ASA. You can name it whatever you'd like, and provide any IP address you wish as it will not affect the lab.

Your "payload" or "body" (like in Postman) or "data" is simply the data that you wish to send to the device. Note that there are, as with everything in programming, a few ways to handle this. For now the simplest way is to just create a string that looks like json that we can send to our firewall.

The screenshot shows a code editor window titled "requests\_task2.py — ~ — Atom". The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. The code itself is a Python script:

```
1 import requests
2 from requests.packages.urllib3.exceptions import InsecureRequestWarning
3
4
5 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
6
7 url = 'https://10.0.0.8/api/'
8 username = 'ignw'
9 password = 'ignw'
10
11 payload = '''
12 {
13     "host": {
14         "kind": "IPv4Address",
15         "value": "8.8.8.8"
16     },
17     "kind": "object#NetworkObj",
18     "name": "leGoogle",
19     "objectId": "leGoogle"
20 }
21 '''
22
```

Note that the triple quotes signify a multi-line string in Python. Here we've simply taken the example from the ASA REST API Documentation, and changed the IP address and name of our object.

Printing out our payload variable you can see that this is just a text representation of the JSON data (JSON is just text after all) that the documentation tells us we need to send to the firewall.

The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~". The menu bar includes File, Edit, View, Search, Terminal, and Help. The command "python3 requests\_task2.py" is run, and the output is:

```
ignw@ignw-jumphost:~$ python3 requests_task2.py
{
    "host": {
        "kind": "IPv4Address",
        "value": "8.8.8.8"
    },
    "kind": "object#NetworkObj",
    "name": "leGoogle",
    "objectId": "leGoogle"
}
ignw@ignw-jumphost:~$
```

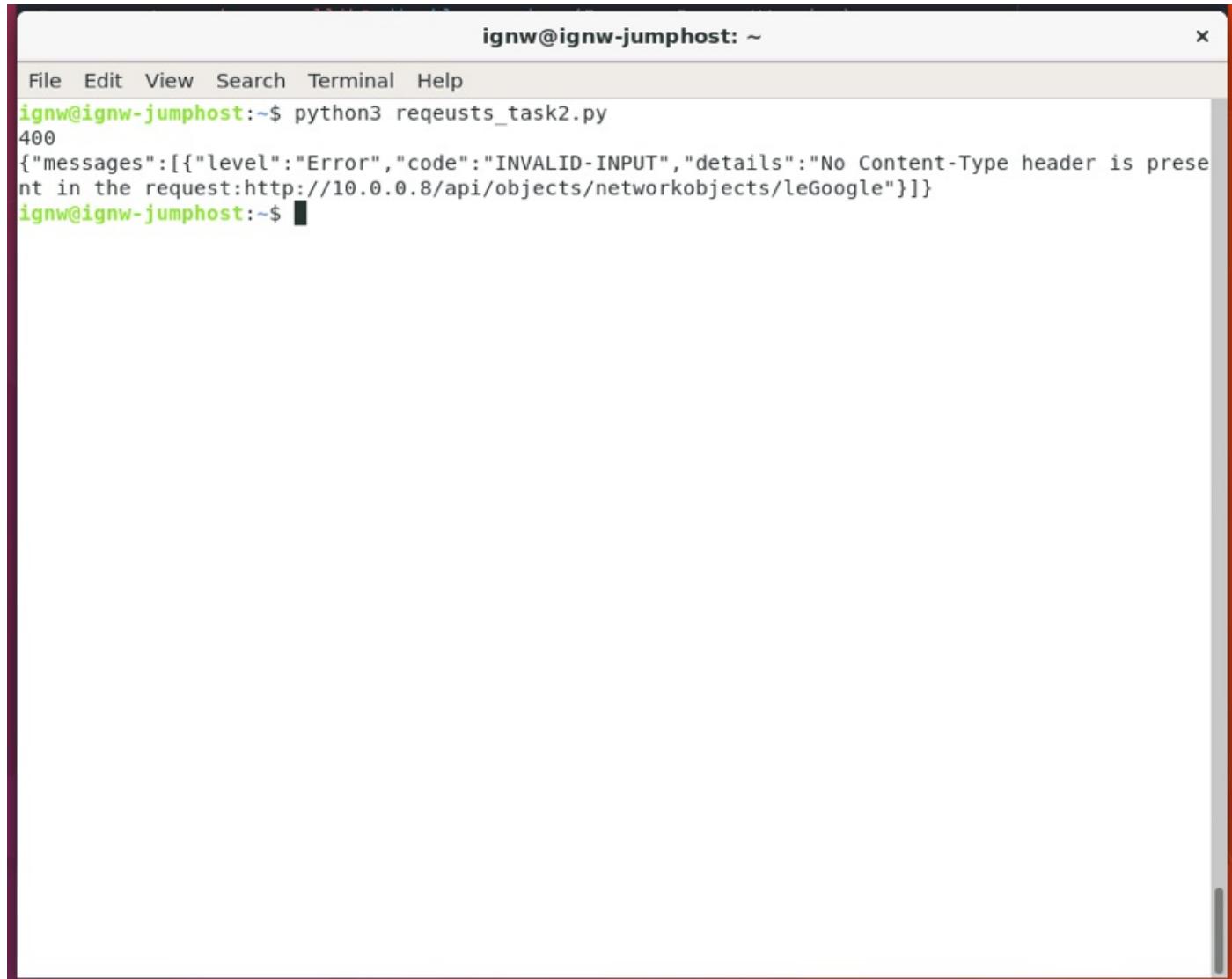
Great, now let's try to PUT our data out on the ASAv. Stealing from your previous script, create your request and assign the value of the outcome to the "resp" variable. You'll need to add an argument as outlined below:

```
resp = requests.put(f'{url}objects/networkobjects/leGoogle', auth=(username, password),
                     data=payload, verify=False)
```

Run your script, what happens? Oh, that's right -- nothing! Add some print statements so that you can see what the result of your request is:

```
print(resp.status_code)
print(resp.text)
```

Once more, run your script.

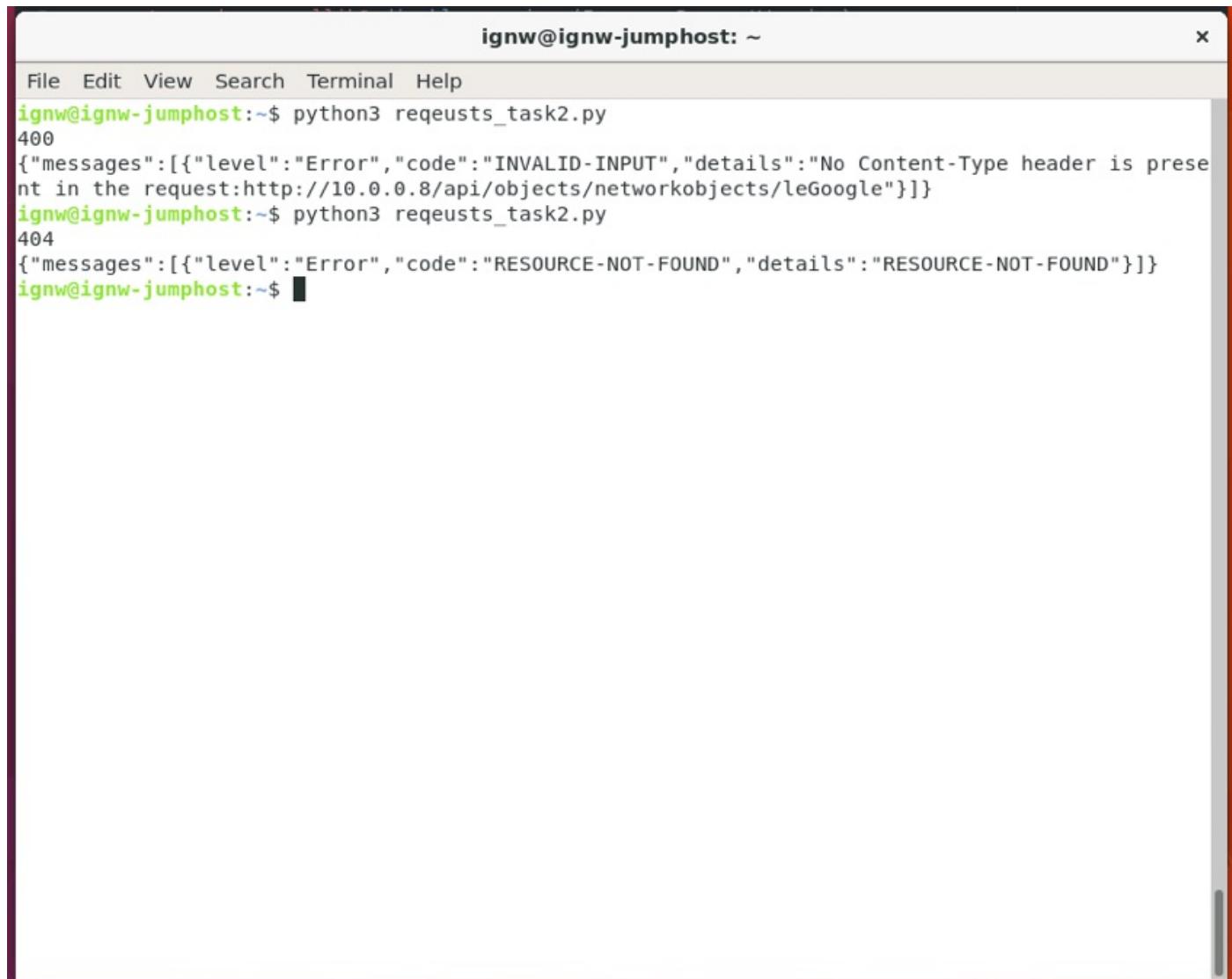


A screenshot of a terminal window titled "ignw@ignw-jumphost: ~". The window has a standard OS X-style interface with a menu bar at the top. The terminal content shows the command "python3 requests\_task2.py" being run, followed by an error message indicating a 400 status code and an "INVALID-INPUT" error. The error message also notes that there is no Content-Type header present in the request.

```
ignw@ignw-jumphost:~$ python3 requests_task2.py
400
{"messages": [{"level": "Error", "code": "INVALID-INPUT", "details": "No Content-Type header is present in the request: http://10.0.0.8/api/objects/networkobjects/leGoogle"}]}
ignw@ignw-jumphost:~$
```

Hmm... "INVALID-INPUT" (in all caps no less!). That seems bad! We also get a note about "No Content-Type header" -- recall how we created that in Postman? It looks like we need to add that to our requests. Luckily,

the requests documentation is pretty easy to find information in! Check out the [quickstart guide](#) and search for "headers" -- try to add a header to your script and run it again. Remember that we want our "Content-Type" to be "application/json".



A screenshot of a terminal window titled "ignw@ignw-jumphost: ~". The window has a standard OS X-style title bar with icons for close, minimize, and zoom. The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The main terminal area shows the following command-line session:

```
ignw@ignw-jumphost:~$ python3 requests_task2.py
{"messages": [{"level": "Error", "code": "INVALID-INPUT", "details": "No Content-Type header is present in the request: http://10.0.0.8/api/objects/networkobjects/leGoogle"}]}
ignw@ignw-jumphost:~$ python3 requests_task2.py
404
{"messages": [{"level": "Error", "code": "RESOURCE-NOT-FOUND", "details": "RESOURCE-NOT-FOUND"}]}
ignw@ignw-jumphost:~$
```

requests\_task2.py — — Atom

File Edit View Selection Find Packages Help

```
1 import requests
2 from requests.packages.urllib3.exceptions import InsecureRequestWarning
3
4
5 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
6
7 url = 'https://10.0.0.8/api/'
8 username = 'ignw'
9 password = 'ignw'
10
11 payload = ''
12 {
13     "host": {
14         "kind": "IPv4Address",
15         "value": "8.8.8.8"
16     },
17     "kind": "object#NetworkObj",
18     "name": "leGoogle",
19     "objectId": "leGoogle"
20 }
21 ...
22
23 headers = {'Content-Type': 'application/json'}
24
25 resp = requests.put(f'{url}objects/networkobjects/leGoogle',
26                      auth=(username,password), data=payload,
27                      headers=headers, verify=False)
28
29 print(resp.status_code)
30 print(resp.text)
31
```

requests\_task2.py 1:16 LF UTF-8 Python 0 files

Uhoh, still not good. Now we are getting "RESOURCE-NOT-FOUND" (still with the caps!!). Take a peak back at the ASA REST API Documentation & Console page.

Recall that not all APIs implement methods in the same way. Mozilla outlined PUT as follows: "The HTTP PUT request method creates a new resource or replaces a representation of the target resource with the request payload." -- certainly not what we're seeing happen here is it!

Back in the ASA documentation take a closer look at the PUT documentation for our network object.

ASA REST API - Mozilla Firefox

ASA REST API

https://10.1.2.102/doc/#feature/objects/networkobjects/{objectId}\_PUT

ASA REST API Documentation & Console

CISCO

**API INFO**  
ASA Version: 9.9(1)2

**Implementation Notes**  
Update an existing network object.

**Parameters**

Parameter	Required	Description	Type	Data Type
objectId	true	The object ID of the network object to be updated.	path	string
body	true	Parameters for the network object to be replaced.	body	TimeRan...

**Response**  
Response Content Type application/json

**NetworkObj Model**

Field	Value	Description	Constraints
kind *	string	The kind of this resource object.	None
description	string	description	None
objectId	string	Unique ID of this resource object	None
host *	object	complete network service data in a ("kind", "Value") pair format	None
host.kind *	string	kind	None
host.value *	string	value	None
selfLink	string	Unique URI for this resource object	None
name *	string	name	None

\* required attribute

**API CONSOLE**

/api/objects/networkobjects

objectId  
The object ID of the network object to be updated.

body  
Parameters for the network object to be replaced.

+ query parameter  
PUT

Response Text Response Info Request Info

Export operation in.. ▾

Interesting! What about POST?

**ASA REST API - Mozilla Firefox**

ASA REST API

https://10.1.2.102/doc/#feature/objects/networkobjects\_POST

ASA REST API Documentation & Console

**API INFO**  
ASA Version: 9.9(1)2

**Implementation Notes**  
Add a new network object.

**Parameters**

Parameter	Required	Description	Type	Data Type
body	true	Parameters for the network object to be added.	body	NetworkObject

**Response**  
Response Content Type application/json

**NetworkObj Model**

Field	Value	Description	Constraints
kind *	string	The kind of this resource object.	None
description	string	description	None
objectId	string	Unique ID of this resource object	None
host *	object	complete network service data in a ("kind", "Value") pair format	None
host.kind *	string	kind	None
host.value *	string	value	None
selfLink	string	Unique URI for this resource object	None
name *	string	name	None

\* required attribute

**/api/objects/networkservicegroups** GET DELETE POST PATCH PUT

API operations on network service (protocol) group objects

**API CONSOLE**

/api/objects/networkobjects

body

Parameters for the network object to be added.

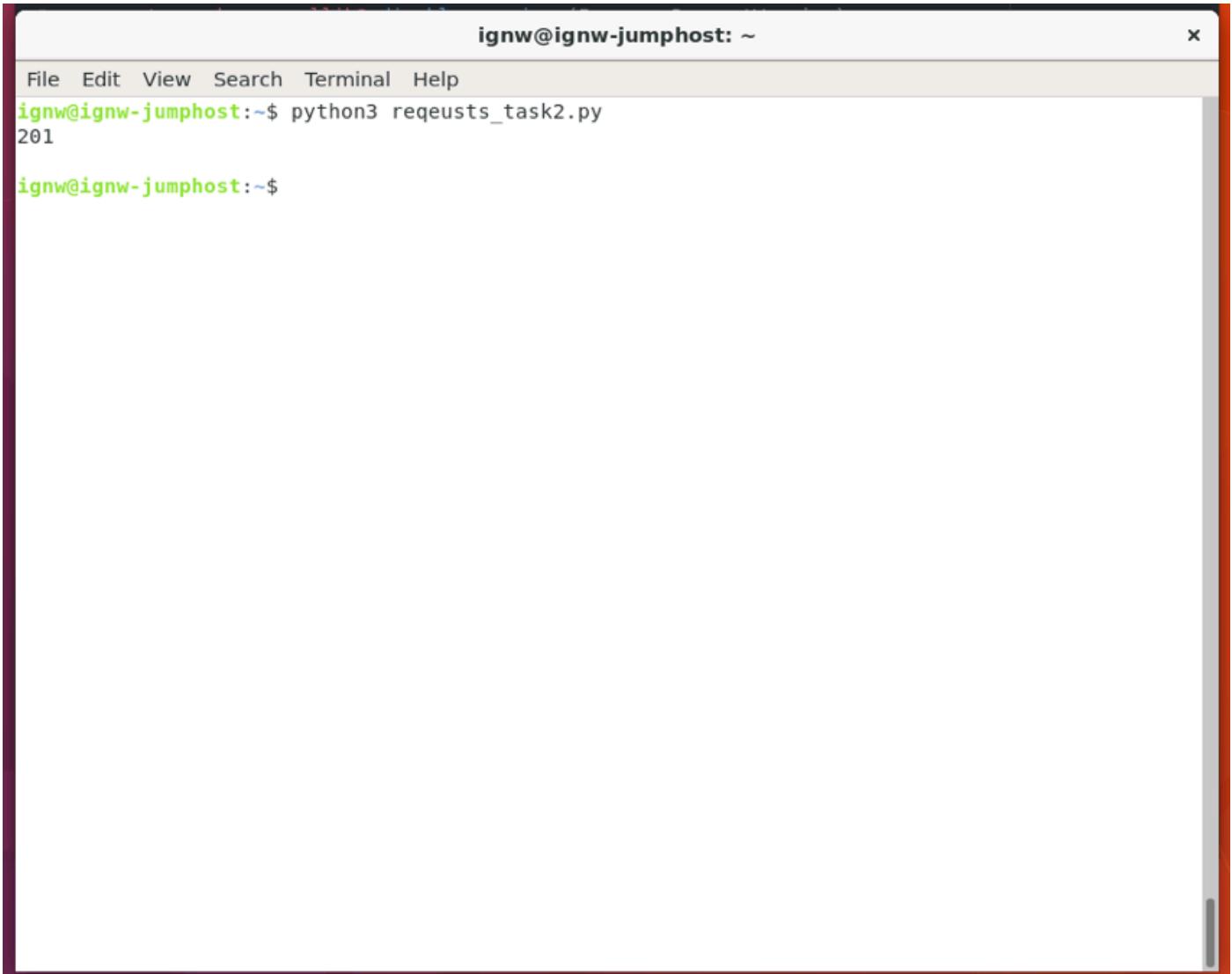
+ query parameter

POST

Response Text Response Info Request Info

Export operation in.. ▾

So why were we able to use PUT in the Postman labs? Simple, we were "PUT"ing to an existing object -- the network interface. In this case we are *creating* a new object, and thus per this API we must use the POST method. Change your script to a POST and see if that does the trick!



A screenshot of a terminal window titled "ignw@ignw-jumphost: ~". The window has a standard OS X-style title bar with a close button ("x") in the top right corner. Below the title bar is a menu bar with options: File, Edit, View, Search, Terminal, and Help. The main pane of the terminal shows the following command and its output:

```
ignw@ignw-jumphost:~$ python3 requests_task2.py
201

ignw@ignw-jumphost:~$
```

requests\_task2.py — — Atom

File Edit View Selection Find Packages Help

```
1 import requests
2 from requests.packages.urllib3.exceptions import InsecureRequestWarning
3
4
5 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
6
7 url = 'https://10.0.0.8/api/'
8 username = 'ignw'
9 password = 'ignw'
10
11 payload = ''
12 {
13     "host": {
14         "kind": "IPv4Address",
15         "value": "8.8.8.8"
16     },
17     "kind": "object#NetworkObj",
18     "name": "leGoogle",
19     "objectId": "leGoogle"
20 }
21 ...
22
23 headers = {'Content-Type': 'application/json'}
24
25 resp = requests.post(f'{url}objects/networkobjects/leGoogle',
26                       auth=(username,password), data=payload,
27                       headers=headers, verify=False)
28
29 print(resp.status_code)
30 print(resp.text)
31
```

requests\_task2.py 20:2 LF UTF-8 Python 0 files

Much better!

In this example you could modify the IP address and use either the PUT (replace configs in place) method *or* the PATCH (change only one part of the object) method. This is because there is only the IP address field to change! Think back to the Postman tasks where making a minor change required PATCH instead of PUT (unless you wanted to send the full configuration for the object).

## Build all the things!

Create a new text document called "requests\_task3.py" in the ubuntu (home) directory, we'll be working out of this file for the task.

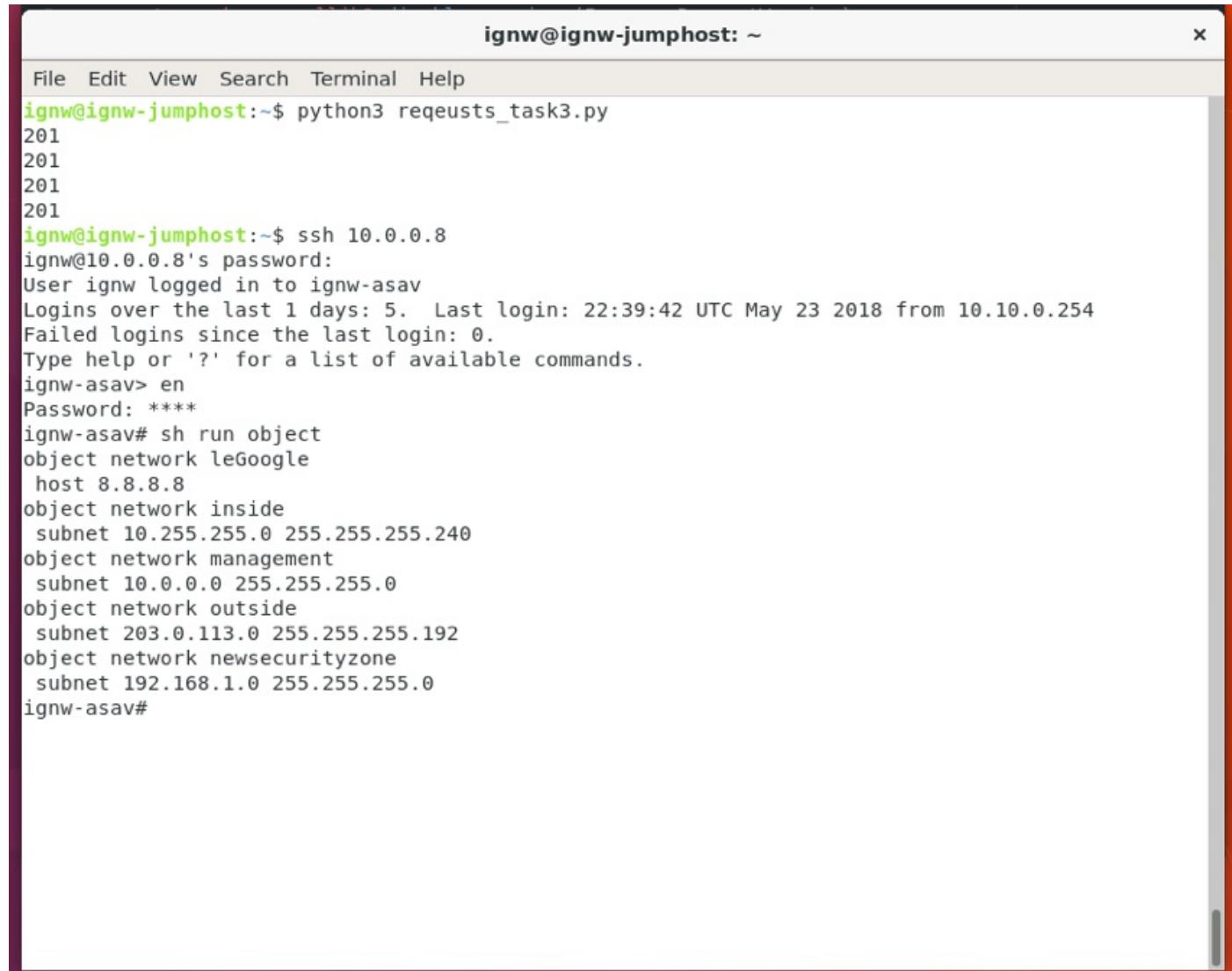
Taking what you've learned about requests and some basic Python logic, capture the IP address information from each interface (three in total -- management, outside, and dmz) on the ASA -- create an object group

named after the security zone of the interface, and assign the subnet of each interface to the corresponding object group.

**Tip:** Check out the "IPv4Network" network object type on the ASA API documentation.

**TIP:** Check out the "ipaddress" module -- specifically "ip\_network" and "strict" mode!

Your results should look similar to that shown below:



A terminal window titled "ignw@ignw-jumphost: ~". The window shows the output of several commands. It starts with a series of "201" responses to a "python3 requests\_task3.py" command. Then it shows an SSH session to 10.0.0.8, where the user logs in and sees their last login details. After logging in, they enter configuration mode ("en") and list network objects. The objects include "leGoogle" (host 8.8.8.8), "inside" (subnet 10.255.255.0/24), "management" (subnet 10.0.0.0/24), and "outside" (subnet 203.0.113.0/24). There is also a "newsecurityzone" object (subnet 192.168.1.0/24).

```
ignw@ignw-jumphost:~$ python3 requests_task3.py
201
201
201
201
ignw@ignw-jumphost:~$ ssh 10.0.0.8
ignw@10.0.0.8's password:
User ignw logged in to ignw-asav
Logins over the last 1 days: 5. Last login: 22:39:42 UTC May 23 2018 from 10.10.0.254
Failed logins since the last login: 0.
Type help or '?' for a list of available commands.
ignw-asav> en
Password: ****
ignw-asav# sh run object
object network leGoogle
 host 8.8.8.8
object network inside
 subnet 10.255.255.0 255.255.255.240
object network management
 subnet 10.0.0.0 255.255.255.0
object network outside
 subnet 203.0.113.0 255.255.255.192
object network newsecurityzone
 subnet 192.168.1.0 255.255.255.0
ignw-asav#
```

If you get stuck, ask your instructor for help. If you get really stuck, check out the example code to see if that will help you along!

## Kicking the tires on NETCONF & what the heck is YANG?

As you've learned APIs come in lots of different "flavors"! NETCONF is one of those flavors. NETCONF uses remote procedure calls via SSH to handle requests -- this is more or less a fancy way of saying that NETCONF transports requests over SSH to the remote device where the request is actually executed (be that a configuration or a query operation). The whole process uses XML encoding to properly frame the procedure.

NETCONF by itself is neat because it provides formally structured and properly encoded messaging format that is widely supported. This is a huge first step to help advance the networking world past the perils of screen scraping, but by itself is really just a structured input/output queue to a device. The data going to and from the device could in theory still be "messy". This is where YANG comes into play.

YANG - Yet Another Next Generation data modeling language -- as its name implies is a *modeling* language. What does that mean to us? In theory it means that we can be provided the entire *model* of an API in a structured data format. This could then be used by us to more easily understand the data structures that a device supports for configuration or query operations. It also means (in theory) that there can be standard data models that are supported across vendors (more on this later), as opposed to vendor-specific CLI syntax that we have today.

Rather than discuss it, let's try to poke our CSR router (which supports NETCONF and YANG) so we can see what this looks like.

In general the first step would be to enable "netconf-yang" on the router (this is supported on most, if not all, modern Cisco IOS-XE platforms). This configuration (literally "netconf-yang") has been done for you.

Open a terminal window on the RDP desktop, and enter the following:

```
ssh -p 830 -l ignw 10.0.0.5 -s netconf
```

A screenshot of a terminal window titled "ignw@ignw-jumphost: ~". The window has a standard OS X-style title bar with icons for File, Edit, View, Search, Terminal, and Help. The main pane shows a command-line interface with the following text:

```
ignw@ignw-jumphost:~$ ssh -p 830 -l ignw 10.0.0.5 -s netconf
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
7</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:RSVP-MIB?module=RSVP-MIB&revision=1998-08-25</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:SNMP-FRAMEWORK-MIB?module=SNMP-FRAMEWORK-MIB&revision=2002-10-14</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:SNMP-PROXY-MIB?module=SNMP-PROXY-MIB&revision=2002-10-14</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:SNMP-TARGET-MIB?module=SNMP-TARGET-MIB&revision=1998-08-04</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:SNMPv2-MIB?module=SNMPv2-MIB&revision=2002-10-16</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:SNMPv2-TC?module=SNMPv2-TC</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:SONET-MIB?module=SONET-MIB&revision=2003-08-11</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:TCP-MIB?module=TCP-MIB&revision=2005-02-18</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:TOKEN-RING-RMON-MIB?module=TOKEN-RING-RMON-MIB</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:TOKENRING-MIB?module=TOKENRING-MIB&revision=1994-10-23</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:TUNNEL-MIB?module=TUNNEL-MIB&revision=2005-05-16</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:UDP-MIB?module=UDP-MIB&revision=2005-05-20</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:VPN-TC-STD-MIB?module=VPN-TC-STD-MIB&revision=2005-11-15</capability>
<capability>urn:ietf:params:xml:ns:netconf:base:1.0?module=ietf-netconf&revision=2011-06-01</capability>
<capability>urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults?module=ietf-netconf-with-defaults&revision=2011-06-01</capability>
<capability>
    urn:ietf:params:netconf:capability:notification:1.1
</capability>
</capabilities>
<session-id>28</session-id></hello>]]>]]>
```

Wow, lots of stuff has happened! So what... did we do? Obviously we SSH'd to the IP of the CSR router, passed in the username ("ignw"), but what else? The "-p" option provides a port, in this case 830 -- which is the default port for NETCONF -- and finally we passed a "-s" argument of "netconf". The "-s" argument "May be used to request invocation of a subsystem on the remote system. Subsystems facilitate the use of SSH as a secure transport for other applications (e.g. sftp(1))." per the SSH man page. OK, so it looks like we've "invoked" the "netconf" subsystem.

What does that actually mean, and what is all of this output? Take a closer look at some of the giant wall of text that you received -- what do you notice about it?

```
ignw@ignw-jumphost:~
```

```
File Edit View Search Terminal Help
```

```
ignw@ignw-jumphost:~$ ssh -p 830 -l ignw 10.0.0.5 -s netconf
ignw@10.0.0.5's password:
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:base:1.1</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability>xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
<capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
<capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>
<capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit&also-supported=report-all-tagged</capability>
<capability>urn:ietf:params:netconf:capability:yang-library:1.0?revision=2016-06-21&module-set-id=88c694c75e847aba17e8ab19254ad090</capability>
<capability>http://tail-f.com/ns/netconf/actions/1.0</capability>
<capability>http://tail-f.com/ns/netconf/extensions</capability>
<capability>http://cisco.com/ns/cisco-xe-ietf-ip-deviation?module=cisco-xe-ietf-ip-deviation&revision=2016-08-10</capability>
<capability>http://cisco.com/ns/cisco-xe-ietf-ipv4-unicast-routing-deviation&revision=2015-09-11</capability>
<capability>http://cisco.com/ns/cisco-xe-ietf-ipv6-unicast-routing-deviation&revision=2015-09-11</capability>
<capability>http://cisco.com/ns/cisco-xe-ietf-ospf-deviation?module=cisco-xe-ietf-ospf-deviation&revision=2018-02-09</capability>
<capability>http://cisco.com/ns/cisco-xe-ietf-routing-deviation?module=cisco-xe-ietf-routing-deviation&revision=2016-07-09</capability>
<capability>http://cisco.com/ns/cisco-xe-openconfig-acl-deviation?module=cisco-xe-openconfig-acl-deviation&revision=2017-08-25</capability>
<capability>http://cisco.com/ns/mpls-static/devs?module=common-mpls-static-devs&revision=2015-09-11</capability>
<capability>http://cisco.com/ns/nvo/devs?module=nvo-devs&revision=2015-09-11</capability>
<capability>http://cisco.com/ns/yang/Cisco-IOS-XE-aaa?module=Cisco-IOS-XE-aaa&revision=2018-01-11</capability>
<capability>http://cisco.com/ns/yang/Cisco-IOS-XE-aaa-oper?module=Cisco-IOS-XE-aaa-oper&revision=2017-11-01</capability>
```

```

ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-diagnostics?module=Cisco-IOS-XE-diagnostics&revision=2017-02-07</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-dot1x?module=Cisco-IOS-XE-dot1x&revision=2017-11-27</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-eem?module=Cisco-IOS-XE-eem&revision=2017-11-20</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-efp-oper?module=Cisco-IOS-XE-efp-oper&revision=2017-02-07</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-eigrp?module=Cisco-IOS-XE-eigrp&revision=2017-09-21</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-environment-oper?module=Cisco-IOS-XE-environment-oper&revision=2017-11-27</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-eta?module=Cisco-IOS-XE-eta&revision=2017-11-27</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-ethernet?module=Cisco-IOS-XE-ethernet&revision=2018-02-02</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-ezpm?module=Cisco-IOS-XE-ezpm&revision=2017-11-27</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-features?module=Cisco-IOS-XE-features&revision=2017-02-07&features=virtual-template,punt-num,parameter-map,multilink,l2vpn,l2,ezpm,eth-enc,esmc,efp,crypto</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-fib-oper?module=Cisco-IOS-XE-fib-oper&revision=2017-07-04</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-flow?module=Cisco-IOS-XE-flow&revision=2018-01-04</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-flow-monitor-oper?module=Cisco-IOS-XE-flow-monitor-oper&revision=2017-11-30</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-http?module=Cisco-IOS-XE-http&revision=2018-01-24</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-icmp?module=Cisco-IOS-XE-icmp&revision=2017-11-27</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-igmp?module=Cisco-IOS-XE-igmp&revision=2017-11-27</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-interface-common?module=Cisco-IOS-XE-interface-common&revision=2017-11-27</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-interfaces-oper?module=Cisco-IOS-XE-interfaces-oper&revision=2017-10-10</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-ip-sla-oper?module=Cisco-IOS-XE-ip-sla-oper&revision=2017-09-25</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-ipv6-oper?module=Cisco-IOS-XE-ipv6-oper&revision=2017-11-01</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-isis?module=Cisco-IOS-XE-isis&revision=2017-11-27</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-iwanfabric?module=Cisco-IOS-XE-iwanfabric&revision=2017-11-27</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-l2vpn?module=Cisco-IOS-XE-l2vpn&revision=2017-11-27</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-l3vpn?module=Cisco-IOS-XE-l3vpn&revision=2017-02-07</capability>
<capability>http://cisco.com/ns.yang/Cisco-IOS-XE-lisp?module=Cisco-IOS-XE-lisp&revision=2017-11-27</capability>

```

One thing that is fairly obvious is that it is XML data, we know that because it tells us it is! The other thing we can see is that it is listing a LOT of "capabilities". With what you know about NETCONF this should probably start making some sense! NETCONF is running over SSH (we know that because we used SSH to make this happen), and we know that it uses XML encoding to negotiate server capabilities. The idea here is that NETCONF is making sure that the client and server are in sync with regards to what capabilities will be used during any interaction.

Now that we know that NETCONF on the CSR router is at the very least functioning, let's try to use Python to interact with it. Create another .py file, call this one ncc\_task1.py, save it in your home directory.

The ncclient module is a Python module for interacting with NETCONF devices. Getting started with it is very much like the other modules that we've used thus far -- we need to create a "connection" object, pass it some information about what we want to connect to and off we go!

You can check out the Readme docs [here](#) on the ncclient Github page. Setup your initial connection information as shown below:

The screenshot shows a code editor window titled "ncc\_task1.py -- ~ -- Atom". The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. The code in the editor is:

```
1 from ncclient import manager
2
3 m = manager.connect(host='10.0.0.5', port=830, username='ignw',
4                     password='ignw', device_params={'name': 'csr'})
5
6
7
8
```

Let's see if we can use Python to get a list of the "capabilities" of our CSR, just like how we did with SSH. Browse the Github page and the readthedocs page ([here](#)) to see if you see any mention of "capabilities" and how we might access them.

## Welcome

*ncclient* is a Python library for NETCONF clients. It aims to offer an intuitive API that sensibly maps the XML-encoded nature of NETCONF to Python constructs and idioms, and make writing network-management scripts easier. Other key features are:

- Supports all operations and capabilities defined in [RFC 4741](#).
- Request pipelining.
- Asynchronous RPC requests.
- Keeping XML out of the way unless really needed.
- Extensible. New transport mappings and capabilities/operations can be easily added.

The best way to introduce is through a simple code example:

```
from ncclient import manager

# use unencrypted keys from ssh-agent or ~/.ssh keys, and rely on known_hosts
with manager.connect_ssh("host", "username", "user") as m:
    assert(":url" in m.server_capabilities)
    with m.locked("running"):
        m.copy_config(source="running", target="file:///new_checkpoint.conf")
        m.copy_config(source="file:///old_checkpoint.conf", target="running")
```

That looks like it may be useful... let's print that out and see what we get:

The screenshot shows two windows: an Atom code editor and a terminal window.

The Atom code editor window has a tab labeled "ncc\_task1.py" and contains the following Python code:

```
1 from ncclient import manager
2
3 m = manager.connect(host='10.0.0.5', port=830, username='ignw',
4                     password='ignw', device_params={'name': 'csr'})
5
6 print(m.server_capabilities)
7
```

The terminal window has a title bar "ignw@ignw-jumphost: ~" and displays the command output:

```
ignw@ignw-jumphost:~$ python3 ncc_task1.py
<dict_keyiterator object at 0x7fed07822cc8>
ignw@ignw-jumphost:~$
```

Well... that is at least something! It seems like it is an iterator of some kind. What do we know about iterators? Could we maybe try to loop over it?

ncc\_task1.py — — Atom

File Edit View Selection Find Packages Help

```
ncc_task1.py
1 from ncclient import manager
2
3 m = manager.connect(host='10.0.0.5', port=830, username='ignw',
4                     password='ignw', device_params={'name': 'csr'})
5
6 print(m.server_capabilities)
7
8 for thing in m.server_capabilities:
9     print(thing)
```

ignw@ignw-jumphost: ~

File Edit View Search Terminal Help

```
urn:ietf:params:xml:ns:yang:smiv2:PerfHist-TC-MIB?module=PerfHist-TC-MIB&revision=1998-11-07
urn:ietf:params:xml:ns:yang:smiv2:Q-BRIDGE-MIB?module=Q-BRIDGE-MIB&revision=2006-01-09
urn:ietf:params:xml:ns:yang:smiv2:RFC-1212?module=RFC-1212
urn:ietf:params:xml:ns:yang:smiv2:RFC-1215?module=RFC-1215
urn:ietf:params:xml:ns:yang:smiv2:RFC1155-SMI?module=RFC1155-SMI
urn:ietf:params:xml:ns:yang:smiv2:RFC1213-MIB?module=RFC1213-MIB
urn:ietf:params:xml:ns:yang:smiv2:RFC1315-MIB?module=RFC1315-MIB
urn:ietf:params:xml:ns:yang:smiv2:RMON-MIB?module=RMON-MIB&revision=2000-05-11
urn:ietf:params:xml:ns:yang:smiv2:RMON2-MIB?module=RMON2-MIB&revision=1996-05-27
urn:ietf:params:xml:ns:yang:smiv2:RSVP-MIB?module=RSVP-MIB&revision=1998-08-25
urn:ietf:params:xml:ns:yang:smiv2:SNMP-FRAMEWORK-MIB?module=SNMP-FRAMEWORK-MIB&revision=2002-10-14
urn:ietf:params:xml:ns:yang:smiv2:SNMP-PROXY-MIB?module=SNMP-PROXY-MIB&revision=2002-10-14
urn:ietf:params:xml:ns:yang:smiv2:SNMP-TARGET-MIB?module=SNMP-TARGET-MIB&revision=1998-08-04
urn:ietf:params:xml:ns:yang:smiv2:SNMPv2-MIB?module=SNMPv2-MIB&revision=2002-10-16
urn:ietf:params:xml:ns:yang:smiv2:SNMPv2-TC?module=SNMPv2-TC
urn:ietf:params:xml:ns:yang:smiv2:SONET-MIB?module=SONET-MIB&revision=2003-08-11
urn:ietf:params:xml:ns:yang:smiv2:TCP-MIB?module=TCP-MIB&revision=2005-02-18
urn:ietf:params:xml:ns:yang:smiv2:TOKEN-RING-RMON-MIB?module=TOKEN-RING-RMON-MIB
urn:ietf:params:xml:ns:yang:smiv2:TOKENRING-MIB?module=TOKENRING-MIB&revision=1994-10-23
urn:ietf:params:xml:ns:yang:smiv2:TUNNEL-MIB?module=TUNNEL-MIB&revision=2005-05-16
urn:ietf:params:xml:ns:yang:smiv2:UDP-MIB?module=UDP-MIB&revision=2005-05-20
urn:ietf:params:xml:ns:yang:smiv2:VPN-TC-STD-MIB?module=VPN-TC-STD-MIB&revision=2005-11-15
urn:ietf:params:xml:ns:netconf:base:1.0?module=ietf-netconf&revision=2011-06-01
urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults?module=ietf-netconf-with-defaults&revision=2011-06-01

    urn:ietf:params:netconf:capability:notification:1.1

ignw@ignw-jumphost:~$
```

Interesting! It looks like this is the same data, but stripped of the XML structure. If you look through the list and compare it to before you'll see that the same capabilities are outlined here.

OK, so we are connected and clearly can get some information from the router -- but that all looks like "capability" information, what about actual configuration?

We can use "dir" as we did before to see what other kinds of attributes/methods are available on our connection object. This may give us a hint of how we can proceed:

ncc\_task1.py — — Atom

File Edit View Selection Find Packages Help

```

ncc_task1.py
1 from ncclient import manager
2
3 m = manager.connect(host='10.0.0.5', port=830, username='ignw',
4                     password='ignw', device_params={'name': 'csr'})
5
6 print(m.server_capabilities)
7
8 for thing in m.server_capabilities:
9     print(thing)
10
11 print(dir(m))
12

```

ignw@ignw-jumphost: ~

File Edit View Search Terminal Help

```

-14
urn:ietf:params:xml:ns.yang:smiv2:SNMP-PROXY-MIB?module=SNMP-PROXY-MIB&revision=2002-10-14
urn:ietf:params:xml:ns.yang:smiv2:SNMP-TARGET-MIB?module=SNMP-TARGET-MIB&revision=1998-08-04
urn:ietf:params:xml:ns.yang:smiv2:SNMPv2-MIB?module=SNMPv2-MIB&revision=2002-10-16
urn:ietf:params:xml:ns.yang:smiv2:SNMPv2-TC?module=SNMPv2-TC
urn:ietf:params:xml:ns.yang:smiv2:SONET-MIB?module=SONET-MIB&revision=2003-08-11
urn:ietf:params:xml:ns.yang:smiv2:TCP-MIB?module=TCP-MIB&revision=2005-02-18
urn:ietf:params:xml:ns.yang:smiv2:TOKEN-RING-RMON-MIB?module=TOKEN-RING-RMON-MIB
urn:ietf:params:xml:ns.yang:smiv2:TOKENRING-MIB?module=TOKENRING-MIB&revision=1994-10-23
urn:ietf:params:xml:ns.yang:smiv2:TUNNEL-MIB?module=TUNNEL-MIB&revision=2005-05-16
urn:ietf:params:xml:ns.yang:smiv2:UDP-MIB?module=UDP-MIB&revision=2005-05-20
urn:ietf:params:xml:ns.yang:smiv2:VPN-TC-STD-MIB?module=VPN-TC-STD-MIB&revision=2005-11-15
urn:ietf:params:xml:ns:netconf:base:1.0?module=ietf-netconf&revision=2011-06-01
urn:ietf:params:xml:ns.yang:ietf-netconf-with-defaults?module=ietf-netconf-with-defaults&revision=2011-06-01

urn:ietf:params:netconf:capability:notification:1.1

['_Manager_set_async_mode', '_Manager_set_raise_mode', '_Manager_set_timeout', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__enter__', '__eq__', '__exit__', '__format__', '__ge__', '__getattr__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '__async_mode', 'device_handler', '__raise_mode', 'session', 'timeout', 'async_mode', 'channel_id', 'channel_name', 'client_capabilities', 'close_session', 'commit', 'connected', 'copy_config', 'create_subscription', 'delete_config', 'discard_changes', 'dispatch', 'edit_config', 'execute', 'get', 'get_config', 'get_schema', 'kill_session', 'lock', 'locked', 'poweroff_machine', 'raise_mode', 'reboot_machine', 'scp', 'server_capabilities', 'session', 'session_id', 'take_notification', 'timeout', 'unlock', 'validate']
ignw@ignw-jumphost:~$ █

```

That looks promising! Try to capture the configuration of your device with the "get\_config" method.

**Note:** It is a pretty bad idea to use variable names like "thing"! Sometimes you may want to assign silly variables like this for testing purposes or to figure out what attributes are available to you like we've done here... it isn't a great idea though! We'll fix that moving forward.

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
lns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv6></interface><interface><name>GigabitEthernet2</name><description>This goes to the ASA</description><type xmlns:ianaif="urn:ietf:params:xml:ns:yang:iana-if-type">ianaif:ethernetCsmacd</type><enabled>true</enabled><ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"><address><ip>203.0.113.1</ip><netmask>255.255.255.192</netmask></address></ipv4><ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv6></interface><interface><name>Loopback0</name><description>I made this with Python</description><type xmlns:ianaif="urn:ietf:params:xml:ns:yang:iana-if-type">ianaif:softwareLoopback</type><enabled>true</enabled><ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"><address><ip>172.16.1.1</ip><netmask>255.255.255.255</netmask></address></ipv4><ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv6></interface><interface><name>Loopback1</name><description>IGNW was here</description><type xmlns:ianaif="urn:ietf:params:xml:ns:yang:iana-if-type">ianaif:softwareLoopback</type><enabled>true</enabled><ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"><address><ip>8.8.4.4</ip><netmask>255.255.255.255</netmask></address></ipv4><ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv6></interface></interfaces><nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"><enable-nacm>true</enable-nacm><read-default>deny</read-default><write-default>deny</write-default><exec-default>deny</exec-default><enable-external-groups>true</enable-external-groups><rule-list><name>admin</name><group>PRIV15</group><rule><name>permit-all</name><module-name>*</module-name><access-operations>*</access-operations><action>permit</action></rule></rule-list></nacm><routing xmlns="urn:ietf:params:xml:ns:yang:ietf-routing"><routing-instance><name>management</name><interfaces><interface>GigabitEthernet1</interface></interfaces><routing-protocols><routing-protocol><type>static</type><name>1</name><static-routes><ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing"><route><destination-prefix>0.0.0.0/0</destination-prefix><next-hop><next-hop-address>10.0.0.1</next-hop-address></next-hop></route></ipv4></static-routes></routing-protocol></routing-protocols></routing-instance><routing-instance><name>default</name><description>default-vrf [read-only]</description><interfaces></interfaces><routing-protocols><routing-protocol><type>static</type><name>1</name><static-routes><ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing"><route><destination-prefix>10.255.255.2/32</destination-prefix><next-hop><next-hop-address>203.0.113.2</next-hop-address></next-hop></route></ipv4></static-routes></routing-protocol></routing-protocols></routing-instance></routing></data></rpc-reply>
ignw@ignw-jumphost:~$
```

Were you able to capture the config from the router? If not, check the examples on the Github page!

The output may look terrible in the terminal, but as you can see it is XML formatted. This may not be the easiest format on the eyes, but to a machine this is much better than the "normal" CLI syntax that we're used to dealing with.

**Tip:** Copy the output and search for "XML Pretty Print" on the web, paste the config in and the output is a little easier to read!

```
<ip>
  <forward-protocol>
    <protocol>nd</protocol>
  </forward-protocol>
  <route>
    <ip-route-interface-forwarding-list>
      <prefix>0.0.0.0</prefix>
      <mask>0.0.0.0</mask>
      <fwd-list>
        <fwd>GigabitEthernet1</fwd>
        <interface-next-hop>
          <ip-address>10.1.2.1</ip-address>
        </interface-next-hop>
      </fwd-list>
    </ip-route-interface-forwarding-list>
    <ip-route-interface-forwarding-list>
      <prefix>10.1.254.0</prefix>
      <mask>255.255.255.0</mask>
      <fwd-list>
        <fwd>10.1.253.6</fwd>
      </fwd-list>
    </ip-route-interface-forwarding-list>
  </route>
```

Now that we can get back *structured* data, it should be fairly easy to pull out specific data that we would like to see, or at least use Python to "Pretty Print" it for us.

Python has built in support for dealing with XML data -- including pretty'ing it -- so we'll use that.

ncc\_task1.py -- ~ -- Atom

```

File Edit View Selection Find Packages Help
ncc_task1.py
1 from ncclient import manager
2 import xml.dom.minidom
3
4 m = manager.connect(host='10.0.0.5', port=830, username='ignw',
5                     password='ignw', device_params={'name': 'csr'})
6
7 # print(m.server_capabilities)
8
9
10 for cap in m.server_capabilities:
11     print(cap)
12
13 # print(dir(m))
14
15 csr_config = (m.get_config(source='running'))
16
17 # print(csr_config.xml)
18
19 print(xml.dom.minidom.parseString(csr_config.xml).toprettyxml())

```

ignw@ignw-jumphost: ~

```

File Edit View Search Terminal Help
</routing-instance>
<routing-instance>
    <name>default</name>
    <description>default-vrf [read-only]</description>
    <interfaces/>
    <routing-protocols>
        <routing-protocol>
            <type>static</type>
            <name>l</name>
            <static-routes>
                <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip:ipv4">
                    <route>
                        <destination-prefix>10.255.255.2/32</destination-prefix>
                        <next-hop>
                            <next-hop-address>203.0.113.2</next-hop-address>
                        </next-hop>
                    </route>
                </ipv4>
            </static-routes>
        </routing-protocol>
    </routing-protocols>
</routing-instance>

```

While it may not look so pretty on the small terminal window as shown here, the XML is now formatted much nicer! We were able to do this with the built in `xml` module and the "parseString" method and finally running the "toprettyxml()" method on it.

If you take a look through the now *beautiful* XML -- do you notice anything strange? Look again... You'll see that the interface configs show up several times! Why do you think that is?

Recall back to the "capabilities" we looked at, do you remember that giant list? You may have noticed "ietf", "cisco", and "openconfig" items in there. That is because there are many different YANG models that define state of devices.

In general, the principle idea for YANG would be that the IETF or OpenConfig group create universally accepted YANG models to... model... state of devices. In reality, the IETF, OpenConfig, and individual vendors are all creating models, that is why we have IETF and OpenConfig models on our CSR, as well as the "native" Cisco models. So we are basically seeing the output of a few different types of configuration models.

## RESTCONF, back to requests

Now that you've gotten a tiny taste of NETCONF its time to briefly check out RESTCONF. RESTCONF is similar in many ways to NETCONF -- it too is a *transport/communications* layer that more or less sits on top of the YANG models that carry the actual state data of a device.

As you may have guessed from the name, RESTCONF is intended to bring RESTful API type capability to the NETCONF/YANG world. As with "normal" RESTful APIs RESTCONF supports both XML and JSON data formats and operates via HTTP methods.

Superficially NETCONF and RESTCONF are more or less interchangeable, and for the purposes of this lab that is probably a relatively fair comparison, though under the covers there are some important differences!

As with NETCONF, the configuration to enable RESTCONF on the router is very simple and has already been configured for you. The following configurations are all that is required to get things up and rolling:

```
restconf
ip http secure-server
```

You can read about further details for the setup [here](#). This is also a good link to have handy for getting things rolling with your first RESTCONF request.

Create a new text document called "restconf\_task1.py" in the ubuntu (home) directory, we'll be working out of this file for the task.

Because RESTCONF is an HTTP based client, we will be once more using the requests library. Steal from your previous scripts to import requests and silence the insecure SSL warnings, as our CSR also has a self-signed certificate.

```
restconf_task1.py •
```

```
1 import requests
2 from requests.packages.urllib3.exceptions import InsecureRequestWarning
3
4
5 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
6
7
8
9
```

Now that you know a bit about requests, you can probably guess some of the data that we'll need to begin crafting our request: type of authentication, URL/endpoint to post to, and any required headers would be a good start.

As for authentication, it is a safe bet to assume that this will be "Basic Authentication" just like it was with the ASA v, so we'll operate under that assumption unless proven otherwise!

As for headers, take a look at the latest [IOS-XE programmability guide](#) -- specifically the REST API Resource section. We can see that it specifically outlines "application/yang-data+json" as the "accept" and "content-type" values. Finally, it also outlines "/restconf" as the root resource.

With all of that in mind, add username, password, url, and headers variables to your script.

```
restconf_task1.py — ~ — Atom
```

```
File Edit View Selection Find Packages Help
```

```
restconf_task1.py
```

```
1 import requests
2 from requests.packages.urllib3.exceptions import InsecureRequestWarning
3
4
5 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
6
7 username = 'ignw'
8 password = 'ignw'
9 url = "https://10.0.0.5/restconf"
10 headers = {'content-type': 'application/yang-data+json',
11             'accept': 'application/yang-data+json'}
```

Notice that the headers argument is a dictionary containing the key/value pairs that were outlined in the documentation.

With the basics, craft your first "GET" request to the root URL ("restconf"), and print out the status and text of your response to verify that you're on the right track.

```

restconf_task1.py — ~ — Atom
File Edit View Selection Find Packages Help
restconf_task1.py
3
4
5 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
6
7 username = 'ignw'
8 password = 'ignw'
9 url = "https://10.0.0.5/restconf"
10 headers = {'content-type': 'application/yang-data+json',
11             'accept': 'application/yang-data+json'}
12
13 resp = requests.get(url, auth=(username, password), headers=headers,
14                       verify=False)
15 print(resp.status_code)
16 print(resp.text)
17
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 restconf_task1.py
200
{"restconf": {"data": {}, "operations": {}, "yang-library-version": "2016-06-21"}}
ignw@ignw-jumphost:~$
```

That's a good start! 200 is of course the "OK" status code, and it looks like we get at the very least some information about the "yang-library-version". Recall that RESTCONF is just the transport and we will actually be dealing with YANG models when getting or sending state data to the device.

Do you recall the different "capabilities" that were listed when querying the device with NETCONF? Take another look at the IOS-XE REST API documentation linked above. Look under the "Methods" section.

## Methods

The content query parameter controls how descendant nodes of the requested data nodes are processed in the reply:

- Must be supported by the server.
- If not present in URI, the default value is: all. Allowed only for GET/HEAD method.

A "400 Bad Request" status-line is returned if used for other methods or resource types.

Examples for allowed values are:

1. <https://10.85.116.59:443/restconf/data/Cisco-IOS-XE-native:native?content=config>
2. <https://10.85.116.59:443/restconf/data/Cisco-IOS-XE-native:native?content=nonconfig>

Does the highlighted endpoint look familiar? It should! "Cisco-IOS-XE-native" is one of the flavors of YANG models that showed up in the capabilities (along with lots of sub-options), as well as in the configuration that you captured. This also gives us the rest of the URL that we will need to fill out to capture the running config. Try to snag the configuration with RESTCONF this time.

restconf\_task1.py — ~ — Atom

```
File Edit View Selection Find Packages Help
restconf_task1.py

1 import requests
2 from requests.packages.urllib3.exceptions import InsecureRequestWarning
3
4
5 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
6
7 username = 'ignw'
8 password = 'ignw'
9 url = "https://10.0.0.5/restconf"
10 headers = {'content-type': 'application/yang-data+json',
11             'accept': 'application/yang-data+json'}
12
13 # resp = requests.get(url, auth=(username, password), headers=headers,
14 #                       verify=False)
15 # print(resp.status_code)
16 # print(resp.text)
17
18 resp = requests.get(f'{url}/data/Cisco-IOS-XE-native:native?content=config',
19                      auth=(username, password), headers=headers,
20                      verify=False)
21 print(resp.status_code)
22 print(resp.text)
23
```

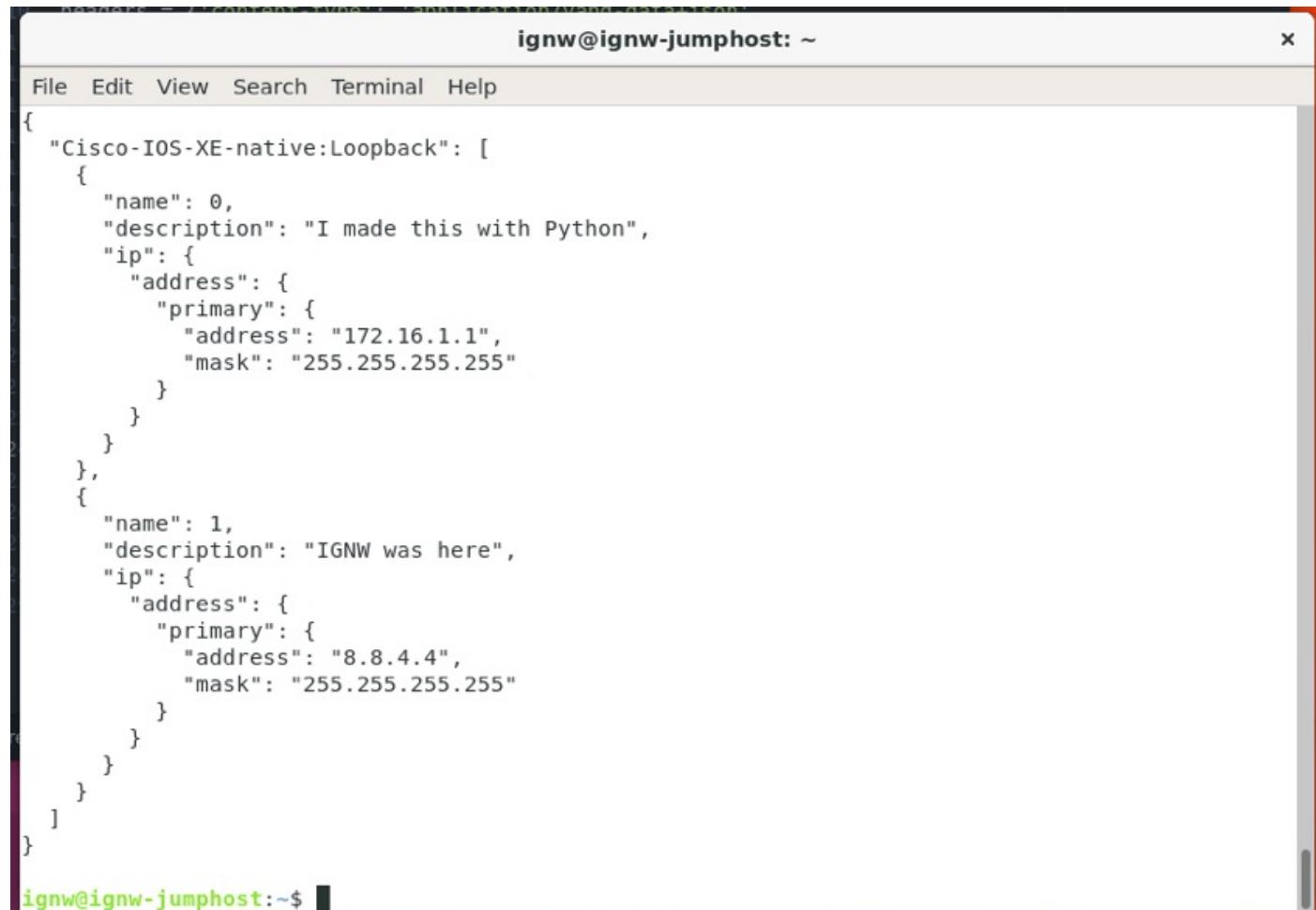
ignw@ignw-jumphost: ~

```
File Edit View Search Terminal Help
200
{
  "Cisco-IOS-XE-native:native": {
    "version": "16.8",
    "boot-start-marker": [null],
    "boot-end-marker": [null],
    "service": {
      "timestamps": {
        "debug": {
          "datetime": {
            "msec": {}
          }
        },
        "log": {
          "datetime": {
            "msec": [null]
          }
        }
      }
    },
    "platform": {
      "Cisco-IOS-XE-platform-console": {
        "name": "Console"
      }
    }
  }
}
```

Notice anything different this time? The returned data is in JSON! With RESTCONF we could request XML

responses by changing our content-type and accept header values, but JSON will work just fine.

With what you've learned about APIs, try to capture the configuration from just the loopbacks on the router.



The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~". The window has a standard OS X-style title bar with icons for close, minimize, and zoom. Below the title bar is a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area of the terminal displays a JSON object representing network interface configurations. The JSON structure is as follows:

```
{  
  "Cisco-IOS-XE-native:Loopback": [  
    {  
      "name": 0,  
      "description": "I made this with Python",  
      "ip": {  
        "address": {  
          "primary": {  
            "address": "172.16.1.1",  
            "mask": "255.255.255.255"  
          }  
        }  
      }  
    },  
    {  
      "name": 1,  
      "description": "IGNW was here",  
      "ip": {  
        "address": {  
          "primary": {  
            "address": "8.8.4.4",  
            "mask": "255.255.255.255"  
          }  
        }  
      }  
    }  
  ]  
}
```

At the bottom of the terminal window, the prompt "ignw@ignw-jumphost:~\$" is visible, indicating the user is ready for the next command.

The screenshot shows a code editor window titled "restconf\_task1.py — ~ — Atom". The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. The code itself is a Python script named "restconf\_task1.py". It uses the "requests" module to make REST API calls to a Cisco router. The script attempts to disable SSL warnings, set authentication (username: ignw, password: ignw), and specify JSON content type and accept headers. It then tries to retrieve configuration data and interface information. The code is numbered from 1 to 29. The status bar at the bottom indicates the file is saved at line 6, encoding is UTF-8, and it's a Python file.

```
restconf_task1.py — ~ — Atom
File Edit View Selection Find Packages Help
restconf_task1.py
1 import requests
2 from requests.packages.urllib3.exceptions import InsecureRequestWarning
3
4
5 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
6
7 username = 'ignw'
8 password = 'ignw'
9 url = "https://10.0.0.5/restconf"
10 headers = {'content-type': 'application/yang-data+json',
11             'accept': 'application/yang-data+json'}
12
13 # resp = requests.get(url, auth=(username, password), headers=headers,
14 #                       verify=False)
15 # print(resp.status_code)
16 # print(resp.text)
17
18 # resp = requests.get(f'{url}/data/Cisco-IOS-native:native?content=config',
19 #                      auth=(username, password), headers=headers,
20 #                      verify=False)
21 # print(resp.status_code)
22 # print(resp.text)
23
24 resp = requests.get(f'{url}/data/Cisco-IOS-XE-native:interface/Loopback',
25                      auth=(username, password), headers=headers,
26                      verify=False)
27 print(resp.status_code)
28 print(resp.text)
29
```

## RESTCONF POST a new Loopback

Create a new text document called "restconf\_task1.py" in the ubuntu (home) directory, we'll be working out of this file for the task.

Using RESTCONF, create a new loopback, loopback2 on your CSR router. Assign it the IP address "172.16.1.2 255.255.255.255".

The output from the previous task, the requests module, and maybe the json module are all that you should need! The screen shot below gives away a bit of one of the ways to go about doing this!

The screenshot shows a Linux desktop environment with two windows open. On the left is a code editor window titled "restconf\_task2.py" in Atom. The code is a Python script for interacting with a Cisco device via RESTCONF. It imports json and requests, disables InsecureRequestWarning, sets up basic auth, and sends a POST request to create a new interface (Loopback 2) with a specific IP configuration. The right window is a terminal window titled "ignw@ignw-jumphost: ~". It shows the command "python3 restconf\_task2.py" being run, followed by the output "201", indicating success.

```
restconf_task2.py -- Atom
File Edit View Selection Find Packages Help
restconf_task2.py
1 import json
2 import requests
3 from requests.packages.urllib3.exceptions import InsecureRequestWarning
4
5
6 requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
7
8 username = 'ignw'
9 password = 'ignw'
10 url = "https://10.0.0.5/restconf"
11 headers = {'content-type': 'application/yang-data+json',
12             'accept': 'application/yang-data+json'}
13 endpoint = '/data/Cisco-IOS-XE-native:native/interface/'
14 data = {
15     'Cisco-IOS-XE-native:Loopback': {
16         'name': 2,
17         'description': 'RESTCONNFFFF WHY IS IT YELLING!!',
18         'ip': {
19             'address': {
20                 'primary': {
21                     'address': '172.16.1.2',
22                     'mask': '255.255.255.255'
23                 }
24             }
25         }
26     }
27 }
28 resp = requests.post(f'{url}/{endpoint}',
29                       auth=(username, password), headers=headers,
30                       verify=False, data=json.dumps(data))
31 print(resp.status_code)
32 print(resp.text)
restconf_task2.py 1:1
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ python3 restconf_task2.py
201
ignw@ignw-jumphost:~$
```

# Ansible for Network Engineer Labs

## Overview and Objectives

In this lab you will learn the basics of Ansible, focusing on basic networking configuration tasks.

Objectives:

- Create Ansible Playbooks
- Create Ansible inventory, group and host vars
- Run multiple Playbooks to learn about Ansible!

## Gathering Facts

As you've learned Ansible can be a powerful tool! The lab environment that you have been using for this course is provisioned using a combination of Packer, Terraform, and Ansible for many of the systems/device-level configurations.

In this section we'll start out by getting right into using Ansible to gather information about our environment, your inventory files and Ansible configuration files are already setup for you (for now!).

In this lab we'll start by using the built in "gather\_facts" -- per the Ansible documentation: "Facts are simply things that are discovered about remote nodes". That seems like a great place to start!

As you've learned, Ansible Playbooks are composed of Tasks, and are eventually executed against an inventory file or a subset of hosts in your inventory. Using the editor of your choice, create a new YAML file called "first\_playbook.yaml" in the ~/ansible\_net\_eng\_pt1 directory (this directory has been created for you and already contains some goodies).

The screenshot shows a Linux desktop environment with a dark theme. In the foreground, there is a terminal window titled "ignw@ignw-jumphost: ~/ansible\_net\_eng\_pt1". It displays the following command-line session:

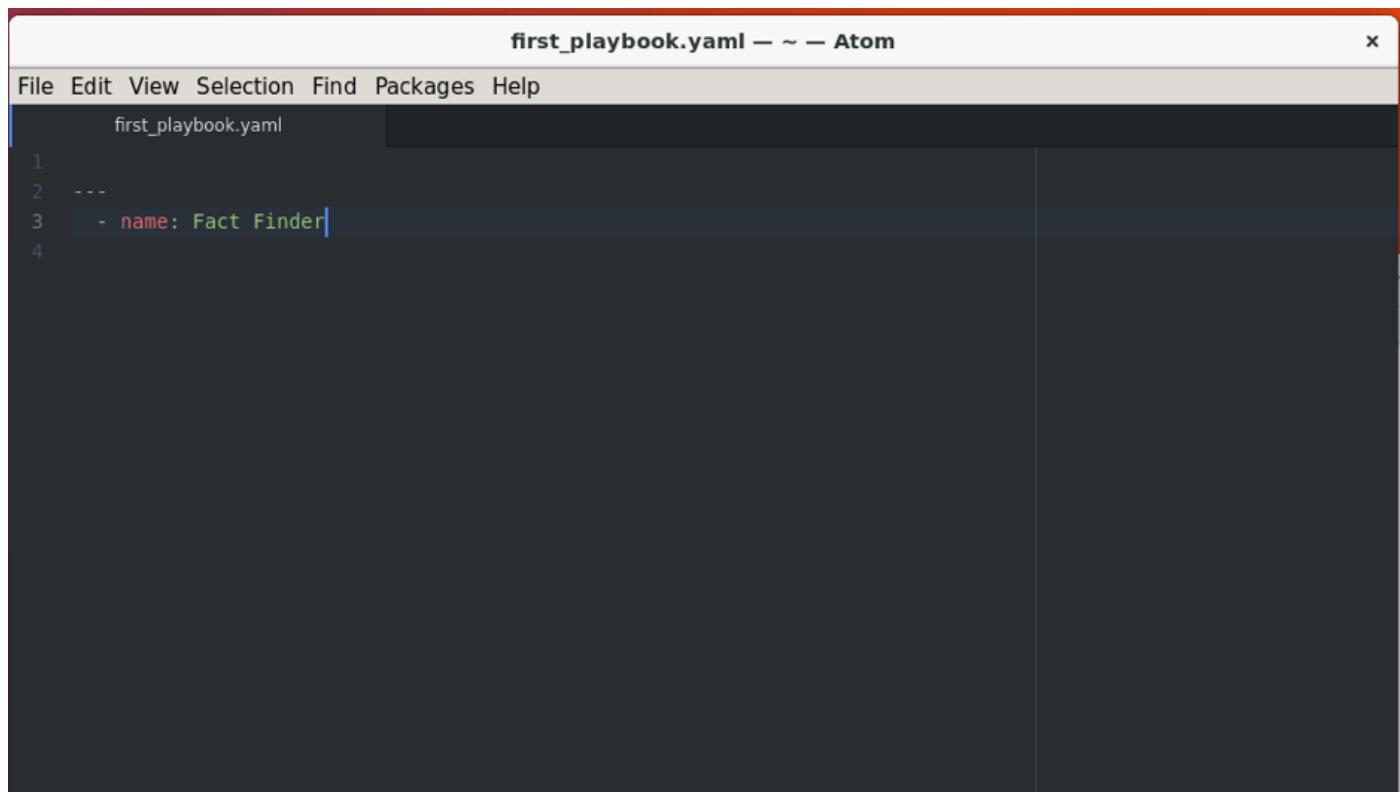
```
ignw@ignw-jumphost:~/ansible_net_eng_pt1$ pwd
/home/ignw/ansible_net_eng_pt1
ignw@ignw-jumphost:~/ansible_net_eng_pt1$ ls
ansible.cfg first_playbook.yaml inventory
ignw@ignw-jumphost:~/ansible_net_eng_pt1$
```

Below the terminal window, the status bar shows the path "ansible\_net\_eng\_pt1/first\_playbook.yaml" and the line number "1:1".

In the background, there is an Atom editor window titled "first\_playbook.yaml -- ~ -- Atom". The file content is as follows:

```
1
2
```

YAML syntax dictates that three dashes indicate the start of a file, so let's get that out of the way. We also know that a Playbook is made up of Plays, and Plays are documented as list elements which happen to be indicated by a dash as well. Each Play and Task can have a name element, so let's call our first Play "Fact Finder"



```
first_playbook.yaml — ~ — Atom
File Edit View Selection Find Packages Help
first_playbook.yaml
1
2 ---
3 - name: Fact Finder
4
```

The next thing we need to do for our Play is to tell it which host or hosts we want to run this Task against. Your inventory file has been created for you and contains a group called "csr" which contains the CSR router as a host. Let's just start with this group for now.

Now that our Play is created, we can create our first Tasks. For now we will use the built in module "debug" to print out a message; our message will be printing out all of the "ansible\_facts" data that Ansible gathers. In order to do this we need to run the setup module and "register" (basically assign) the output of that to a variable "ansible\_facts" (this could be an arbitrary name, though we'll call it "ansible\_facts" because that's what it is!).

Finally, we'll create another Task called "Print Ansible Facts", and use the debug module to print a message containing our previously registered variable

first\_playbook.yaml — ~ — Atom

File Edit View Selection Find Packages Help

```
first_playbook.yaml
1
2 ---
3   - name: Fact Finder
4     hosts: csr
5     tasks:
6       - name: Register Ansible Facts
7         setup:
8           register: ansible_facts
9       - name: Print Ansible Facts
10        debug:
11          msg: "{{ ansible_facts }}"
12
```

ansible\_net\_eng\_pt1/first\_playbook.yaml 11:34 LF UTF-8 YAML 0 files

If your playbook looks similar to that above you're ready to try to run it! Use the "ansible-playbook" command and pass it your Playbook's name:

```
ansible-playbook first_playbook.yaml
```

Does your Playbook run? What does it output? Does it look like it has anything at all to do with a CSR router? Probably not! Recall from the lecture that Ansible is an agent-less utility -- what does that mean? Simply that there are no agents installed and that Ansible uses SSH to connect to a remote device to execute (Python) commands. Python is only available natively on some network operating systems so that entire work-flow doesn't really work for network devices such as our CSR. To combat this Ansible has a configuration option (that has been configured for you at this point) called "ansible\_connection" which in our case is set to "local" -- this means that Ansible will execute tasks on the local device as opposed to the host or hosts specified in your Play. So in this case, Ansible is actually gathering facts from our local host, hence the strange (non-router) data!

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
/home/ignw/ansible_net_eng_pt1
ignw@ignw-jumphost:~/ansible_net_eng_pt1$ ls
ansible.cfg  first_playbook.yaml  inventory
ignw@ignw-jumphost:~/ansible_net_eng_pt1$ ansible-playbook first_playbook.yaml

PLAY [Fact Finder] ****
TASK [Gathering Facts] ****
ok: [ignw-csr]

TASK [Register Ansible Facts] ****
ok: [ignw-csr]

TASK [Print Ansible Facts] ****
ok: [ignw-csr] => {
  "msg": {
    "ansible_facts": {
      "ansible_all_ipv4_addresses": [
        "10.10.0.254"
      ],
      "ansible_all_ipv6_addresses": [
        "fe80::250:56ff:fe93:16f3"
      ],
      "ansible_apparmor": {
        "status": "enabled"
      },
      "ansible_architecture": "x86_64",
      "ansible_bios_date": "04/05/2016",
      "ansible_bios_version": "6.00",
      "ansible_cmdline": {
        "BOOT_IMAGE": "/boot/vmlinuz-4.13.0-43-generic",
        "quiet": true,
        "ro": true,
        "root": "/dev/mapper/ignw--jumphost--vg-root"
      },
      "ansible_date_time": {
        "date": "2018-05-24",
        "day": "24",
        "month": "May",
        "year": "2018"
      }
    }
  }
}
```

It is important to note that this is a *network* specific "problem" with Ansible -- if our target host(s) was a Linux server, and our `ansible_connection` argument was *not* set to local, Ansible would have properly gathered facts about the remote system.

Since we are dealing with a router at the moment, we need to figure out a way that we can capture "ansible\_facts" type data about our device. There is an Ansible module specifically for this exact purpose! The `ios-facts` will do just that for us and is even included in the "base" Ansible installation!

Since we know we don't need to "gather\_facts" anymore as it isn't doing what we were hoping, set the "gather\_facts" argument to "no", then create a new Task named "Gather IOS Facts". Under your Gather IOS Facts task, run the "ios\_facts" module as pictured below:

first\_playbook.yaml — ~ — Atom

File Edit View Selection Find Packages Help

```
first_playbook.yaml

1
2 ---
3   - name: Fact Finder
4     gather_facts: no
5     hosts: csr
6     tasks:
7       - name: Register Ansible Facts
8         setup:
9           register: ansible_facts
10      - name: Print Ansible Facts
11        debug:
12          msg: "{{ ansible_facts }}"
13      - name: Gather IOS Facts
14        ios_facts:
15          register: ios_facts_output
16
```

ansible\_net\_eng\_pt1/first\_playbook.yaml 16:1 LF UTF-8 YAML 0 files

Run your playbook once more. What happens?

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
l5IY/n9bQjdyo4N0whZs07JDMh3PDUmZbMzPwI8bQiQ5H2Bw4pncZRHuLWU0EyualoBhicBal+Z0st6lZoKhKedJzCmbgjhHA3b02irTzTbh
jdhEbsJ4rFr+GTZpMKfc3gw/5yQ0akYUAjU4DRmENbn7PjmTpEjq00vG+C1XJdRhZQKY82z0z5/5W4AY0Ta6hfo9dtgJP3MS",
    "ansible_swapfree_mb": 8191,
    "ansible_swaptotal_mb": 8191,
    "ansible_system": "Linux",
    "ansible_system_capabilities": [
        ""
    ],
    "ansible_system_capabilities_enforced": "True",
    "ansible_system_vendor": "VMware, Inc.",
    "ansible_uptime_seconds": 57404,
    "ansible_user_dir": "/home/ignw",
    "ansible_user_gecos": "ignw,,",
    "ansible_user_gid": 1000,
    "ansible_user_id": "ignw",
    "ansible_user_shell": "/bin/bash",
    "ansible_user_uid": 1000,
    "ansible_userspace_architecture": "x86_64",
    "ansible_userspace_bits": "64",
    "ansible_virtualization_role": "guest",
    "ansible_virtualization_type": "VMware",
    "gather_subset": [
        "all"
    ],
    "module_setup": true
},
"changed": false,
"failed": false
}
}

TASK [Gather IOS Facts] *****
fatal: [ignw-csr]: FAILED! => {"msg": "Authentication failed."}

PLAY RECAP *****
ignw-csr : ok=2    changed=0    unreachable=0    failed=1

ignw@ignw-jumphost:~/ansible_net_eng_pt1$
```

Red is probably bad! What does the error say?

We haven't yet told Ansible what credentials to use, so that has got to be our issue -- how could it authenticate if it doesn't have creds! In this section we have already stored the credentials in the inventory file, so you just need to "pass" them to the ios-facts module so that it knows how to connect! Looking at the `ios_facts` documentation we can see that the module would like to receive the login information via a dictionary called "provider": [IOS Facts Documentation](#).

Syntax in Ansible is a little different than in native Python, but the concept is the same, create your "credentials" dictionary as shown below:

first\_playbook.yaml — ~ — Atom

File Edit View Selection Find Packages Help

```
first_playbook.yaml
1
2 ---
3   - name: Fact Finder
4     gather_facts: no
5     hosts: csr
6     tasks:
7       - name: Register Ansible Facts
8         setup:
9           register: ansible_facts
10      - name: Print Ansible Facts
11        debug:
12          msg: "{{ ansible_facts }}"
13      - name: Gather IOS Facts
14        vars:
15          credentials:
16            host: "{{ ansible_host }}"
17            username: "{{ username }}"
18            password: "{{ password }}"
19          ios_facts:
20            provider: "{{ credentials }}"
21            register: ios_facts_output
22
```

ansible\_net\_eng\_pt1/first\_playbook.yaml 20:37 LF UTF-8 YAML 0 files

Re-run your script to see what happens, hopefully things are looking better!

```

ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
l5IY/n9b0jdyo4N0whZs07JDMh3PDUmZbMzPwI8bQi95H2Bw4pncZRHuLWU0EyualoBhicBal+Z0st6lZoKhKedJzCmbgjhHA3b02irTzTbh
jdhEbsJ4rFr+GTZpMKfc3gw/5yQ0akYUAjU4DRmENbn7PjmTpEjq00vG+C1XJdRhZQKY82z0z5/5W4AY0Ta6hfo9dtgJP3MS",
    "ansible_swapfree_mb": 8191,
    "ansible_swaptotal_mb": 8191,
    "ansible_system": "Linux",
    "ansible_system_capabilities": [
        ""
    ],
    "ansible_system_capabilities_enforced": "True",
    "ansible_system_vendor": "VMware, Inc.",
    "ansible_uptime_seconds": 58129,
    "ansible_user_dir": "/home/ignw",
    "ansible_user_gecos": "ignw,,",
    "ansible_user_gid": 1000,
    "ansible_user_id": "ignw",
    "ansible_user_shell": "/bin/bash",
    "ansible_user_uid": 1000,
    "ansible_userspace_architecture": "x86_64",
    "ansible_userspace_bits": "64",
    "ansible_virtualization_role": "guest",
    "ansible_virtualization_type": "VMware",
    "gather_subset": [
        "all"
    ],
    "module_setup": true
},
"changed": false,
"failed": false
}
}

TASK [Gather IOS Facts] *****
ok: [ignw-csr]

PLAY RECAP *****
ignw-csr : ok=3    changed=0    unreachable=0    failed=0
ignw@ignw-jumphost:~/ansible_net_eng_pt1$ 

```

Now that we've got the script running, let's try to print out the "ios\_facts" to see if we get any useful data there. Just like the previous task where we had to "register" the output of "gather\_facts" we must do the same with our ios\_facts task. Register the output of the ios\_facts Task to a new variable called "ios\_facts\_output", and add a Task to print this variable.

first\_playbook.yaml — ~ — Atom

File Edit View Selection Find Packages Help

```
1 first_playbook.yaml
2 ---
3   - name: Fact Finder
4     gather_facts: no
5     hosts: csr
6     tasks:
7       - name: Register Ansible Facts
8         setup:
9           register: ansible_facts
10      - name: Print Ansible Facts
11        debug:
12          msg: "{{ ansible_facts }}"
13      - name: Gather IOS Facts
14        vars:
15          credentials:
16            host: "{{ ansible_host }}"
17            username: "{{ username }}"
18            password: "{{ password }}"
19          ios_facts:
20            provider: "{{ credentials }}"
21            register: ios_facts_output
22      - name: Print IOS Facts
23        debug:
24          msg: "{{ ios_facts_output }}"
25
```

ansible\_net\_eng\_pt1/first\_playbook.yaml 25:1 LF UTF-8 YAML 0 files

Run your playbook one more time -- does it work? Do you get data that you would expect to see from the CSR?

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
    "duplex": null,
    "ipv4": [
        {
            "address": "172.16.1.2",
            "subnet": "32"
        }
    ],
    "lineprotocol": "up",
    "macaddress": "172.16.1.2/32",
    "mediatype": null,
    "mtu": 1514,
    "operstatus": "up",
    "type": null
},
{
    "ansible_net_memfree_mb": 2061065,
    "ansible_net_memtotal_mb": 2396128,
    "ansible_net_model": "CSR1000V",
    "ansible_net_neighbors": {
        "null": [
            {
                "host": null,
                "port": null
            }
        ]
    },
    "ansible_net_serialnum": "9E42JNXTJAN",
    "ansible_net_version": "16.08.01a"
},
"changed": false,
"failed": false
}
}

PLAY RECAP ****
ignw-csr : ok=4      changed=0      unreachable=0      failed=0
ignw@ignw-jumphost:~/ansible_net_eng_pt1$
```

## Conditionals

In this lab we'll learn a little about conditional statements in Ansible -- the logic is pretty much the same as native Python, but of course the syntax is a little bit different.

Create a new Playbook called "ios\_conditional.yaml" and save it in the ansible\_net\_eng\_pt1 directory.

In this new Playbook create a Task or Tasks to gather and print the IOS facts as we did in the previous task. Ensure that you don't bother with "gathering facts" as we now know that this would only get us facts about our local system as Ansible is currently configured. Ensure that your Play runs against the "csr" host group. Once you think you're ready, run your Playbook to see if it works!

ios\_conditional.yaml — ~ — Atom

File Edit View Selection Find Packages Help

```
1 ---  
2 - name: IOS Conditional Fact Checker  
3   gather_facts: no  
4   hosts: csr  
5   tasks:  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20
```

Imagine now that we had a group of hundreds of routers (virtual or otherwise!) or systems and we wanted to generate a report that printed the serial number and hostname of all routers (or systems) running a specific version of code. Look back through the facts that the `ios_facts` module gathered -- do you see any facts that may help us with this task?

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
    "duplex": null,
    "ipv4": [
        {
            "address": "172.16.1.2",
            "subnet": "32"
        }
    ],
    "lineprotocol": "up",
    "macaddress": "172.16.1.2/32",
    "mediatype": null,
    "mtu": 1514,
    "operstatus": "up",
    "type": null
},
"ansible_net_memfree_mb": 2061065,
"ansible_net_memtotal_mb": 2396128,
"ansible_net_model": "CSR1000V",
"ansible_net_neighbors": {
    "null": [
        {
            "host": null,
            "port": null
        }
    ]
},
"ansible_net_serialnum": "9E42JNXTJAN",
"ansible_net_version": "16.08.01a"
},
"changed": false,
"failed": false
}
}

PLAY RECAP ****
ignw-csr : ok=4      changed=0      unreachable=0      failed=0
ignw@ignw-jumphost:~/ansible_net_eng_pt1$
```

Add a Task to your Play to print out the serial number and version of your router.

ios\_conditional.yaml — ~ — Atom

File Edit View Selection Find Packages Help

```
ios_conditional.yaml
```

```
1 ---  
2 - name: IOS Conditional Fact Checker  
3   gather_facts: no  
4   hosts: csr  
5   tasks:  
6     - name: Gather IOS Facts  
7       vars:  
8         credentials:  
9           host: "{{ ansible_host }}"  
10          username: "{{ username }}"  
11          password: "{{ password }}"  
12     ios_facts:  
13       provider: "{{ credentials }}"  
14       register: ios_facts_output  
15     - name: Print IOS Facts  
16       debug:  
17         msg: "{{ ios_facts_output }}"  
18     - name: Print IOS Serial Number and Version  
19       debug:  
20         msg: "SN: {{ ansible_net_serialnum }}, Version: {{ ansible_net_version }}"  
21  
22  
23  
24  
25
```

```

ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
    },
    "lineprotocol": "up",
    "macaddress": "172.16.1.2/32",
    "mediatype": null,
    "mtu": 1514,
    "operstatus": "up",
    "type": null
  },
  "ansible_net_memfree_mb": 2061065,
  "ansible_net_memtotal_mb": 2396128,
  "ansible_net_model": "CSR1000V",
  "ansible_net_neighbors": {
    "null": [
      {
        "host": null,
        "port": null
      }
    ]
  },
  "ansible_net_serialnum": "9E42JNXTJAN",
  "ansible_net_version": "16.08.01a"
},
"changed": false,
"failed": false
}
}

TASK [Print IOS Serial Number and Version] ****
ok: [ignw-csr] => {
  "msg": "SN: 9E42JNXTJAN, Version: 16.08.01a"
}

PLAY RECAP ****
ignw-csr : ok=3    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt1$ █

```

Now that we know the exact serial number and version of our router, let's try to print out the hostname IF the serial number or version is a match to a value we provide (to start, let's just provide the serial number).

To accomplish this we need to use the "when" keyword in Ansible, this is very much like conditional statements in native Python -- when [some conditions are true] run this particular task. Below is an example of how to use this keyword, using this information, try to create a new Task to only display the hostname if the serial is a match:

```

debug:
  msg: "It's a match!"
when:
  "'yum, tacos' == some_variable"

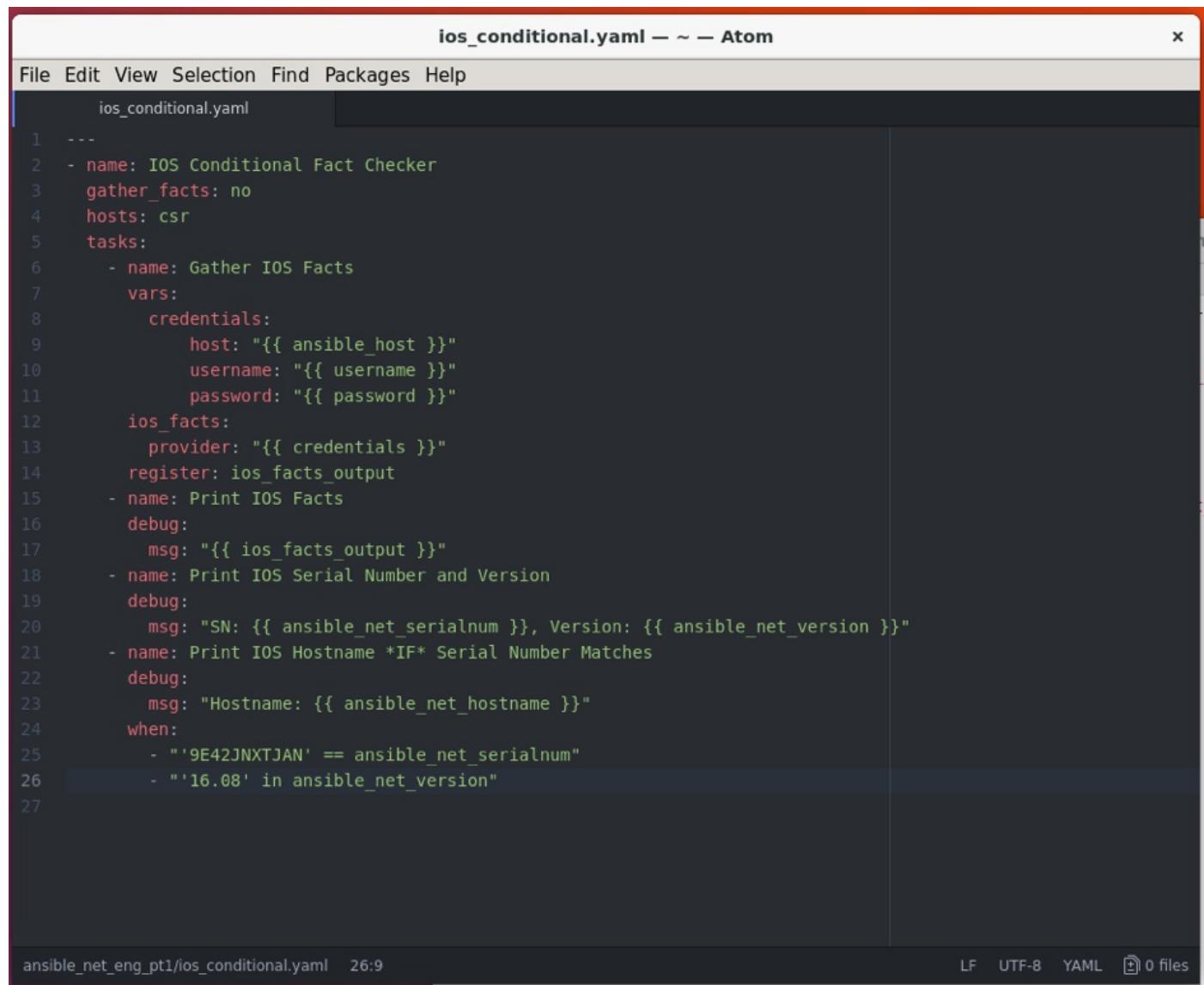
```

If you were able to get that working, try to change the serial number in your "when" clause to validate that it does only work when the serial number is a match! If that worked, you should see that Ansible "skipped" that Task because the condition you provided was not True.

Ansible also allows us to match multiple conditionals very simply by having a list in our "when" clause -- try to

add another list item to your when statement to match the version of the CSR.

Just like Python we have some flexibility about how we craft our conditionals -- so far we have just checked for equality, but we can also see if a string is "in" a variable. An example of where this can be useful is for capturing all devices which have a major version number of 16.07 -- perhaps we don't care about the minor release, we just need all 16.08.XX devices. Change your "when" clause to check for 16.08 "in" the version and re-run your playbook.



```
ios_conditional.yaml -- ~ -- Atom
File Edit View Selection Find Packages Help
ios_conditional.yaml
1 ---
2 - name: IOS Conditional Fact Checker
3   gather_facts: no
4   hosts: csr
5   tasks:
6     - name: Gather IOS Facts
7       vars:
8         credentials:
9           host: "{{ ansible_host }}"
10          username: "{{ username }}"
11          password: "{{ password }}"
12     ios_facts:
13       provider: "{{ credentials }}"
14       register: ios_facts_output
15     - name: Print IOS Facts
16       debug:
17         msg: "{{ ios_facts_output }}"
18     - name: Print IOS Serial Number and Version
19       debug:
20         msg: "SN: {{ ansible_net_serialnum }}, Version: {{ ansible_net_version }}"
21     - name: Print IOS Hostname *IF* Serial Number Matches
22       debug:
23         msg: "Hostname: {{ ansible_net_hostname }}"
24       when:
25         - "'9E42JNXTJAN' == ansible_net_serialnum"
26         - "'16.08' in ansible_net_version"
27
```

ansible\_net\_eng\_pt1/ios\_conditional.yaml 26:9 LF UTF-8 YAML 0 files

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
    "mtu": 1514,
    "operstatus": "up",
    "type": null
  },
  "ansible_net_memfree_mb": 2061065,
  "ansible_net_memtotal_mb": 2396128,
  "ansible_net_model": "CSR1000V",
  "ansible_net_neighbors": {
    "null": [
      {
        "host": null,
        "port": null
      }
    ]
  },
  "ansible_net_serialnum": "9E42JNXTJAN",
  "ansible_net_version": "16.08.01a"
},
"changed": false,
"failed": false
}
}

TASK [Print IOS Serial Number and Version] ****
ok: [ignw-csr] => {
  "msg": "SN: 9E42JNXTJAN, Version: 16.08.01a"
}

TASK [Print IOS Hostname *IF* Serial Number Matches] ****
ok: [ignw-csr] => {
  "msg": "Hostname: *****-csr"
}

PLAY RECAP ****
ignw-csr                  : ok=4      changed=0      unreachable=0      failed=0
ignw@ignw-jumphost:~/ansible_net_eng_pt1$
```

As you can see the hostname has been modified to hide the "ignw" part -- this is because of the terrible lab practice of having our username/password in the hostname. In real life you *probably* won't have to worry about this!

## Working with Lists

In this lab we'll learn how to deal with lists in Ansible.

Create a new Playbook called "lists.yaml" and save it in the ansible\_net\_eng\_pt1 directory.

Remember the data structures you learned about in "regular" Python? As you may have guessed, many of the same things exist in Ansible. Let's take a look at working with lists. To start, steal from your previous playbook to get this new Playbook up to gathering and printing out the ios\_facts.

```
lists.yaml

1  ---
2  - name: Dealing with Lists
3    gather_facts: no
4    hosts: csr
5    tasks:
6      - name: Gather IOS Facts
7        vars:
8          credentials:
9            host: "{{ ansible_host }}"
10           username: "{{ username }}"
11           password: "{{ password }}"
12          ios_facts:
13            provider: "{{ credentials }}"
14            register: ios_facts_output
15        - name: Print IOS Facts
16          debug:
17            msg: "{{ ios_facts_output }}"
18
```

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
    "duplex": null,
    "ipv4": [
        {
            "address": "172.16.1.2",
            "subnet": "32"
        }
    ],
    "lineprotocol": "up",
    "macaddress": "172.16.1.2/32",
    "mediatype": null,
    "mtu": 1514,
    "operstatus": "up",
    "type": null
},
{
    "ansible_net_memfree_mb": 2061065,
    "ansible_net_memtotal_mb": 2396128,
    "ansible_net_model": "CSR1000V",
    "ansible_net_neighbors": {
        "null": [
            {
                "host": null,
                "port": null
            }
        ]
    },
    "ansible_net_serialnum": "9E42JNXTJAN",
    "ansible_net_version": "16.08.01a"
},
{
    "changed": false,
    "failed": false
}
}

PLAY RECAP ****
ignw-csr : ok=2    changed=0    unreachable=0    failed=0
ignw@ignw-jumphost:~/ansible_net_eng_pt1$
```

Let's take a deeper look at this mess of data that is `ios_facts`... starting with the overall structure -- what does it look like to you? It looks a lot like a Python dictionary doesn't it? Or even like JSON? If you copy all of the `ios_facts` data and put it into a JSON validation tool you would indeed see that it is valid JSON. JSON can contain nested JSON (just like Python with dictionaries in dictionaries) or lists. Looking through the `ios_facts`, do you see any lists contained in the data?

How about the "`ansible_net_all_ipv4_addresses`" object -- that sure looks like a list!

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
PLAY [Dealing with Lists] ****
TASK [Gather IOS Facts] ****
ok: [ignw-csr]
TASK [Print List Items] ****
ok: [ignw-csr] => {
  "msg": {
    "ansible_facts": [
      "ansible_net_all_ipv4_addresses": [
        "172.16.1.2",
        "172.16.1.1",
        "10.0.0.5",
        "203.0.113.1",
        "8.8.4.4"
      ],
      "ansible_net_all_ipv6_addresses": [],
      "ansible_net_filesystems": [
        "bootflash:"
      ],
      "ansible_net_gather_subset": [
        "hardware",
        "default",
        "interfaces"
      ],
      "ansible_net_hostname": "*****-CSR",
      "ansible_net_image": "bootflash:packages.conf",
      "ansible_net_interfaces": {
        "GigabitEthernet1": {
          "bandwidth": 1000000,
          "description": "Management - Do not Modify!",
          "duplex": "Full",
          "ipv4": [
            {
              "address": "10.0.0.5",
              "subnet": "24"
            }
          ]
        }
      }
    ]
  }
}
```

Now, how would we go about working with that? Let's try to iterate over that list, we can do this in Ansible using the "with\_items" clause. "with\_items" functions pretty similarly to the "when" clause we used previously, but instead of providing a conditional to evaluate, we provide a list to iterate through.

Remove the task to print all of the ios\_facts out and replace it with a task to iterate over a list.

```
lists.yaml
1 ---
2 - name: Dealing with Lists
3   gather_facts: no
4   hosts: csr
5   tasks:
6     - name: Gather IOS Facts
7       vars:
8         credentials:
9           host: "{{ ansible_host }}"
10          username: "{{ username }}"
11          password: "{{ password }}"
12       ios_facts:
13         provider: "{{ credentials }}"
14       register: ios facts output
15     - name: Print List Items
16       debug:
17         msg: "{{ item }}"
18       with_items: "{{ ansible_net_all_ipv4_addresses }}"
19
```

Run your Playbook and see what happens!

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt1$ ansible-playbook lists.yaml

PLAY [Dealing with Lists] ****
TASK [Gather IOS Facts] ****
ok: [ignw-csr]

TASK [Print List Items] ****
ok: [ignw-csr] => (item=None) => {
    "msg": "172.16.1.2"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "172.16.1.1"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "10.0.0.5"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "203.0.113.1"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "8.8.4.4"
}

PLAY RECAP ****
ignw-csr : ok=2    changed=0    unreachable=0    failed=0
ignw@ignw-jumphost:~/ansible_net_eng_pt1$
```

## Working with Dictionaries

In this lab we'll learn how to deal with dictionaries in Ansible.

Create a new Playbook called "dicts.yaml" and save it in the ansible\_net\_eng\_pt1 directory.

Using the same logic as we used in the previous task dealing with lists, we'll now print out items from the "ansible\_net\_interfaces" dictionary. You should be able to steal from the previous task and simply change the "with\_items" to "with\_dict" -- run your Playbook to see what the output looks like.

```
dicts.yaml
1 ---
2 - name: Dealing with Lists
3   gather_facts: no
4   hosts: csr
5   tasks:
6     - name: Gather IOS Facts
7       vars:
8         credentials:
9           host: "{{ ansible_host }}"
10          username: "{{ username }}"
11          password: "{{ password }}"
12       ios_facts:
13         provider: "{{ credentials }}"
14       register: ios_facts_output
15     - name: Print Dict Items
16       debug:
17         msg: "{{ item }}"
18       with_dict: "{{ ansible_net_interfaces }}"
19
```

```

ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
    }
],
"lineprotocol": "up",
"macaddress": "0050.5693.d372",
"mediatype": "Virtual",
"mtu": 1500,
"operstatus": "up",
"type": "CSR vNIC"
}
}
ok: [ignw-csr] => (item=None) => {
  "msg": {
    "key": "Loopback1",
    "value": {
      "bandwidth": 8000000,
      "description": "IGNW was here!",
      "duplex": null,
      "ipv4": [
        {
          "address": "8.8.4.4",
          "subnet": "32"
        }
      ],
      "lineprotocol": "up",
      "macaddress": "8.8.4.4/32",
      "mediatype": null,
      "mtu": 1514,
      "operstatus": "up",
      "type": null
    }
  }
}

PLAY RECAP ****
ignw-csr : ok=2    changed=0    unreachable=0    failed=0
ignw@ignw-jumphost:~/ansible_net_eng_pt1$
```

Now what if we wanted something actually useful as a message? Based on this example, what would be something useful? Perhaps the name of the interface? How do you think we can "access" that variable? The generally idea is the same as if this was "normal" Python, however the execution is slightly different as we are actually working with the Jinja2 templating language. The "top level" key of our dictionary is "item" -- this comes from the "with\_items"/"with\_dict" assigning the current value we are iterating over as "item". Within that dict, we see that "key" and "value" are keys within this nested dictionary. We ideally would like to print out "GigabitEthernetX" as our message, so we know that we want to "access" the "key". In normal Python we may access it by doing the following:

```
print(item[key])
```

Try to update the "msg" to look like the above and run your Playbook. Uh-oh, that doesn't look good.... "key is undefined" -- no it isn't! Notice in the error description Ansible mentions that this may be a syntax error, and that is exactly what the issue is. In Jinja2 we use a dot (".") instead of the square brackets to access items in our dictionary. Change your Playbook to look like: "item.key" and re-run it.

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt1
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt1$ ansible-playbook dicts.yaml

PLAY [Dealing with Lists] ****
TASK [Gather IOS Facts] ****
ok: [ignw-csr]

TASK [Print List Items] ****
ok: [ignw-csr] => (item=None) => {
    "msg": "Loopback2"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "Loopback0"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "GigabitEthernet1"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "GigabitEthernet2"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "Loopback1"
}

PLAY RECAP ****
ignw-csr : ok=2    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt1$
```

```
dicts.yaml
```

```
1 ---  
2 - name: Dealing with Lists  
3   gather_facts: no  
4   hosts: csr  
5   tasks:  
6     - name: Gather IOS Facts  
7       vars:  
8         credentials:  
9           host: "{{ ansible_host }}"  
10          username: "{{ username }}"  
11          password: "{{ password }}"  
12       ios_facts:  
13         provider: "{{ credentials }}"  
14         register: ios_facts_output  
15     - name: Print Dict Items  
16       debug:  
17         msg: "{{ item.key }}"  
18         with_dict: "{{ ansible_net_interfaces }}"
```

Thats more like it!

*Note:* Depending on the Ansible version you are using there are minor differences in the "msg" / "var" syntax and output. For your reference, this lab is running Ansible 2.5.2.

Now that you've got the lay of the land it is time for you to start with a clean slate to get Ansible rolling on your own. After that you'll start running through some of the same tasks you've done with other tools thus far, as well as some new tasks to highlight some cool things about Ansible.

## Ansible Setup

Ansible is very simple to install, and can simply be "pip" installed on any \*nix system (including MacOS). For this task your environment is setup exactly how it would be immediately after installing Ansible. The two most important steps at this point will be to validate the `ansible.cfg` file which is the primary user modifiable configuration source, and to setup an inventory file to document all of your hosts.

The "`ansible.cfg`" file is a simple file that users are able to modify to change the default behavior of Ansible. Take a look at the official documentation to get an idea of the things that can be adjusted: [Ansible Configuration File Documentation](#).

There are only a few items that we need to care about to get going -- the first of which is the "inventory" argument -- this does exactly what you would think it does: point Ansible to the inventory so it knows which hosts are available and which groups they reside in.

Another important configuration -- particularly for networking devices -- is the "host\_key\_checking" option. Have you ever SSH'd to a device and seen the "REMOTE HOST IDENTIFICATION HAS CHANGED"? This may happen when a router is replaced and the new router has the same configurations -- your laptop (or any

box you would be SSH'ing from) had stored the crypto keys of the previous device, but of course the new router has a different crypto signature! You may also want to disable this so you don't have to manually accept SSH fingerprints. In any case, we'll disable host\_key\_checking for the lab so we have one less thing to worry about!

For now those are the only configurations we need to be worried about, if you looked at the documentation you'll have noticed that both of these arguments fall under the "defaults" heading and as such should be configured under a "[defaults]" heading in our configuration file.

Finally, where does this configuration file actually go? Good question! It can go in any of four locations:

1. ANSIBLE\_CONFIG (an environment variable pointing to your .cfg file living wherever you would like)
2. ansible.cfg in the current directory (where you are executing your Playbook)
3. ansible.cfg in your home directory
4. /etc/ansible/ansible.cfg

The above list is the order of precedence -- the first location/ansible.cfg file is the configuration file that is processed. Create a new folder in your home directory called "ansible\_net\_eng\_pt2" -- we'll simply build our ansible.cfg file here.

```
cd ~/  
mkdir ansible_net_eng_pt2
```

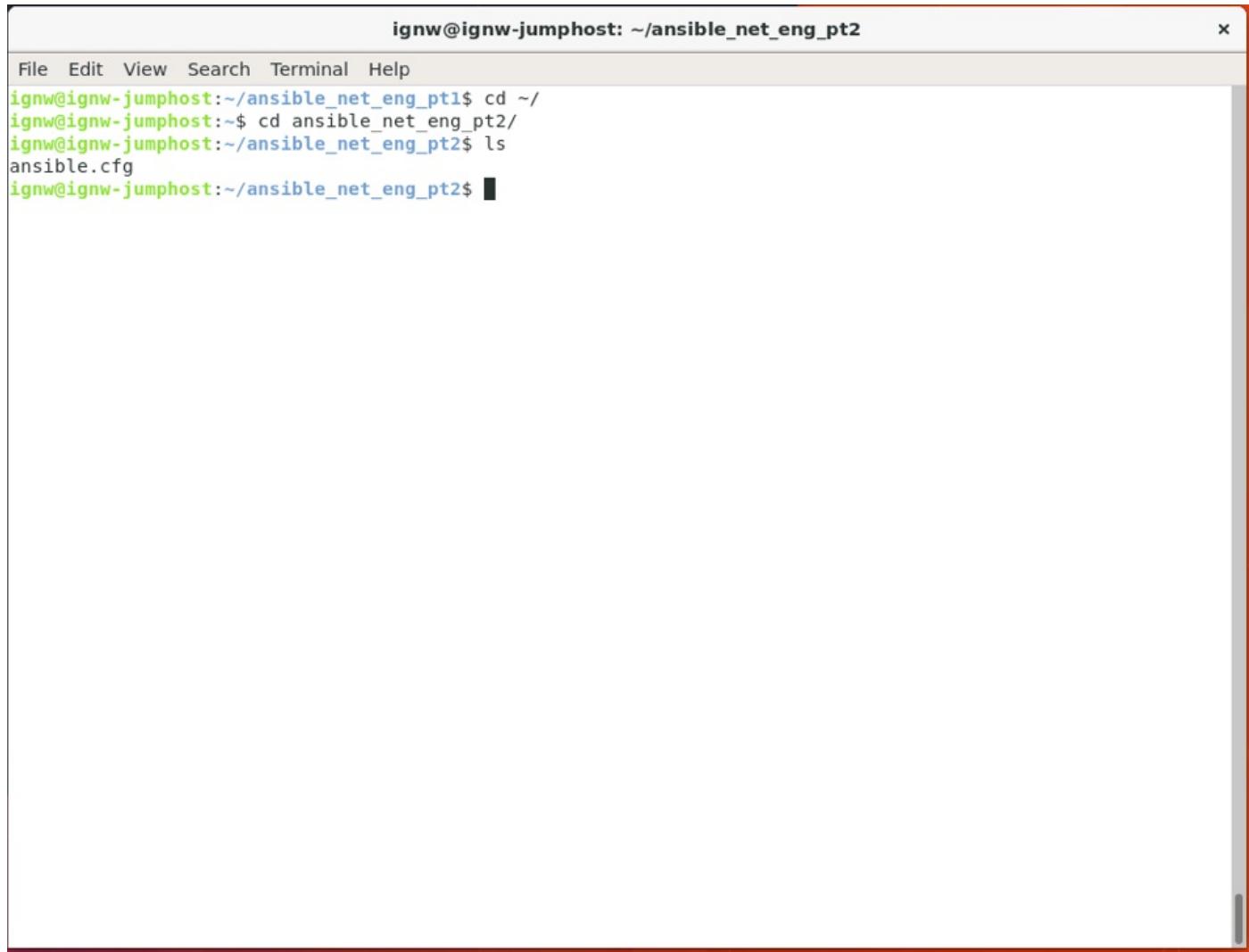
Feel free to use the text editor of your choice to create your file. Create your .cfg file with the appropriate options as depicted below.

ansible.cfg -- ~ -- Atom

File Edit View Selection Find Packages Help

```
ansible.cfg
1 [defaults]
2 host_key_checking = False
3 inventory = /home/ingw/ansible_net_eng_pt2/inventory
4
```

ansible\_net\_eng\_pt2/ansible.cfg 1:1 LF UTF-8 Plain Text 0 files



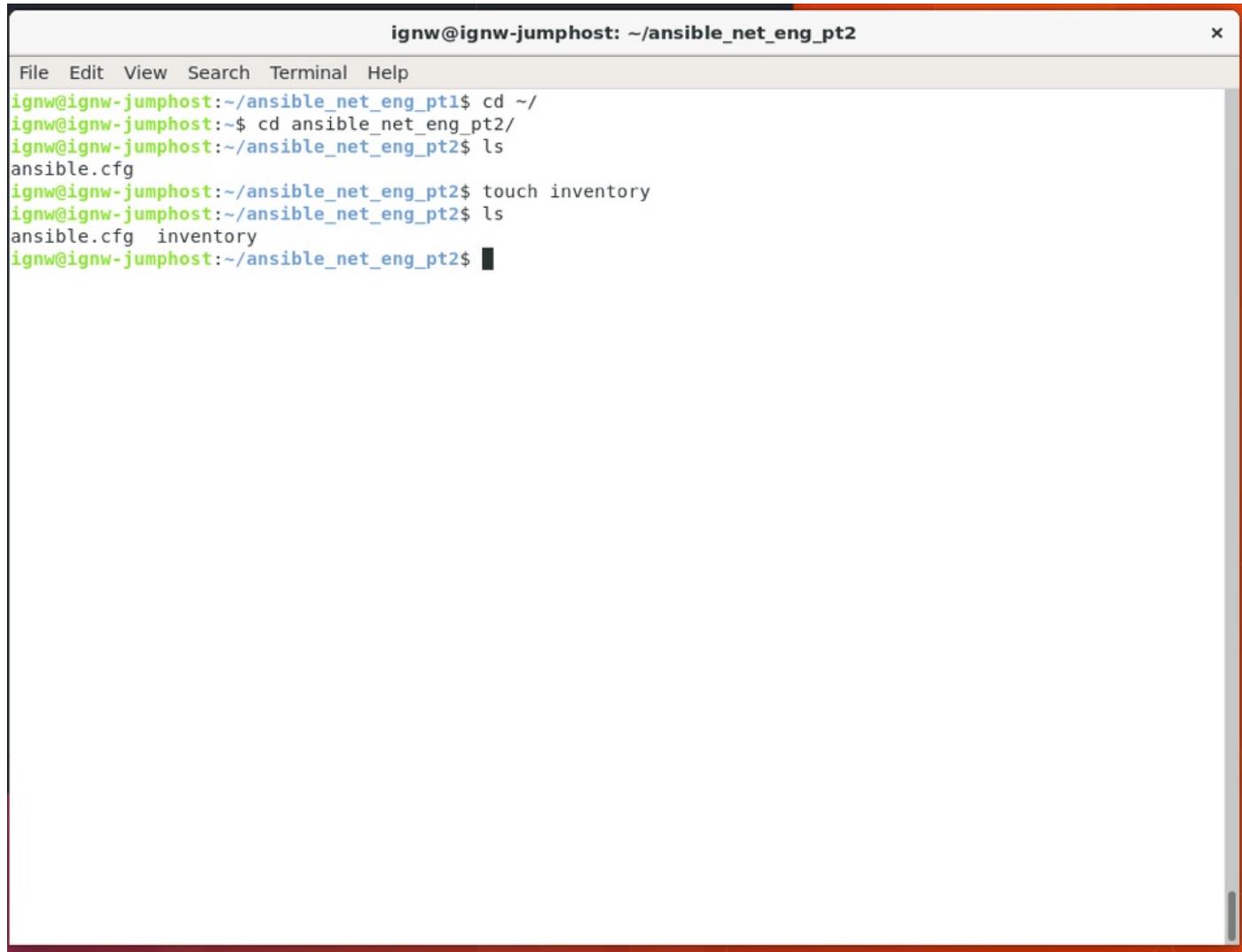
A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/ansible\_net\_eng\_pt2". The window has a standard OS X-style title bar with icons for close, minimize, and zoom. The menu bar at the top includes "File", "Edit", "View", "Search", "Terminal", and "Help". Below the menu bar, the terminal prompt is "ignw@ignw-jumphost:~/ansible\_net\_eng\_pt1\$ cd ~/" followed by "ignw@ignw-jumphost:~\$ cd ansible\_net\_eng\_pt2/". The next line shows "ignw@ignw-jumphost:~/ansible\_net\_eng\_pt2\$ ls" and the file "ansible.cfg". The final line is "ignw@ignw-jumphost:~/ansible\_net\_eng\_pt2\$ █". The terminal window is set against a light gray background.

The inventory file is the next critical piece. In this lab or in small scale production environments this will likely be a very simple file. In larger environments inventory management will almost certainly become one of your prime concerns! For now, we'll stick with a single inventory file, however be aware that Ansible allows for multiple inventory files. Ansible also provides an argument to select the inventory file to execute a Playbook against -- this means you could completely ignore the inventory location in the `ansible.cfg` file, and simply "point" to an inventory file each time you run your "ansible-playbook" commands. Again, for purposes of simplicity and learning we'll stick to a single file.

Your inventory file can come in two flavors -- YAML, or "INI-like"; which you chose is mostly a matter of personal preference. For this task we will create an "INI-like" inventory file as that is the default kind.

The location of the inventory file does not matter, as long as the `.cfg` file is "pointing" to the right place. To make things simple, create your configuration file right in your home directory with the `.cfg` file and your Playbooks from before.

Note that the inventory file (if using INI-like) has no file extension. Create your inventory file using vi/vim/nano/Atom -- you can call it whatever you'd like, but remember that the point is that our `.cfg` file points to it so the names should match!



The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~/ansible\_net\_eng\_pt2". The window has a standard OS X-style title bar with icons for close, minimize, and zoom. The main area of the terminal shows the following command-line session:

```
ignw@ignw-jumphost:~/ansible_net_eng_pt1$ cd ~/
ignw@ignw-jumphost:~$ cd ansible_net_eng_pt2/
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ls
ansible.cfg
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ touch inventory
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ls
ansible.cfg  inventory
ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

Within the inventory file headers in square brackets define groups. You can also assign variables to groups by creating a second header containing the groupname:vars -- ex. "[mygroup:vars]" -- this is a handy way to store variables that affect large swaths of the devices Ansible will be interacting with.

Inventory files also can contain groups of groups in a hierarchical fashion -- the keyword "children" is used to identify a parent group. Take for example two groups "mygroup1" and "mygroup2" a parent group for these would be configured with the following header: "[mysupergroup:children]" -- the child objects of this group would be the child groups "mygroup1" and "mygroup2". Take a look at the example below which contains host groups, a group of groups, and group vars.

inventory — ~ — Atom

File Edit View Selection Find Packages Help

ansible.cfg inventory

```
1 [csr]
2 iignw-csr ansible_host=10.0.0.5
3
4 [asa]
5 iignw-asa ansible_host=10.0.0.8
6
7 [cisco:children]
8 csr
9 asa
10
```

ansible\_net\_eng\_pt2/inventory 10:1 LF UTF-8 Plain Text 0 files

One last point regarding the inventory file is that there is a default group "all" that contains all hosts/groups defined in the inventory file - this is implicit and you do not need to configure it yourself, however you may elect to assign some vars globally -- the "[all:vars]" group vars section is a great place to do that.

Based on the previous example, and what you've learned about inventory files, try to create your inventory file. In the next step we will modify one of the previously created Playbooks to test out our Ansible configurations. Don't worry, if you run into issues your instructor will help you out or provide you an inventory file you can use!

The screenshot shows the Atom code editor interface with a dark theme. The title bar reads "inventory — ~ — Atom". The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. There are two tabs open: "ansible.cfg" and "inventory". The "inventory" tab contains the following YAML configuration:

```
1 [all:vars]
2 username=ignw
3 password=ignw
4
5 [csr]
6 ignw-csr ansible_host=10.0.0.5
7
8 [asa]
9 ignw-asa ansible_host=10.0.0.8
10
11 [cisco:children]
12 csr
13 asa
```

The status bar at the bottom shows "ansible\_net\_eng\_pt2/inventory 3:14" on the left and "LF UTF-8 Plain Text 0 files" on the right.

## Validate Ansible Configuration

In your `ansible_net_eng_pt2` directory copy one of your previously created Playbooks into a new file called `"validate_ansible.yaml"` -- we'll use this file for this task.

Modify the Playbook so that the only Task in the Playbook/Play is to gather `ios_facts` (you can print this out as before if you'd like). Pay attention to the `"hosts"` keyword on the Play -- do you need to modify this based on the inventory file you created?

```
validate_ansible.yaml

1  ---
2  - name: Validating our Ansible CFG File
3    gather_facts: no
4    hosts: csr
5    tasks:
6      - name: Gather IOS Facts
7        vars:
8          credentials:
9            host: "{{ ansible_host }}"
10           username: "{{ username }}"
11           password: "{{ password }}"
12         ios_facts:
13           provider: "{{ credentials }}"
14           register: ios_facts_output
15     - name: Print List Items
16       debug:
17         msg: "{{ item.key }}"
18         with_dict: "{{ ansible_net_interfaces }}"
19
```

Once you think your new Playbook is ready to run, give it a shot!

Did your Playbook execute? Did you get any errors? At this point a common error would be "Unable to parse [some path] as an inventory source" -- if you get this error, double check to make sure you've entered the correct path for your inventory file. You may also have gotten an error about "invalid connection" as depicted below.

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ pwd
/home/ignw/ansible_net_eng_pt2
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ls
ansible.cfg  inventory  validate_ansible.yaml
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook validate_ansible.yaml

PLAY [Validating our Ansible CFG File] ****
TASK [Gather IOS Facts] ****
fatal: [ignw-csr]: FAILED! => {"changed": false, "msg": "Connection type ssh is not valid for this module"}
      to retry, use: --limit @/home/ignw/ansible_net_eng_pt2/validate_ansible.retry

PLAY RECAP ****
ignw-csr                  : ok=0      changed=0      unreachable=0      failed=1

ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

If you got this error, why do you suppose you did? If you did *not* get it, what do you think you did to prevent you from getting it? Think about the error -- "expected connection=local, got ssh" -- now that you've spent some time with Ansible you know that by default Ansible will SSH to remote systems to run Tasks, but as we've covered that isn't an option for network devices, so Ansible actually connects to itself, figures out its Tasks, and then executes them via SSH or an API -- in other words, the initial connection should be set to "local".

When dealing with Ansible specifically for networking, it is common to setup this under [all:vars] so that it is applied to all hosts -- obviously if you are using Ansible to configure networking devices and systems you may want to reconsider this. Below is an example inventory file that can be used for the remainder of the Ansible tasks -- compare yours to this to get things working.

inventory — ~ — Atom

File Edit View Selection Find Packages Help

ansible.cfg inventory

```
1 [all:vars]
2 username=ignw
3 password=ignw
4 ansible_connection=local
5
6 [csr]
7 ignw-csr ansible_host=10.0.0.5
8
9 [asa]
10 ignw-asa ansible_host=10.0.0.8
11
12 [cisco:children]
13 csr
14 asa
15
```

ansible\_net\_eng\_pt2/inventory 15:1 LF UTF-8 Plain Text 0 files

Your Playbook output should look similar to that shown below if you elected to print out the facts.

```

ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook validate_ansible.yaml

PLAY [Validating our Ansible CFG File] ****
TASK [Gather IOS Facts] ****
ok: [ignw-csr]

TASK [Print List Items] ****
ok: [ignw-csr] => (item=None) => {
    "msg": "Loopback2"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "Loopback0"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "GigabitEthernet1"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "GigabitEthernet2"
}
ok: [ignw-csr] => (item=None) => {
    "msg": "Loopback1"
}

PLAY RECAP ****
ignw-csr : ok=2      changed=0      unreachable=0      failed=0
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ █

```

## Host and Group Vars -- Dealing with more Variables

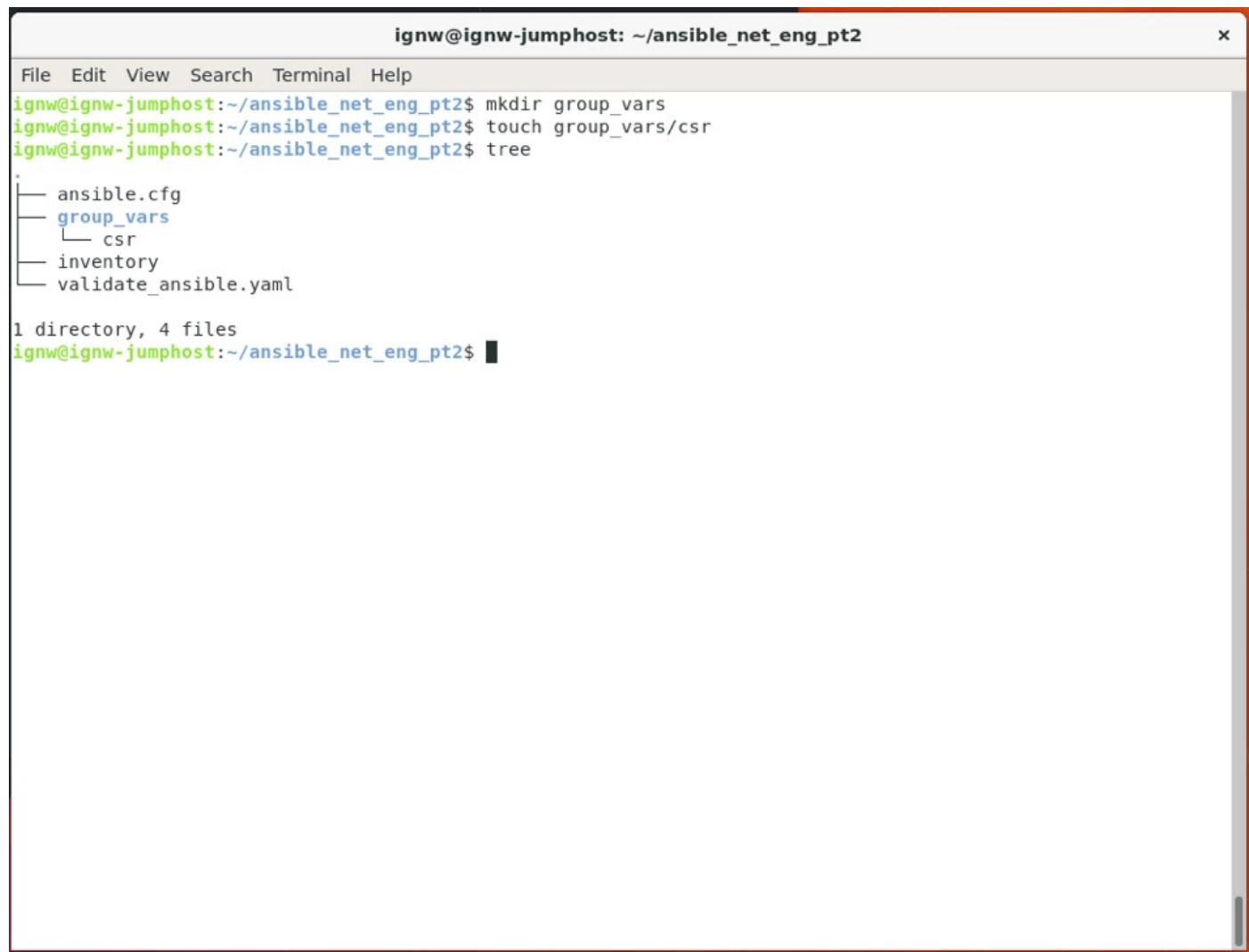
As you've learned, the inventory file can contain variables that are associated with groups or individual hosts (think about the "username" under the all:vars section, or "ansible\_host" under the groups). This is a handy feature and works great for small scale work like what we're doing here in the lab, however when you are using Ansible to manage hundreds of devices across multiple sites/groups the inventory file can get very messy very quickly! Thankfully, Ansible provides several other easy ways to deal with storing host or group specific variables -- with the aptly named host\_vars and group\_vars concept.

What would be an example of a "group" specific variable? Any answer to that question would of course depend on how groups are arranged in your Ansible deployment -- a group could be a site or a region, or it could be a type of device (note that a device can be part of more than one group!). For our simple lab, our groups are based on device types, given that, what types of variables do you think would apply to all devices of type "CSR"?

This is of course an incredibly subjective question, but it's only meant to get you thinking! As an example, let's assume that all our devices of type "CSR" have a different NTP server configuration from that of our non-router devices. Let's use the "group\_var" configuration to handle this.

Group vars are stored in a folder appropriately named "group\_vars" -- this folder should be stored in the same

directory as the inventory file that you are running your Playbook against. Within the group\_vars folder another INI-like file is named after each group for which you would like to configure variables.



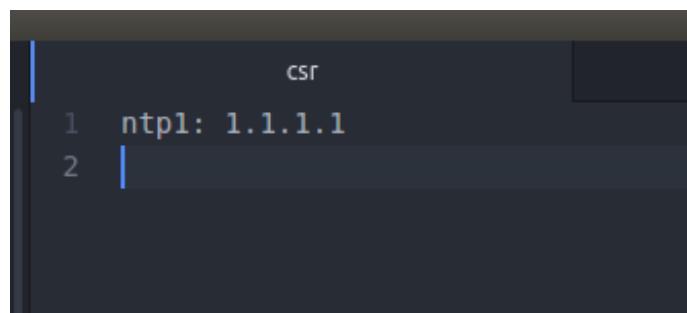
```
ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ mkdir group_vars
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ touch group_vars/csr
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ tree
.
├── ansible.cfg
└── group_vars
    └── csr
└── inventory
└── validate_ansible.yaml

1 directory, 4 files
ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

Create your group\_vars folder, and "csr" (no extension -- also acceptable are .yml, .yaml, or .json files, but we'll stick with the INI-like format for this lab).

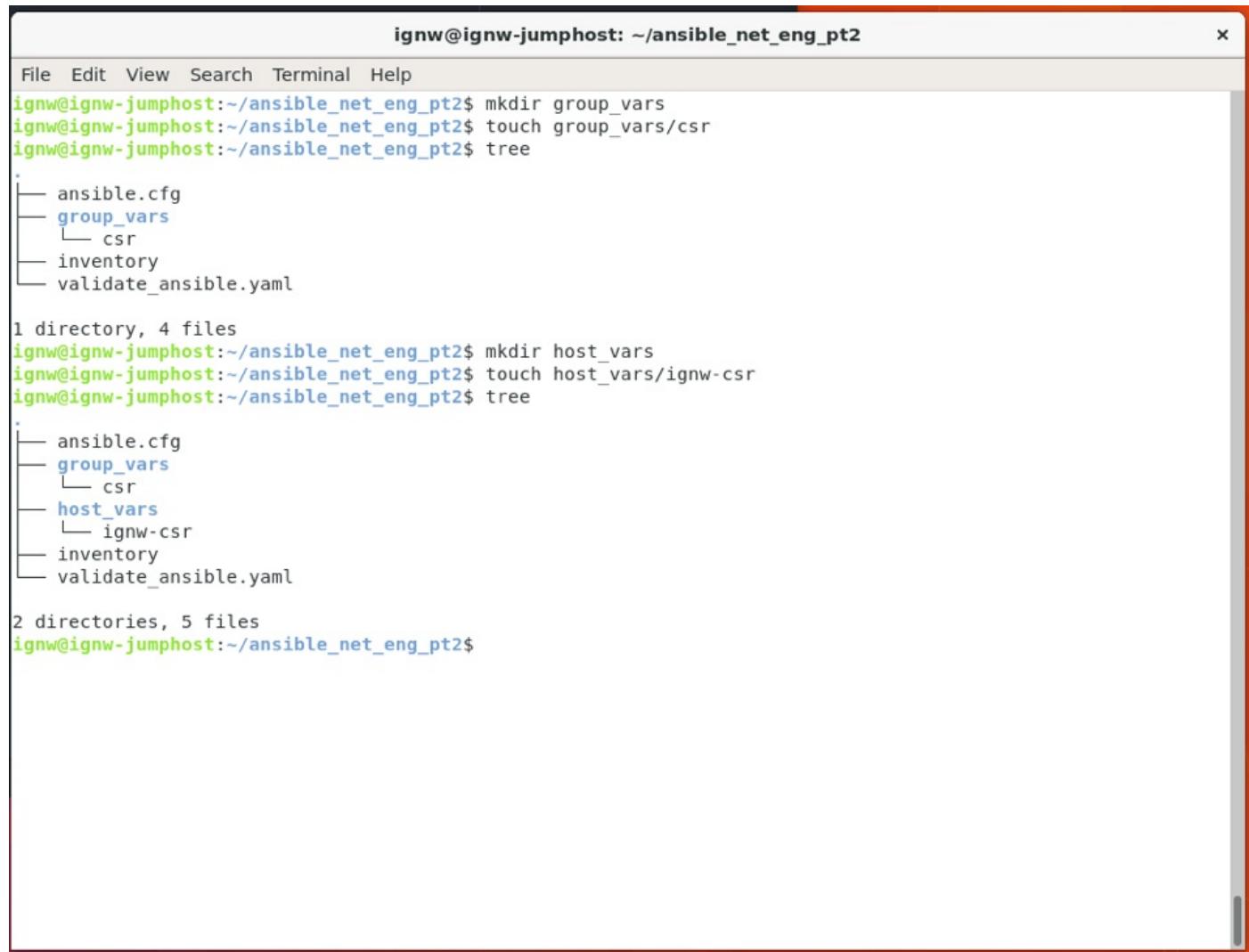
**Note:** If you used different naming for your devices or groups in your inventory file, ensure that the hostname/group name you provided Ansible and the host/group vars file matches your inventory file!

Within your csr file, add a simple variable for NTP as shown below:



```
csr
1 ntp1: 1.1.1.1
2 |
```

Before we move on and test the group vars, let's also try to build out our host vars. Host vars are configured in nearly the exact same way, so go ahead and build a new folder called `host_vars` (in the same location as the `group_vars` folder) and create a file called "CSR-1" to represent our single CSR host. Remember to make sure this name matches the name you provided in your inventory!



```
ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ mkdir group_vars
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ touch group_vars/csr
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ tree
.
├── ansible.cfg
├── group_vars
│   └── csr
└── inventory
└── validate_ansible.yaml

1 directory, 4 files
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ mkdir host_vars
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ touch host_vars/ignw-csr
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ tree
.
├── ansible.cfg
├── group_vars
│   └── csr
└── host_vars
    └── ignw-csr
└── inventory
└── validate_ansible.yaml

2 directories, 5 files
ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

The idea behind host vars is of course variables that would be locally significant to a particular host. This could be things like IP addressing, BGP peers or ASN, specific ACLs, or any other "one-off" configuration. At scale, managing unique configurations obviously becomes more and more difficult, so with Ansible -- or any other automation/orchestration -- it is wise to try to eliminate unique configurations wherever possible. For now, we'll simply assume that our host CSR-1 is in a different timezone than the rest of our devices so we want to set that as a unique configuration for it.

In your `ignw-csr` file, add a variable for "timezone" with a value of PST.

The screenshot shows the Atom code editor interface. The title bar reads "ignw-csr — ~ — Atom". The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. Below the menu is a tab bar with four tabs: "ansible.cfg", "inventory", "csr", and "ignw-csr". The "ignw-csr" tab is active, displaying the following YAML content:

```
1 timezone: pst
2
```

The status bar at the bottom left shows "ansible\_net\_eng\_pt2/host\_vars/ignw-csr 2:1" and "validate ansible.yaml". The status bar at the bottom right shows "LF" (Line Feed), "UTF-8", "Plain Text", and "0 files".

Now, let's validate that our variables are working as intended -- we'll cover actual configuration changes in the next task, so for now let's use our friend the "debug" module to simply print out our variables as validation.

Create a new Playbook file -- "validate\_variables.yaml" -- in your user directory. For this Playbook, provide a name, set gather\_facts to "no", for your hosts, we'll run this against the "csr" group (or your group name if you used a different one) as that will test both our host and group variables.

Create two Tasks -- one to print your group var (ntp1), and a second to print your host var (timezone). Run your playbook to see if you were successful.

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook validate_variables.yaml

PLAY [Validating our Group and Host Vars] ****
TASK [Print Group Vars] ****
ok: [ignw-csr] => {
    "msg": "1.1.1.1"
}

TASK [Print Host Vars] ****
ok: [ignw-csr] => {
    "msg": "pst"
}

PLAY RECAP ****
ignw-csr : ok=2     changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

```
validate_variables.yaml
```

```
1  ---
2  - name: Validating our Group and Host Vars
3    gather_facts: no
4    hosts: csr
5    tasks:
6      - name: Print Group Vars
7        debug:
8          msg: "{{ ntp1 }}"
9      - name: Print Host Vars
10        debug:
11          msg: "{{ timezone }}"
12
```

While we won't play with it in these labs, do know that you can store lists and dictionaries in the host or group vars files -- this gives you a bit more flexibility with managing these constructs.

**Note:** Check out the Ansible Best Practices guide for more details on how to structure your inventory, variable files, Playbooks, and even more stuff we haven't covered yet!

## Beyond Facts, the "ios\_command" Module

Up to this point we've worked exclusively with facts and the structure of Ansible, but what if you want to gather "facts" (data!?) that is *not* contained within the ios\_facts module? Recall back to looking at the full output of the ios\_facts -- there was quite a bit of good information, but it was very much platform/interface specific facts/configurations and was completely devoid of useful "show" type functionality.

The "ios\_command" module allows us to send "arbitrary commands to an ios node and returns the results read from the device" ([IOS Command Documentation](#)), let's build a new Playbook called "ios\_commands.yaml" to get started.

The first part of your Playbook should be very familiar to you by now -- triple hashes at the top, give the Play a name, turn off gather\_facts, and set a target for hosts (we'll continue to just use the csr group).

ios\_command, exactly like ios\_facts, requires a "provider" -- that is the host (IP/name) and login credentials. We previously defined these under the individual task (in the ios\_conditionals task for example), but we can store these at the Play level instead which is arguably a bit "cleaner" as they may be reused throughout a Play. Configure your Play as shown below:

```
ios_commands.yaml
1  ---
2  - name: Finally, some SHOW Commands!
3    gather_facts: no
4    hosts: csr
5    vars:
6      credentials:
7        host: "{{ ansible_host }}"
8        username: "{{ username }}"
9        password: "{{ password }}"
10 |
```

Now create two Tasks -- one to gather (and register!) the output of "show ip route", and a second to debug (print) the output of your show command. Use the documentation (link above) and the "ios\_conditional.yaml" Playbook as an example of how to do this (ios\_command is very similar to ios\_facts!). If you get stuck, look ahead for a screen shot of a complete Playbook or ask your instructor for help!

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
.0/32 is subnetted, 2 subnets\nC      172.16.1.1 is directly connected, Loopback0\nC      172.16.1.2 is
directly connected, Loopback2\n      203.0.113.0/24 is variably subnetted, 2 subnets, 2 masks\nC      203.
0.113.0/26 is directly connected, GigabitEthernet2\nL      203.0.113.1/32 is directly connected, GigabitEt
hernet2"
],
"stdout_lines": [
[
    "Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP",
    "        D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area",
    "        N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2",
    "        E1 - OSPF external type 1, E2 - OSPF external type 2",
    "        i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2",
    "        ia - IS-IS inter area, * - candidate default, U - per-user static route",
    "        o - ODR, P - periodic downloaded static route, H - NHRP, l - LISPs",
    "        a - application route",
    "        + - replicated route, % - next hop override, p - overrides from PfR",
    "",
    "Gateway of last resort is not set",
    "",
    "        8.0.0.0/32 is subnetted, 1 subnets",
    "C          8.8.4.4 is directly connected, Loopback1",
    "        10.0.0.0/32 is subnetted, 1 subnets",
    "S          10.255.255.2 [1/0] via 203.0.113.2",
    "        172.16.0.0/32 is subnetted, 2 subnets",
    "C          172.16.1.1 is directly connected, Loopback0",
    "C          172.16.1.2 is directly connected, Loopback2",
    "        203.0.113.0/24 is variably subnetted, 2 subnets, 2 masks",
    "C          203.0.113.0/26 is directly connected, GigabitEthernet2",
    "L          203.0.113.1/32 is directly connected, GigabitEthernet2"
]
]
}
}

PLAY RECAP ****
ignw-csr : ok=2    changed=0    unreachable=0    failed=0
ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

```
ios_commands.yaml

1  ---
2  - name: Finally, some SHOW Commands!
3    gather_facts: no
4    hosts: csr
5    vars:
6      credentials:
7        host: "{{ ansible_host }}"
8        username: "{{ username }}"
9        password: "{{ password }}"
10   tasks:
11     - name: Gimme the routes!
12       ios_command:
13         provider: "{{ credentials }}"
14         commands: show ip route
15         register: ios_commands_output
16     - name: Print IOS Command Output
17       debug:
18         msg: "{{ ios_commands_output }}"
19
```

Look at the data that is output -- what type of data structure is returned? Recall that just like in Python square brackets ("[", "]") indicate a list and squiggly brackets ("{", "}") indicate a dictionary -- it looks like the data returned is a dictionary containing lists for "stdout" and "stdout\_lines". Notice that the `ios_command` module gave us two ways to consume the output -- either as a "blob" of data (much like we got back from Pexpect!), or formatted into individual lines that would be a bit easier to parse.

## Finally, Configurations!

Now that you have worked with both the `ios_facts` and the `ios_command` module, it's time to learn about the `ios_config` module. This module is very similar to the previous modules, but as the name implies it is used to send configurations to IOS devices!

Create another Playbook called "first\_configurations.yaml", and start with the same data as the previous task (we'll need the `credentials` variable just like before), for hosts continue to use the `csr` group for now.

Create a Task in your Playbook using the "`ios_config`" module. This module requires the "provider" argument the same as before, so pass that the `credentials` dictionary we created.

Configuration lines are passed to the `ios_config` module with the "lines" option -- for now we'll pass a single line of config, but if we needed to pass more (an ACL with multiple lines, or multiple configs for an interface for example) we could do so by using a YAML list with a config per line.

For this task, let's provide an interface on our CSR with a new description. Assign the value of your description to the "lines" key as pictured below.

```
first_configurations.yaml
```

```
1  ---
2  - name: Woohoo! Configuration Time!
3    gather_facts: no
4    hosts: csr
5    vars:
6      credentials:
7        host: "{{ ansible_host }}"
8        username: "{{ username }}"
9        password: "{{ password }}"
10   tasks:
11     - name: Starting Small
12       ios_config:
13         provider: "{{ credentials }}"
14         lines: description Yay Ansible!
```

Think about when you log into a router to configure a description -- do you simply log into the router and type "description yada yada yada"? Of course not! You first ensure that you are in the appropriate interface that you wish to apply the description to -- Ansible is no different! The `ios_config` module has an argument named "parents" for exactly this. The parents argument is exactly what it sounds like -- in our case the "parent" object for our description is the interface we would like the description to be applied to.

Build your Play out as depicted below:

```
first_configurations.yaml
```

```
1  ---
2  - name: Woohoo! Configuration Time!
3    gather_facts: no
4    hosts: csr
5    vars:
6      credentials:
7        host: "{{ ansible_host }}"
8        username: "{{ username }}"
9        password: "{{ password }}"
10   tasks:
11     - name: Starting Small
12       ios_config:
13         provider: "{{ credentials }}"
14         lines: description Yay Ansible!
15         parents: interface GigabitEthernet2
16
```

Once you think you've got your Playbook ready, try to run it.

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook first_configurations.yaml
PLAY [Woohoo! Configuration Time!] ****
TASK [Starting Small] ****
changed: [ignw-csr]

PLAY RECAP ****
ignw-csr : ok=1    changed=1    unreachable=0    failed=0

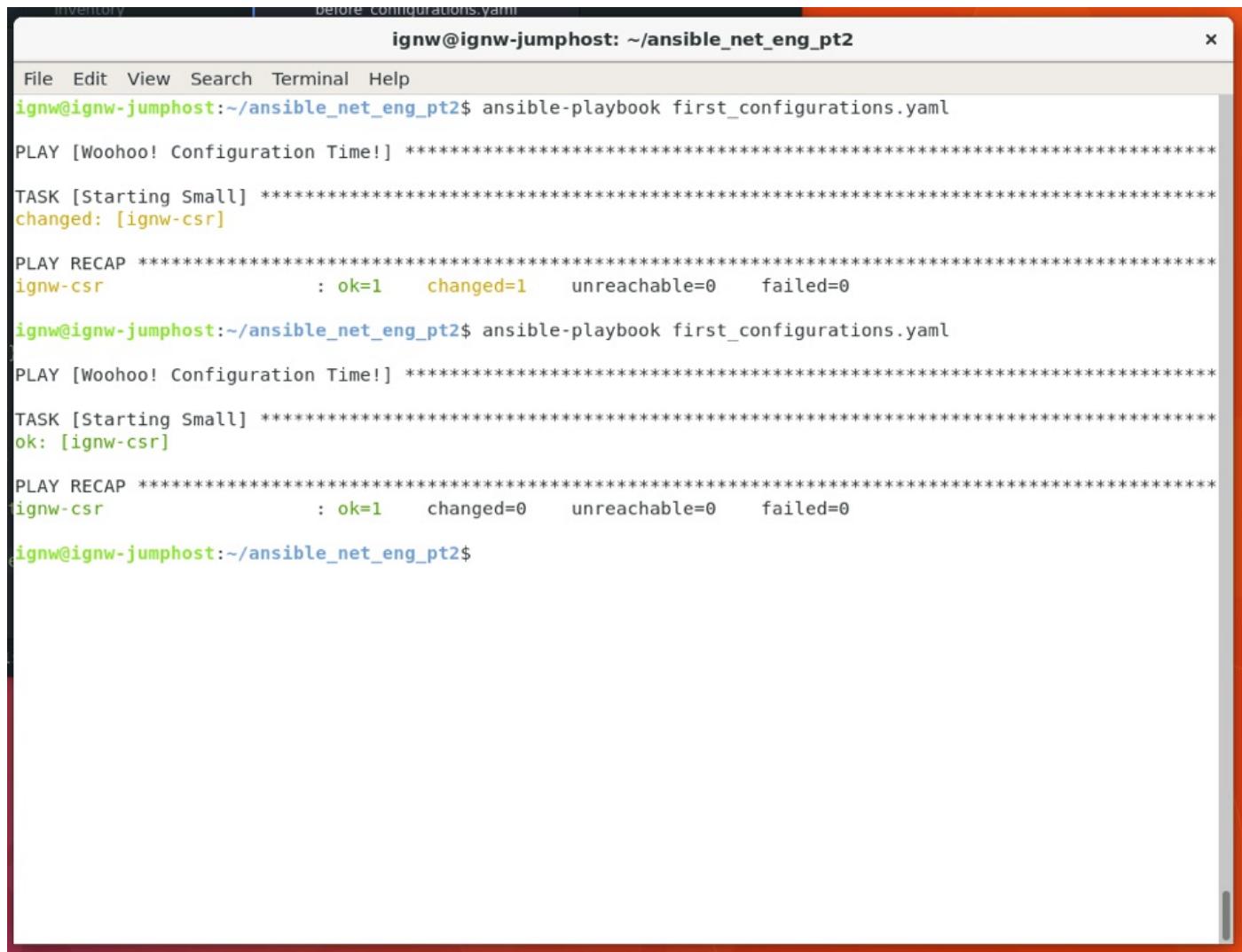
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook first_configurations.yaml
PLAY [Woohoo! Configuration Time!] ****
TASK [Starting Small] ****
ok: [ignw-csr]

PLAY RECAP ****
ignw-csr : ok=1    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

Hey, that's a new one! We haven't seen Ansible come back with yellow output yet have we? If red is bad, what is yellow.... slow down? Speed up?

With Ansible yellow simply indicates that a change has occurred. Let's unpack this a little bit... what is a change (queue up thoughts of the Phoenix Project! You should read it if you haven't already!)? Before we think about that too much, run your Playbook again.



The screenshot shows a terminal window titled "before Configurations.yaml" with the command "ignw@ignw-jumphost: ~/ansible\_net\_eng\_pt2\$ ansible-playbook first\_configurations.yaml". The output shows:

```
PLAY [Woohoo! Configuration Time!] ****
TASK [Starting Small] ****
changed: [ignw-csr]

PLAY RECAP ****
ignw-csr : ok=1    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook first_configurations.yaml

PLAY [Woohoo! Configuration Time!] ****
TASK [Starting Small] ****
ok: [ignw-csr]

PLAY RECAP ****
ignw-csr : ok=1    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

Weird, it's all green now?! If your Playbook is still showing changed, go back and look very carefully at the your Playbook compared to the screen shots above! (**tip:** check for case!)

So why does our Playbook return differing outputs? If you log into the router you'll see that nothing has changed, and the script did successfully change the description. If you remove the description and add it back in, what does the Playbook output?

Interesting! When the Playbook makes a change it reports it, and when it doesn't need to change anything (because the appropriate configurations are already in place) it doesn't make a change and simply reports that everything is "OK". This is a critical concept in configuration management/infrastructure as code, and in Ansible in particular -- the concept of **idempotency**.

An "official" definition may or may not help to clear up what exactly this is: "denoting an element of a set that is unchanged in value when multiplied or otherwise operated on by itself." That's quite a definition! A perhaps simpler, and certainly more applicable definition could be: "for an operation (or service call) to be idempotent, clients can make that same call repeatedly while producing the same result" -- if you've got a few minutes take a look at this video to learn a bit more about **idempotence**.

Back to our Playbook -- the Playbook understands when the configuration is already in the *desired state* and therefore makes no configuration changes -- thus giving us the "OK", if we go and remove or change our description manually and re-run the Playbook Ansible sees that the *current state* and the *desired state* are not the same, and executes a change -- and gives us our yellow "changed" output.

A last, and very important piece to idempotency in Ansible (specifically for network devices, less so for systems!), the structure of our Playbook can cause our Plays to *not* be idempotent. If your Playbook was showing "changed" every time you ran it earlier, were you able to fix it? The fix was (probably) to ensure that the "parents" argument was set to *exactly* the name of the interface that you were hoping to configure. The reason for this is that Ansible is (for now) simply screen-scraping the CSR config -- it expects (remember all the fun with the Pexpect labs!) to see the interface name exactly as it is in the running configuration, with the description (or other configurations in the lines argument) nestled under it. If the parents argument does not match *exactly* as it is shown in the running configuration Ansible can't match current to desired state, and thus runs the configurations anyway.

## IOS\_Config - Before and Match

Sometimes when configuring devices it may be desirable to "no" or negate a configuration before entering new configs. The easy example of this is for access-control-lists -- often times instead of relying on sequence numbers operators will "no" the previous access-list and paste in a new access-list in its entirety. The `ios_config` module has an argument for exactly this purpose -- the "before" argument.

Before takes a list of commands that should be ran *before* the config lines in your Task.

**Note:** while this is maybe a contrived example, any scenario where it may be beneficial to negate previous configurations could be handled with the "before" argument.

Create another Playbook called "before\_configurations.yaml", and start with the same data as the previous tasks.

Create a Task as you did in the previous lab, for your "lines" argument provide a list with three extended access-list entries. Example:

```
- permit ip host 1.1.1.1 host 8.8.8.8 log  
- permit ip host 1.1.1.2 host 8.8.8.8 log  
- permit ip host 1.1.1.3 host 8.8.8.8 log
```

Run your Playbook.

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook before_configurations.yaml
PLAY [The Before Argument] ****
TASK [Build me an ACL!!] ****
An exception occurred during task execution. To see the full traceback, use -vvv. The error was: ignw-csr(config)#
fatal: [ignw-csr]: FAILED! => {"changed": false, "module_stderr": "Traceback (most recent call last):\n File \"/tmp/ansible_eigPN9/ansible_module_ios_config.py\", line 583, in <module>\n     main()\n File \"/tmp/ansible_eigPN9/ansible_module_ios_config.py\", line 512, in main\n     load_config(module, commands)\n File \"/tmp/ansible_eigPN9/ansible_modlib.zip/ansible/module_utils/network/ios/ios.py\", line 162, in load_config\n File \"/tmp/ansible_eigPN9/ansible_modlib.zip/ansible/module_utils/connection.py\", line 149, in __rpc__\nansible.module_utils.connection.ConnectionError: permit ip host 1.1.1.1 host 8.8.8.8 log\r\n ^\r\n% Invalid input detected at '^' marker.\r\nr\nignw-csr(config)#\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 1}
      to retry, use: --limit @/home/ignw/ansible_net_eng_pt2/before_configurations.retry

PLAY RECAP ****
ignw-csr                  : ok=0    changed=0    unreachable=0    failed=1
ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

```
before_configurations.yaml
1  ---
2  - name: The Before Argument
3    gather_facts: no
4    hosts: csr
5    vars:
6      credentials:
7        host: "{{ ansible_host }}"
8        username: "{{ username }}"
9        password: "{{ password }}"
10   tasks:
11     - name: Build me an ACL!!
12       ios_config:
13         provider: "{{ credentials }}"
14         lines:
15           - permit ip host 1.1.1.1 host 8.8.8.8 log
16           - permit ip host 8.8.8.8 host 1.1.1.1 log
17           - permit ip host 2.2.2.2 host 8.8.8.8 log
18
```

Uh-oh, what gives? "invalid input detected"... hmm, think back to the previous lab, how did you "enter" interface config mode? We need to do something similar here; instead of interface config mode, we need to get into the ACL config mode. Add a "parents" argument for "ip access-list extended AnsibleACL", then try your Playbook once more.

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook before_configurations.yaml
PLAY [The Before Argument] ****
TASK [Build me an ACL!!] ****
An exception occurred during task execution. To see the full traceback, use -vvv. The error was: ignw-csr(config)#
fatal: [ignw-csr]: FAILED! => {"changed": false, "module_stderr": "Traceback (most recent call last):\n File \"/tmp/ansible_eigPN9/ansible_module_ios_config.py\", line 583, in <module>\n     main()\n File \"/tmp/ansible_eigPN9/ansible_module_ios_config.py\", line 512, in main\n     load_config(module, commands)\n File \"/tmp/ansible_eigPN9/ansible_modlib.zip/ansible/module_utils/network/ios/ios.py\", line 162, in load_config\n File \"/tmp/ansible_eigPN9/ansible_modlib.zip/ansible/module_utils/connection.py\", line 149, in __rpc__\nansible.module_utils.connection.ConnectionError: permit ip host 1.1.1.1 host 8.8.8.8 log\r\n    ^\r\n% Invalid input detected at '^' marker.\r\n\r\nignw-csr(config)#\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 1}
      to retry, use: --limit @/home/ignw/ansible_net_eng_pt2/before_configurations.retry

PLAY RECAP ****
ignw-csr : ok=0     changed=0     unreachable=0     failed=1

ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook before_configurations.yaml
PLAY [The Before Argument] ****
TASK [Build me an ACL!!] ****
changed: [ignw-csr]

PLAY RECAP ****
ignw-csr : ok=1     changed=1     unreachable=0     failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

SSH to your CSR to validate that the changes you were expecting occurred.

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ssh 10.0.0.5
Password:

ignw-csr#sh run | b access-list
ip access-list extended test
permit ip host 1.1.1.1 host 8.8.8.8 log
permit ip host 8.8.8.8 host 1.1.1.1 log
permit ip host 2.2.2.2 host 8.8.8.8 log
!
!
control-plane
!
!
!
!
line con 0
stopbits 1
line vty 0 4
login local
!
wsma agent exec
!
wsma agent config
!
wsma agent filesys
!
wsma agent notify
!
!
end
ignw-csr#
```

If all is well, what do you think would happen if you ran your Playbook again? Would it be idempotent? Give it a shot!

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook before_configurations.yaml
PLAY [The Before Argument] ****
TASK [Build me an ACL!!] ****
ok: [ignw-csr]

PLAY RECAP ****
ignw-csr : ok=1    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

Great! You've written a Play to idempotently enforce desired state of an ACL! What happens if you need to modify your ACL a bit? Add a line to your ACL in the second position, then re-run your Playbook.

before\_configurations.yaml — ~ — Atom

File Edit View Selection Find Packages Help

ansible.cfg	inventory	before_configurations.yaml
-------------	-----------	----------------------------

```

1  ---
2  - name: The Before Argument
3    gather_facts: no
4    hosts: csr
5    vars:
6      credentials:
7        host: "{{ ansible_host }}"
8        username: "{{ username }}"
9        password: "{{ password }}"
10   tasks:
11     - name: Build me an ACL!!
12       ios_config:
13         provider: "{{ credentials }}"
14         lines:
15           - permit ip host 1.1.1.1 host 8.8.8.8 log
16           - permit ip host 7.7.7.7 host 6.6.6.6 log
17           - permit ip host 8.8.8.8 host 1.1.1.1 log
18           - permit ip host 2.2.2.2 host 8.8.8.8 log
19       parents: ip access-list extended test
20

```

ignw@ignw-jumphost: ~/ansible\_net\_eng\_pt2

File Edit View Search Terminal Help

```

ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook before_configurations.yaml

PLAY [The Before Argument] ****
TASK [Build me an ACL!!] ****
changed: [ignw-csr]

PLAY RECAP ****
ignw-csr : ok=1    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

The Playbook ran, and it knew that a change had to occur, and it did in fact add the new line to our ACL, however the line is not in the appropriate order... and order matters for ACLs!! We need to tell Ansible to match our ACL lines *exactly* -- it turns out there is an argument "match" that can do just that for us. Add a match argument to your Task as outlined in the screen shot below, then run your Playbook again.

before\_configurations.yaml — ~ — Atom

File Edit View Selection Find Packages Help

ansible.cfg	inventory	before_configurations.yaml
-------------	-----------	----------------------------

```
1 ---  
2 - name: The Before Argument  
3   gather_facts: no  
4   hosts: csr  
5   vars:  
6     credentials:  
7       host: "{{ ansible_host }}"  
8       username: "{{ username }}"  
9       password: "{{ password }}"  
10  tasks:  
11    - name: Build me an ACL!!  
12      ios_config:  
13        provider: "{{ credentials }}"  
14        lines:  
15          - permit ip host 1.1.1.1 host 8.8.8.8 log  
16          - permit ip host 7.7.7.7 host 6.6.6.6 log  
17          - permit ip host 8.8.8.8 host 1.1.1.1 log  
18          - permit ip host 2.2.2.2 host 8.8.8.8 log  
19        parents: ip access-list extended test  
20        match: exact
```

ignw@ignw-jumphost: ~/ansible\_net\_eng\_pt2

File Edit View Search Terminal Help

Password:

```
ignw-csr#sh run | b access-list  
ip access-list extended test  
permit ip host 1.1.1.1 host 8.8.8.8 log  
permit ip host 8.8.8.8 host 1.1.1.1 log  
permit ip host 2.2.2.2 host 8.8.8.8 log  
permit ip host 7.7.7.7 host 6.6.6.6 log  
!  
!  
!  
control-plane  
!
```

Well Ansible is now indicating that things are changed, so take a look at the router to confirm. What gives!? Nothing changed? In this case, we need to provide one more argument to Ansible -- the "before" argument.

Before takes a command (or list of commands) that must be ran prior to the execution of the commands in the "lines" argument. In this case, we can provide a value of "no ip access-list extended test" to the before argument, thus allowing Ansible to remove the ACL if a change is needed!

ignw@ignw-jumphost: ~/ansible\_net\_eng\_pt2

File Edit View Search Terminal Help

```
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook before_configurations.yaml

PLAY [The Before Argument] ****
TASK [Build me an ACL!!] ****
changed: [ignw-csr]

PLAY RECAP ****
ignw-csr : ok=1    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

```
before_configurations.yaml

1  ---
2  - name: The Before Argument
3    gather_facts: no
4    hosts: csr
5    vars:
6      credentials:
7        host: "{{ ansible_host }}"
8        username: "{{ username }}"
9        password: "{{ password }}"
10   tasks:
11     - name: Build me an ACL!!
12       ios_config:
13         provider: "{{ credentials }}"
14         lines:
15           - permit ip host 1.1.1.1 host 8.8.8.8 log
16           - permit ip host 8.8.8.8 host 2.2.2.2 log
17           - permit ip host 8.8.8.8 host 1.1.1.1 log
18           - permit ip host 2.2.2.2 host 8.8.8.8 log
19         parents: ip access-list extended test
20         match: exact
21         before: no ip access-list extended test
22
```

Run your Playbook a few times to ensure it is executing in an idempotent fashion!

```
ignw@ignw-jumphost: ~/ansible_net_eng_pt2
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook before_configurations.yaml
PLAY [The Before Argument] ****
TASK [Build me an ACL!!] ****
changed: [ignw-csr]

PLAY RECAP ****
ignw-csr : ok=1    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt2$ ansible-playbook before_configurations.yaml
PLAY [The Before Argument] ****
TASK [Build me an ACL!!] ****
ok: [ignw-csr]

PLAY RECAP ****
ignw-csr : ok=1    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/ansible_net_eng_pt2$
```

## Handlers - Saving your Work!

So far you've used Ansible to gather facts, gather output from commands, configure things, enforce state in an idempotent fashion -- but you haven't done one very important task -- save your work!

Ansible has a concept called "Handlers" -- you can think of Handlers like a Task (or Tasks) that is only ran when "asked to". You could of course simply use the "ios\_command" module to just send a "copy run start" or a "write mem" at the end of every Playbook, but unless a change has occurred there isn't really any point.

Since we know when changes do occur (because Ansible is reporting this to us) we can very simply add a "handler" to only write mem when it is warranted.

Copy your Playbook from the last task into a new file called "wr\_handler.yaml".

Under the Play, add a section called "handlers" as shown in the screen shot below:

```

wr_handler.yaml •
1  ---
2  - name: Handle This!
3    gather_facts: no
4    hosts: csr
5    vars:
6      credentials:
7        host: "{{ ansible_host }}"
8        username: "{{ username }}"
9        password: "{{ password }}"
10       timeout: 30
11
12  tasks:
13    - name: Build me an ACL!!
14      ios_config:
15        provider: "{{ credentials }}"
16        lines:
17          - permit ip host 1.1.1.1 host 8.8.8.8 log
18          - permit ip host 8.8.8.8 host 1.1.1.1 log
19          - permit ip host 2.2.2.2 host 8.8.8.8 log
20        parents: ip access-list extended test
21        match: exact
22        before: no ip access-list extended test
23
24  handlers:
25    - name: Save your work!
26      ios_command:
27        provider: "{{ credentials }}"
28        commands: "wr"

```

Great, that section is fairly straight forward, but how do we "trigger" it? In a Task where we execute a config we can use the "notify" argument -- this will notify a handler that a Task has ran. Add a "notify" argument pointing to the name of your handler in the Build ACL Task:

```
wr_handler.yaml
```

```
1 ---  
2 - name: Handle This!  
3   gather_facts: no  
4   hosts: csr  
5   vars:  
6     credentials:  
7       host: "{{ ansible_host }}"  
8       username: "{{ username }}"  
9       password: "{{ password }}"  
10      timeout: 30  
11  tasks:  
12    - name: Build me an ACL!!  
13      ios_config:  
14        provider: "{{ credentials }}"  
15        lines:  
16          - permit ip host 1.1.1.1 host 8.8.8.8 log  
17          - permit ip host 8.8.8.8 host 1.1.1.1 log  
18          - permit ip host 2.2.2.2 host 8.8.8.8 log  
19        parents: ip access-list extended test  
20        match: exact  
21        before: no ip access-list extended test  
22      notify: "Save your work!"  
23  
24  handlers:  
25    - name: Save your work!  
26      ios_command:  
27        provider: "{{ credentials }}"  
28        commands: "wr"
```

Run your Playbook -- if no change was made, you should *not* see the Handler pop up in the Ansible output. SSH in and delete your ACL, then re-run your Playbook -- Ansible should re-deploy the ACL AND notify the handler to save your configurations!

# Setting up Jenkins

## Overview and Objectives

In this lab you will setup a fresh installation of Jenkins. In later labs we will be using Jenkins to orchestrate what is effectively "CI/CD" for network devices.

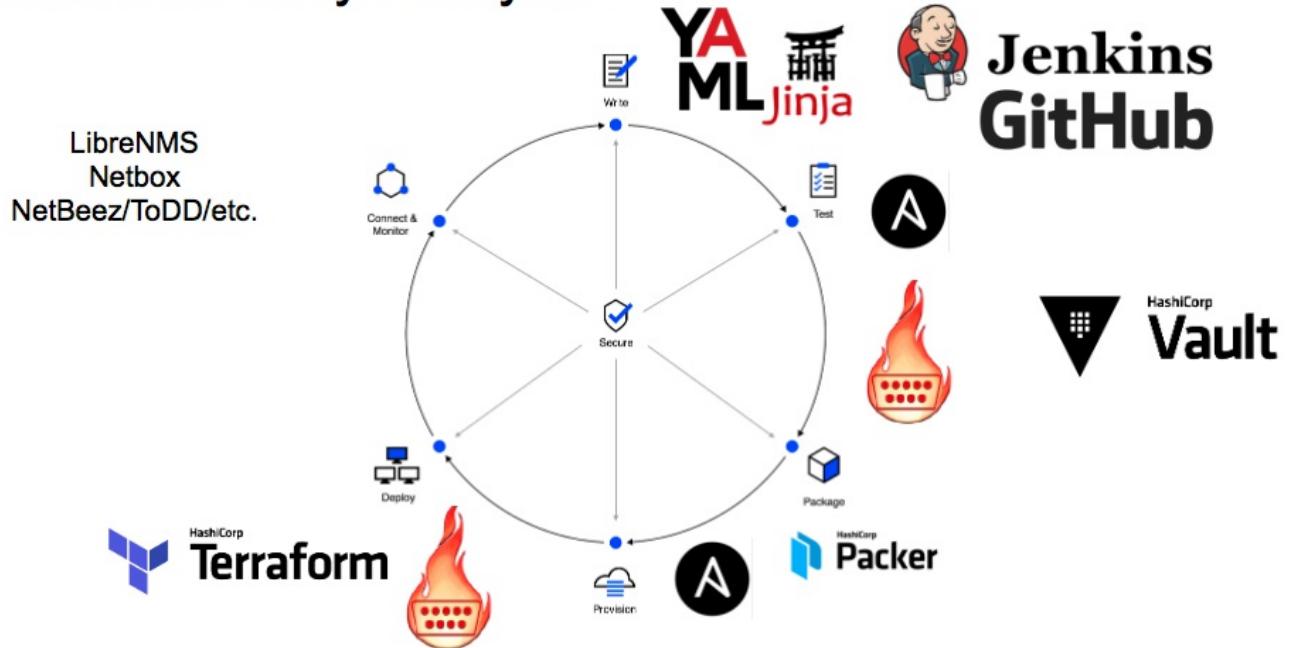
Objectives:

- Perform initial configuration of Jenkins
- Gain basic familiarity with the Jenkins interface

## Network Delivery Lifecycle Overview

Jenkins is a core piece of the Network Delivery Lifecycle -- in our case we will be using it as a CI/CD orchestrator for our configuration management and deployment. Below is a very high level view of our Network Lifecycle overlaid on the HashiCorp Application Lifecycle map:

## Network Delivery Lifecycle



At this point we're just going to be getting Jenkins setup, but by the end of this course you should have a strong understanding of how Jenkins fits into the above chart!

## Jenkins Installation

As you've learned, Jenkins is an open-source automation server. We'll be using Jenkins to drive our network "Infrastructure-as-Code" (IaC) initiative, leveraging Jenkins' integration with GitHub, and ability to execute arbitrary scripts at "stages" in our Pipeline.

The lab environment contains a Jenkins server for each student. On these servers Jenkins has been installed

but not setup. The installation process for Jenkins is quite simple and can be accomplished as follows (on Debian based systems such as the Ubuntu Server instance in the lab):

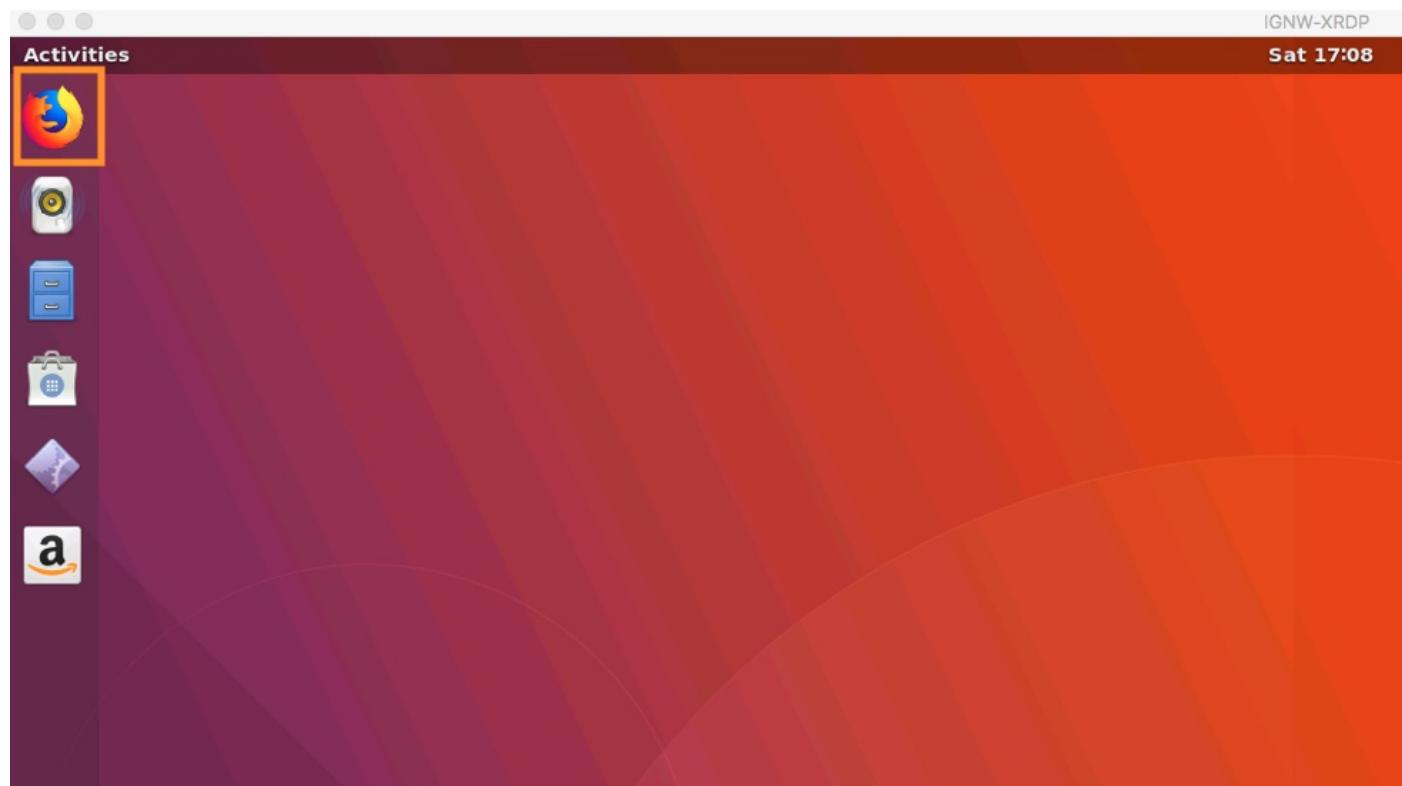
**Note You do not need to do the below, it is strictly informational!**

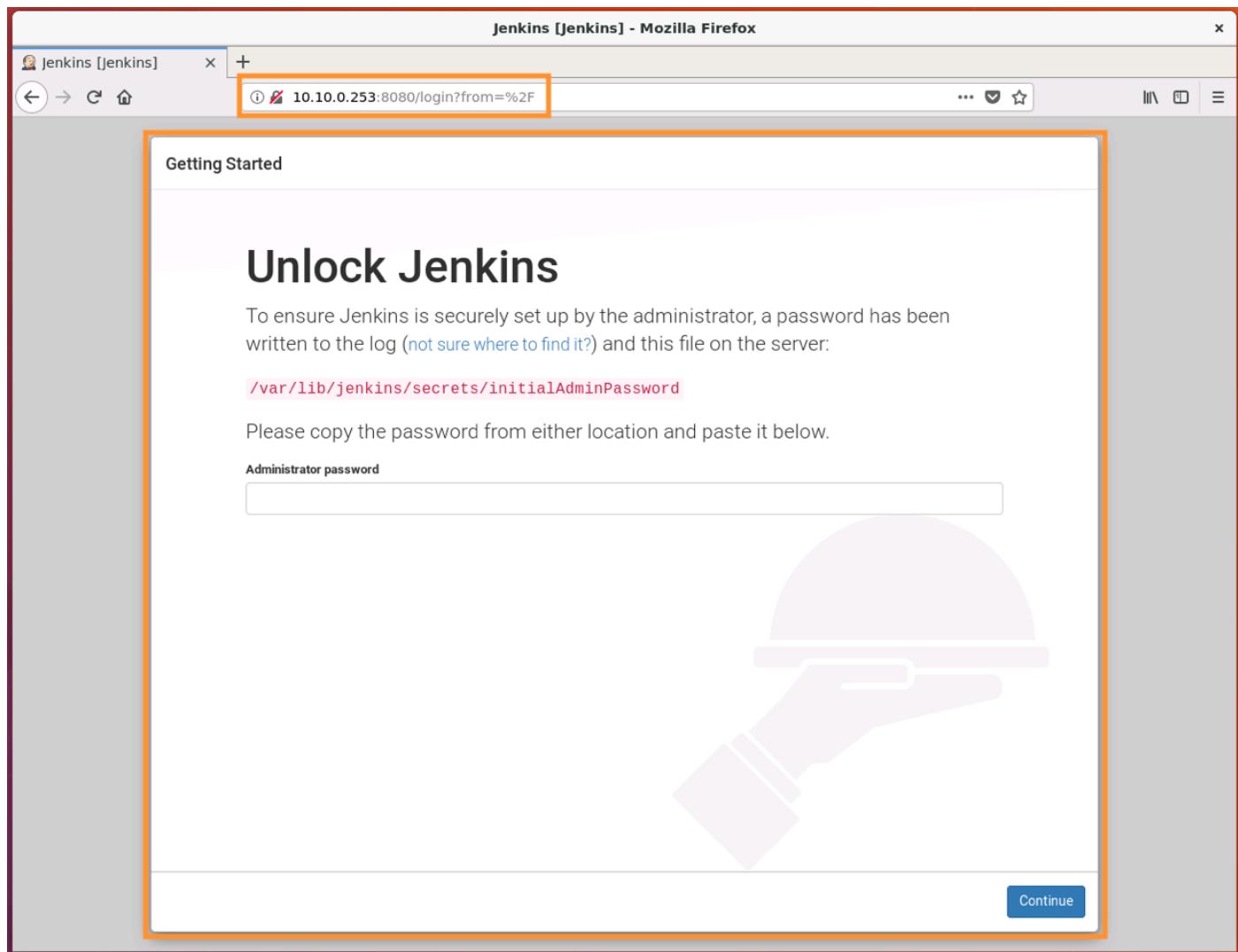
```
wget https://pkg.jenkins.io/debian/jenkins-ci.org.key  
sudo apt-key add /home/ignw/jenkins-ci.org.key  
echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list  
sudo apt-get update && sudo apt-get upgrade  
sudo apt-get install jenkins
```

## Initial Log In

With the basic installation out of the way, we are ready to get Jenkins setup. A standard installation of Jenkins will, by default, provide its HTTP service on port 8080. You can access your Jenkins instance in one of two ways:

- Connect to your Jumphost (RDP), and use the Firefox browser to connect to "<http://10.10.0.253:8080>".
- Using a browser on your local system, connect to "http://[IP Provided by Instructor]:80XX" where XX is your student number.



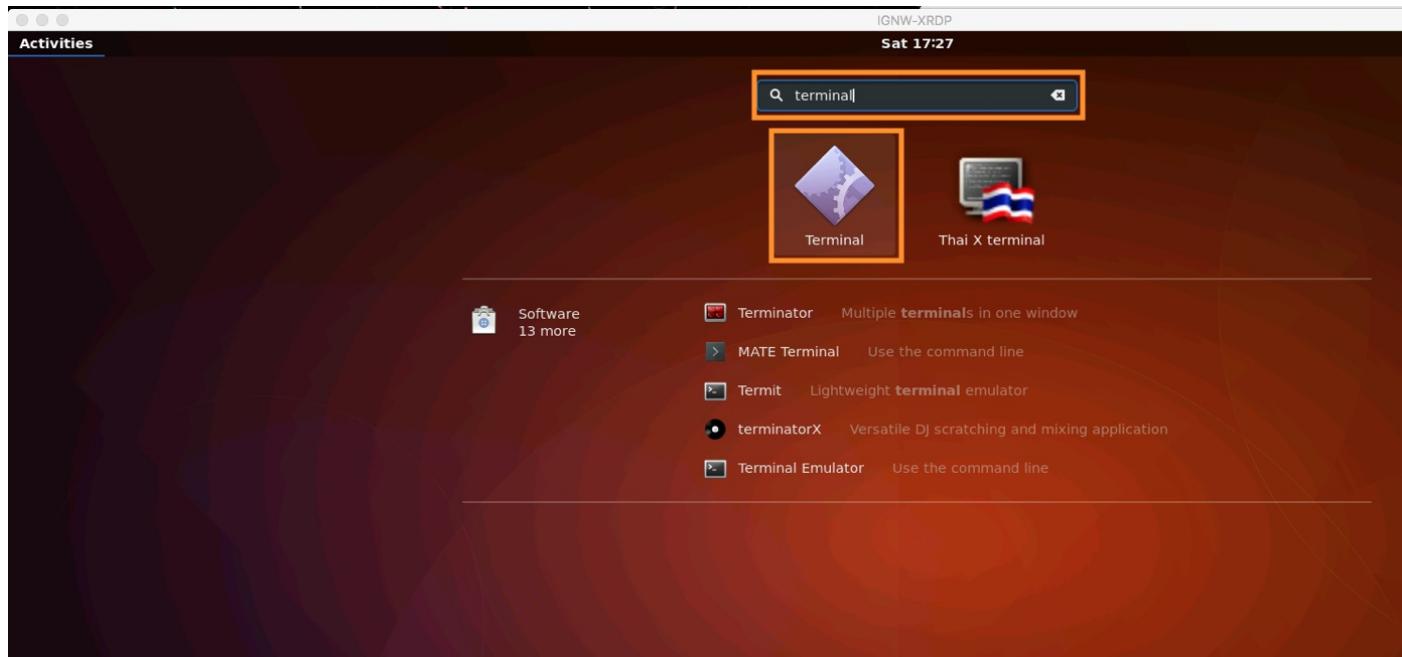
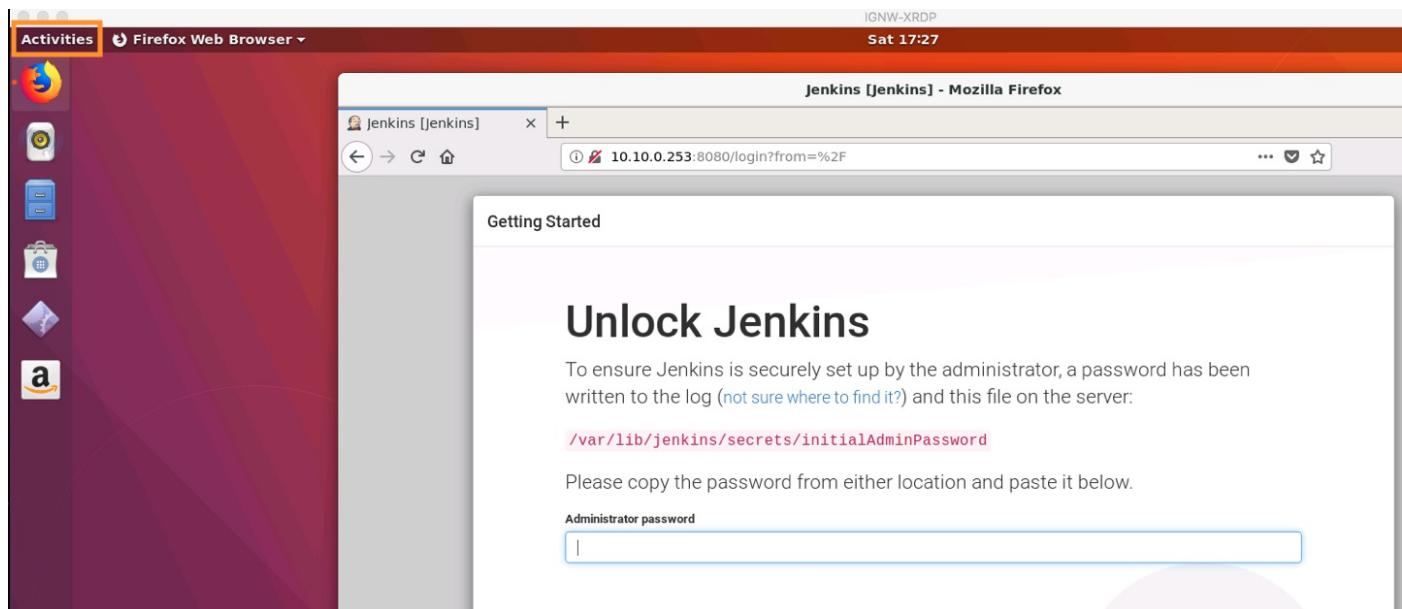


Once you've successfully connected to the Jenkins server, you'll be greeted with a "Getting Started" page. Jenkins stores a secure password in a log file during installation which it is now prompting us to find. Happily for us, Jenkins is nice enough to provide the path where we can find this:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

SSH into the Jenkins server to go ahead and snag this password. Once again, you can do this directly from the Jumphost or from your workstation via the provided SSH port (see the Getting Started section).

Click on the "Activities" button on the top left of the Jumphost, and then type "terminal", and launch the terminal app that we'll use for SSH.



SSH to the Jenkins server (10.10.0.253) with a username of "ignw". When prompted about the authenticity of the SSH key, enter "yes" to add the hosts SSH key to the known hosts for the Jumphost. The password for the Jenkins server is "ignw".

```
ignw@ignw-jenkins: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ ssh ignw@10.10.0.253
The authenticity of host '10.10.0.253 (10.10.0.253)' can't be established.
ECDSA key fingerprint is SHA256:MER0V0q5Tp7KeG8I00B6yu3Vd810iltw7V3E3K+/twI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.10.0.253' (ECDSA) to the list of known hosts.
ignw@10.10.0.253's password:
Welcome to Ubuntu 17.10 (GNU/Linux 4.13.0-39-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

New release '18.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sat May  5 16:42:17 2018 from 10.10.10.80
ignw@ignw-jenkins:~$
```

cat the contents of the file that Jenkins is asking us to interrogate, you will need to do this as root:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

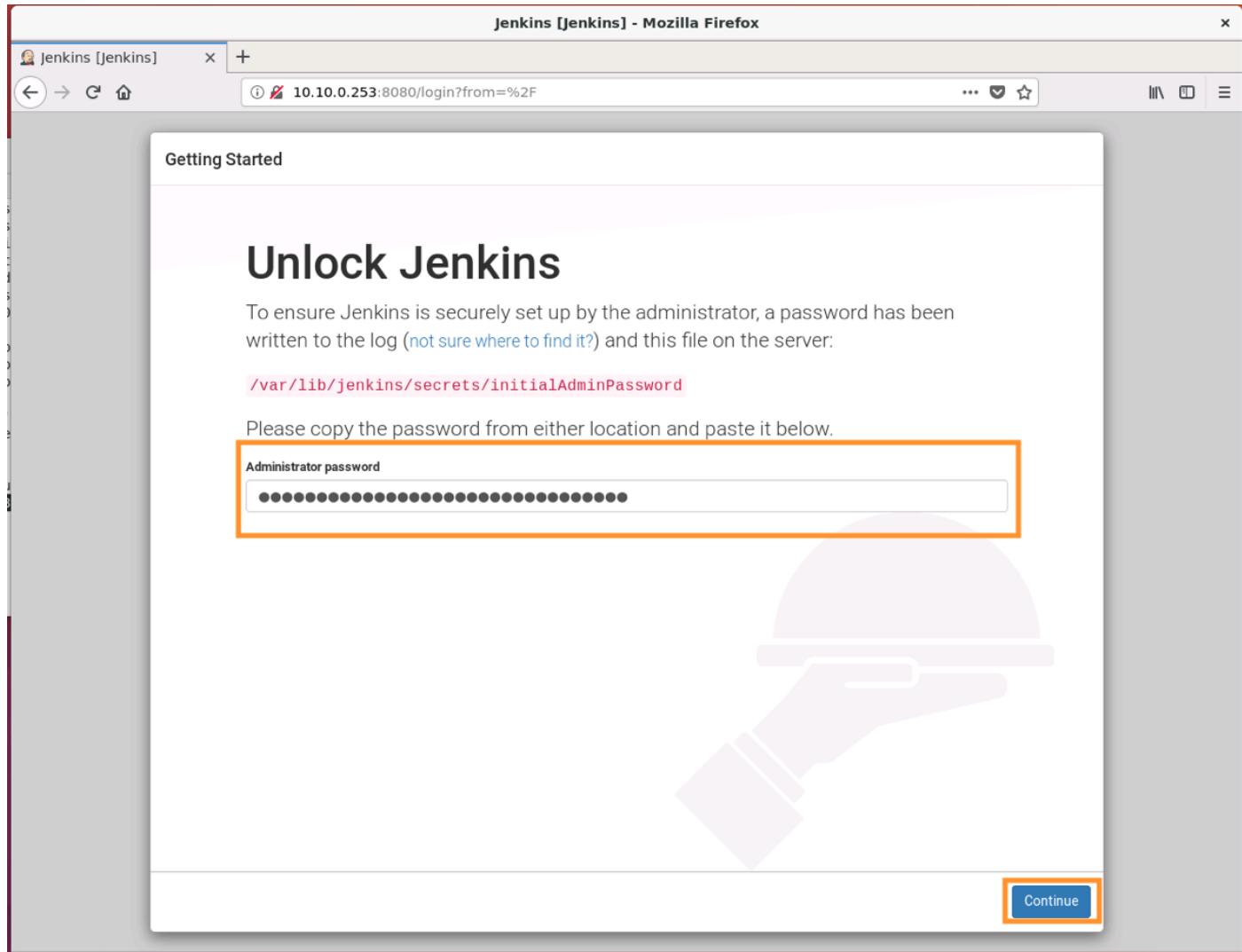
```
ignw@ignw-jenkins: ~
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ ssh ignw@10.10.0.253
The authenticity of host '10.10.0.253 (10.10.0.253)' can't be established.
ECDSA key fingerprint is SHA256:NEB0VQq5TpZKeG8LQ0B6yu3Vd8l0iltw7V3E3K+/twI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.10.0.253' (ECDSA) to the list of known hosts.
ignw@10.10.0.253's password:
Welcome to Ubuntu 17.10 (GNU/Linux 4.13.0-39-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

New release '18.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

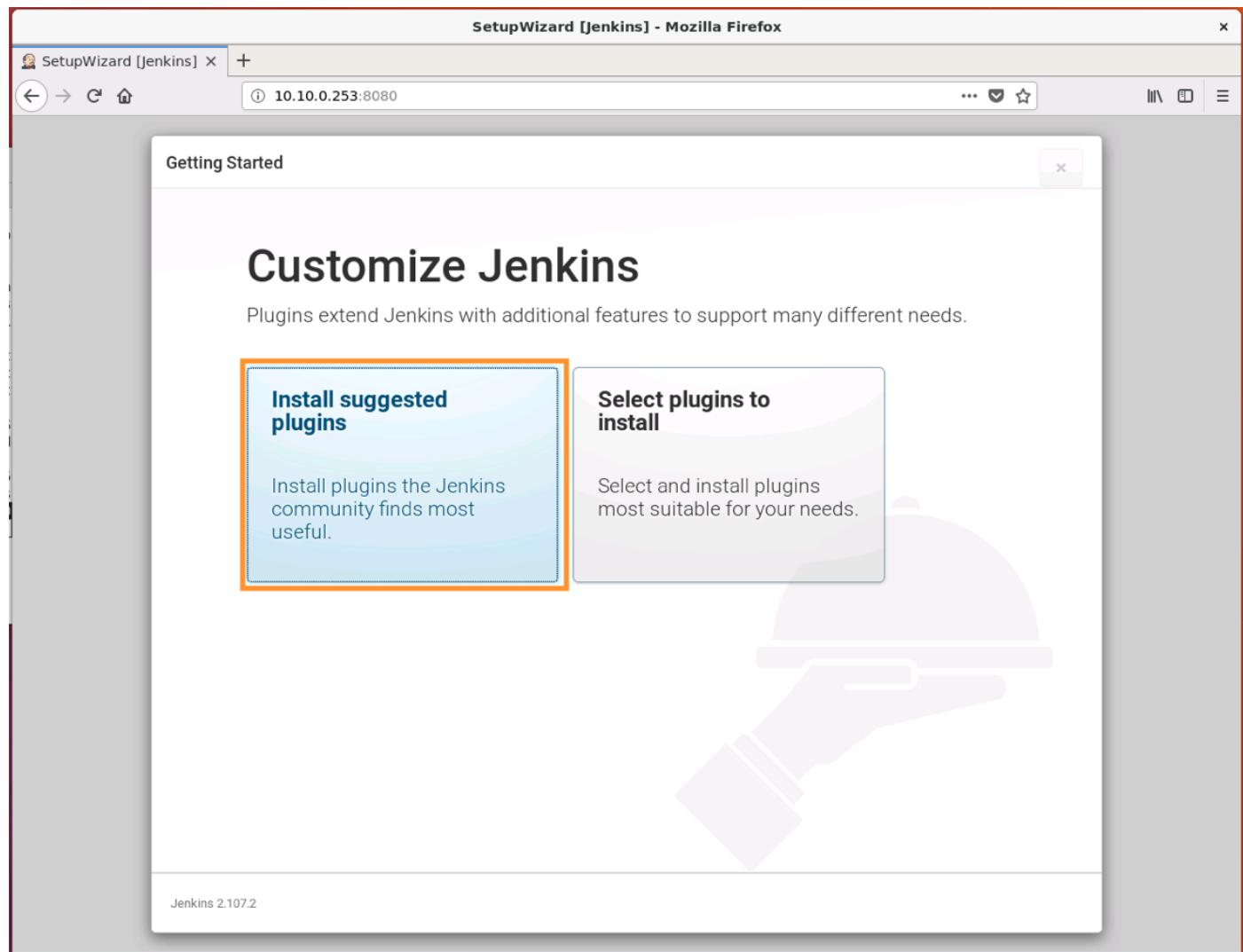
Last login: Sat May  5 17:30:15 2018 from 10.10.0.254
ignw@ignw-jenkins:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
02840833e26f430bac2fa038915c122b
ignw@ignw-jenkins:~$
```

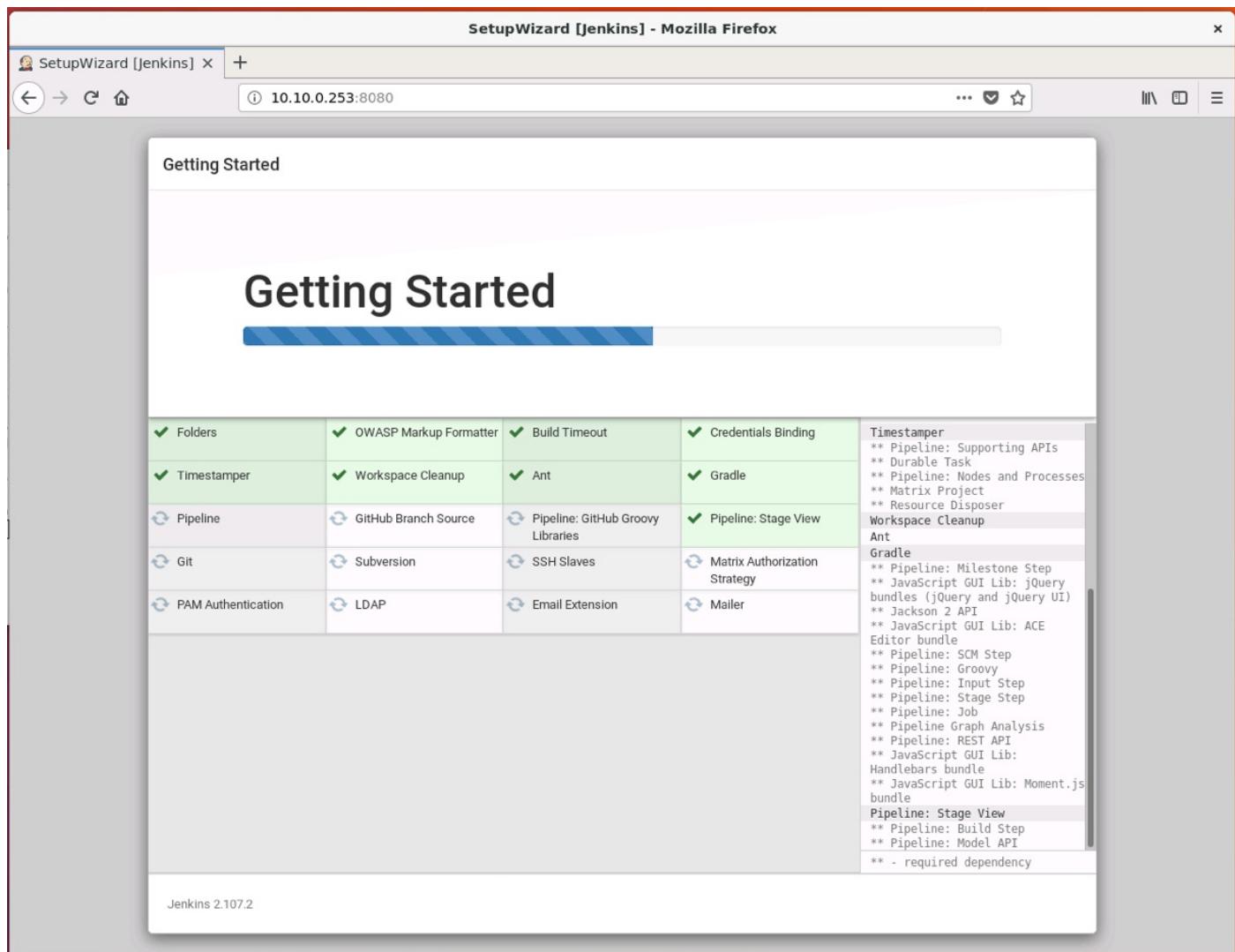
Copy the password and paste it into the Jenkins prompt, then click "Continue":



## Plugins

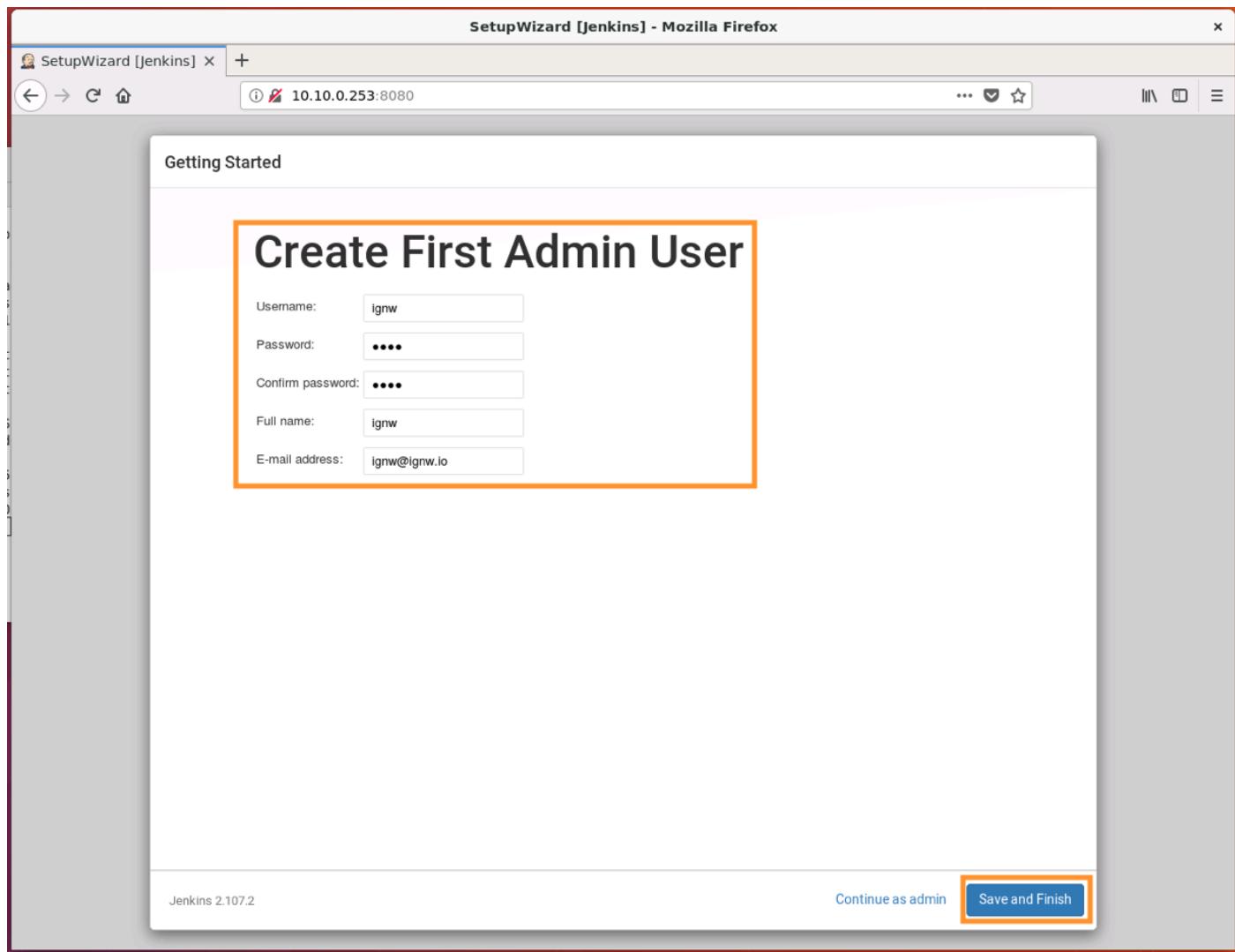
Jenkins, having been around a while, has the concept of "Plugins" to help extend its functionality. Rather than manually picking plugins to install we'll simply install the suggested plugins, these should cover most if not all of what we will need, and of course plugins can always be added later.



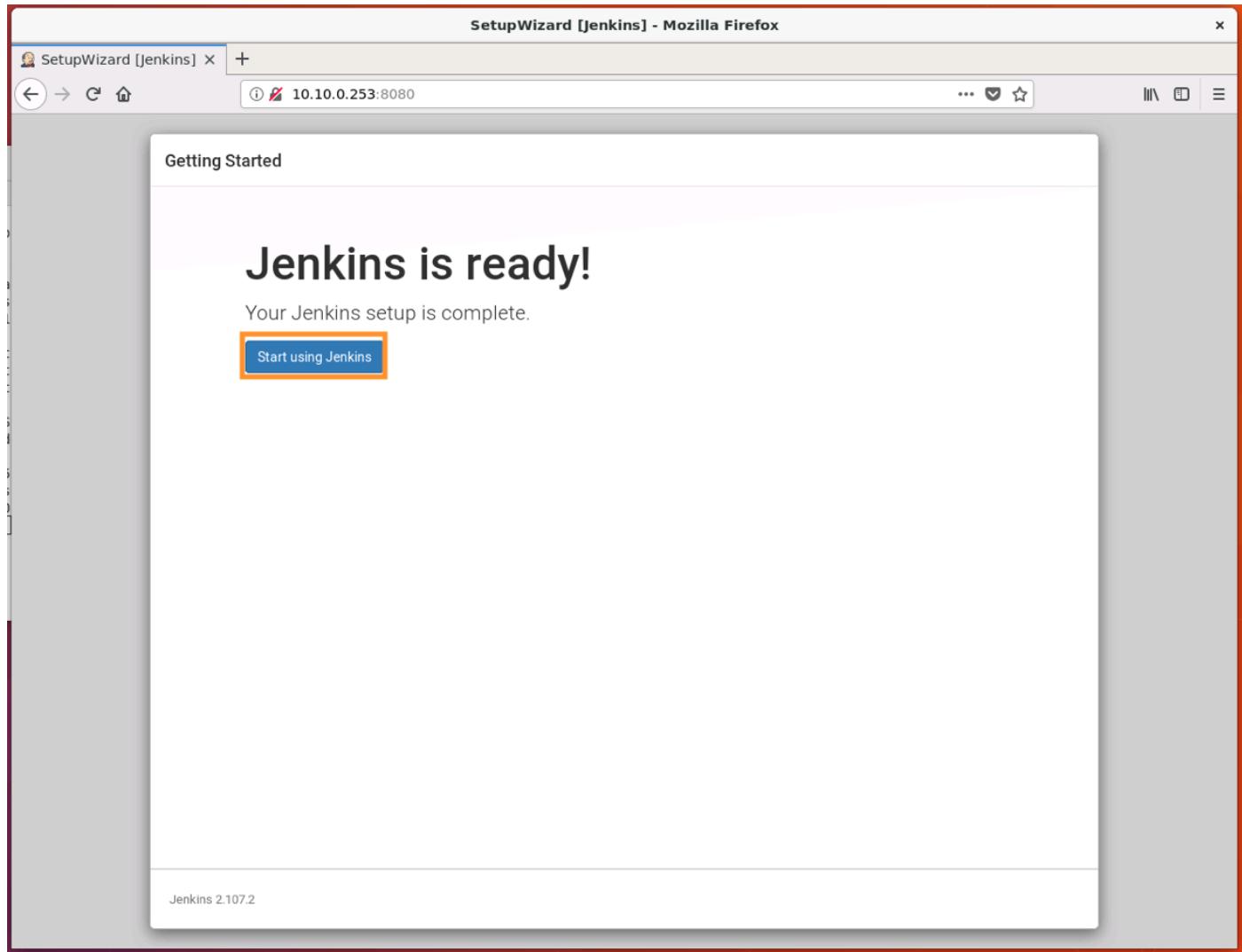


**Note** If you receive an error such as: "An error occurred during installation: No such plugin: cloudbees-folder" -- SSH to the Jenkins host and restart the service "sudo service jenkins restart".

Once Jenkins has completed installing plugins for us it will prompt us to create our first admin user. Enter whatever you would like in these fields, but don't forget your password! We suggest simply using "ignw" as the username and password to keep things simple! Once complete, click "Save and Finish".



Click "Start using Jenkins" to wrap up the initial setup.



Dashboard [Jenkins] - Mozilla Firefox

Dashboard [Jenkins] + 10.10.0.253:8080 search ... Ignw | log out ENABLE AUTO REFRESH

# Jenkins

Jenkins > [New Item](#) [add description](#)

New Item People Build History Manage Jenkins My Views Credentials New View

Welcome to Jenkins!

Please [create new jobs](#) to get started.

**Build Queue** No builds in the queue.

**Build Executor Status** 1 Idle 2 Idle

Page generated: May 5, 2018 6:03:43 PM UTC [REST API](#) Jenkins ver. 2.107.2

The screenshot shows the Jenkins dashboard interface. At the top, there's a header bar with the Jenkins logo, the URL '10.10.0.253:8080', a search bar, and navigation links for 'Ignw' and 'log out'. Below the header is a main menu with links for 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', 'Credentials', and 'New View'. A large central area displays a 'Welcome to Jenkins!' message with a call to action to 'create new jobs'. Below this are two collapsed sections: 'Build Queue' (showing 'No builds in the queue.') and 'Build Executor Status' (listing '1 Idle' and '2 Idle'). At the bottom of the page, there's a footer with the generation timestamp 'May 5, 2018 6:03:43 PM UTC', links to 'REST API' and 'Jenkins ver. 2.107.2', and a redacted URL.

# Creating your first Pipeline

## Overview and Objectives

In this lab you will create your first Pipeline. This is a very simple Pipeline that really won't do much! The primary purpose is for you to understand the basics of a Pipeline as we set the stage for building out CI/CD for the networking devices.

Objectives:

- Setup Jenkins to be "hooked" to a GitHub repository
- Create a basic multi-stage Pipeline in Jenkins

## What are Pipelines?

Now that we've got Jenkins ready to go we need to learn a bit about Pipelines. We'll start off by making a very simple repository and getting it "hooked" to Jenkins.

Start off by creating a new folder for our repository:

```
cd ~/  
mkdir my_first_pipeline
```

Create a README file in there and put something witty in it as a place holder:

```
cd my_first_pipeline  
touch README.md  
echo >> "got a serious case of the mondays...."
```

We can now initialize the repo, add our README, and make our commit:

```
git init  
git add README.md  
git commit -m 'first commit'
```

As you've done before, log into your GitHub account and create your repository -- make sure to leave the "initialize this repository with a README" *unticked*.

Finally, connect the local repo to GitHub and push:

```
git remote add origin git@github.com:[yourusername]/my_first_pipeline  
git push -u origin master
```

With that out of the way... head back to Jenkins. Click on the "create new jobs" button on the front page of your

Jenkins server.

create new jobs to get started.' This message is highlighted with an orange border. Below it are sections for 'Build Queue' (empty) and 'Build Executor Status' (1 Idle, 2 Idle). At the bottom right of the page is a footer with the text 'Page generated: May 5, 2018 6:03:43 PM UTC REST API Jenkins ver. 2.107.2'." data-bbox="64 115 922 623"/&gt;

There are several kinds of "jobs" that we can create with Jenkins depending on what exactly we want to do.

"Freestyle project" as the description indicates is a very flexible open-ended project type that can be integrated into any SCM (Source Control Management) system that Jenkins has a plugin for. This is a powerful tool but it is a bit more complicated/open-ended than what we need to do right now.

"Multi-configuration projects" are interesting in that they can be used for testing against a variety of conditions - such as different browsers or databases. This is probably also a bit too open-ended for what we need.

A "Folder" is pretty much what it sounds like, and not something we need at this point. As the description implies it creates a unique namespace so you can have multiple folders with the same named workflows inside of them for example.

A "GitHub Organization" provides a mechanism for scanning a GitHub organization (such as your place of work's GitHub account) for repos that match *something*. Again, not quite what we're looking for right now.

A "Multibranch" Pipeline will allow you to fire off builds based on any new branches you add to a project. Could be cool, but we don't need that functionality at this point.

The job type that we care about for now is the "Pipeline" job. As the name suggests the purpose of this job type is to build a pipeline or a work flow around a project, in our case around our Github repository.

## Create a Pipeline

Provide a name for your pipeline, select "Pipeline", and then click "OK".

The screenshot shows the Jenkins 'New Item' creation interface. At the top, there's a header bar with the title 'New Item [Jenkins] - Mozilla Firefox' and a URL '10.10.0.253:8080/newJob'. Below the header is the Jenkins logo and a search bar. The main content area has a form titled 'Enter an item name' with a text input field containing 'my\_first\_pipeline'. This input field is highlighted with an orange border. Below the input field is a note: '» Required field'. To the right of the input field is a 'Freestyle project' section with a description. Further down, a 'Pipeline' section is highlighted with an orange border; it contains a description and a note: 'Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.' Below the Pipeline section are other project types: 'Multi-configuration project', 'Folder', 'GitHub Organization', and 'Multibranch Pipeline'. At the bottom of the form is an 'OK' button, which is also highlighted with an orange border. The footer of the page includes a timestamp 'Page generated: May 5, 2018 7:23:00 PM UTC' and links to 'REST API' and 'Jenkins ver. 2.107.2'.

Jenkins will then prompt us for some information about the pipeline we are wanting to build. As you may be beginning to notice, Jenkins is a very, very powerful tool with a myriad of options/plugins available, this page is quite representative of that! There are tons of widgets available for us to set, however as we are building our very first pipeline, we'll keep things simple for now!

Select the "GitHub project" tick box and provide the URL for your project. We don't actually *need* to fill out this information as this section is informational, however it is a good habit to get into. Provide the URL to your Github repository to be used for this task.

my\_first\_pipeline Config [Jenkins] - Mozilla Firefox

my\_first\_pipeline Conf +

10.10.0.253:8080/job/my\_first\_pipeline/configure

Jenkins my\_first\_pipeline

General Build Triggers Advanced Project Options Pipeline

Pipeline name my\_first\_pipeline

Description

[Plain text] [Preview](#)

Discard old builds [?](#)

Do not allow concurrent builds [?](#)

Do not allow the pipeline to resume if the master restarts. [?](#)

GitHub project [?](#)

Project url  [?](#)

Display name  [?](#)

Pipeline speed/durability override [?](#)

This project is parameterized [?](#)

Throttle builds [?](#)

**Build Triggers**

Build after other projects are built [?](#)

Build periodically [?](#)

GitHub hook trigger for Git Scm polling [?](#)

**Save** **Apply**

Scroll down the page to the "Advanced Project Options" section. Notice the Pipeline section -- this is probably a pretty important section given that we are creating a project of type "Pipeline"! The default behavior is to configure our pipeline script here natively in Jenkins. Jenkins is even kind enough to provide us some example Pipeline script syntax. Click on the "Pipeline Syntax" link to check out what kind of options are available in our pipeline.

my\_first\_pipeline Config [Jenkins] - Mozilla Firefox

my\_first\_pipeline Config | +

10.10.0.253:8080/job/my\_first\_pipeline/configure

Jenkins > my\_first\_pipeline >

General Build Triggers Advanced Project Options Pipeline

Quiet period  Trigger builds remotely (e.g., from scripts)

**Advanced Project Options**

[Advanced...](#)

**Pipeline**

Definition Pipeline script

Script 1 try sample Pipeline...

Use Groovy Sandbox

[Pipeline Syntax](#)

[Save](#) [Apply](#)

Page generated: May 5, 2018 7:24:34 PM UTC REST API Jenkins ver. 2.107.2

The link will take you to the "Snippet Generator" section where we can view some sample steps that we may wish to take, and even turn that into executable pipeline script syntax. Take a look at the options available to us in the drop down "Sample Step" menu.

Pipeline Syntax: Snippet Generator [Jenkins] - Mozilla Firefox

my\_first\_pipeline Config Pipeline Syntax: Snippet Generator Jenkins 10.10.0.253:8080/job/my\_first\_pipeline/pipeline-syntax/ search Ignw | log out

## Jenkins

Back Snippet Generator Declarative Directive Generator Declarative Online Documentation Steps Reference Global Variables Reference Online Documentation IntelliJ IDEA GDSL

### Overview

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

#### Steps

Sample Step archiveArtifacts: Archive the artifacts

Files to archive Advanced...

Generate Pipeline Script

Global Variables

There are many features of the Pipeline that are not steps. These are often exposed via global variables, which are not supported by the snippet generator. See the [Global Variables Reference](#) for details.

Page generated: May 5, 2018 7:27:12 PM UTC Jenkins ver. 2.107.2

Since we're starting with a very simple example, let's just pick on the "echo: Print Message" option to see if we can get our Pipeline echoing out some text as our build is executed.

In the message section enter some message that you would like to have echo'd during your execution, then click the "Generate Pipeline Script" button.

Pipeline Syntax: Snippet Generator [Jenkins] - Mozilla Firefox

my\_first\_pipeline Confi Pipeline Syntax: Snipp +  
10.10.0.253:8080/job/my\_first\_pipeline/pipeline-syntax/ ... Ignw | log out

# Jenkins

Jenkins > my\_first\_pipeline > Pipeline Syntax

[Back](#)

**Snippet Generator**

- Declarative Directive Generator
- Declarative Online Documentation
- Steps Reference
- Global Variables Reference
- Online Documentation
- IntelliJ IDEA GDSL

**Overview**

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

**Steps**

Sample Step echo: Print Message

Message automate that &!%#& !!

**Generate Pipeline Script**

echo 'automate that &!%#& !!'

**Global Variables**

There are many features of the Pipeline that are not steps. These are often exposed via global variables, which are not supported by the snippet generator. See the [Global Variables Reference](#) for details.

Page generated: May 5, 2018 7:27:12 PM UTC Jenkins ver. 2.107.2

The screenshot shows the Jenkins Pipeline Syntax Snippet Generator page. On the left, there's a sidebar with links to other Jenkins features like Declarative Directive Generator and Steps Reference. The main area has a heading 'Overview' with a brief description of what the snippet generator does. Below that is a section titled 'Steps' containing a 'Sample Step' labeled 'echo: Print Message'. Underneath it is a 'Message' field with the value 'automate that &!%#& !!'. A large orange-bordered box contains a 'Generate Pipeline Script' button and the resulting pipeline script: 'echo \'automate that &!%#& !!\''. At the bottom, there's a 'Global Variables' section with a note that global variables are not supported.

Well that was kinda uneventful, but that's OK! It turns out that echoing things during our job execution is pretty much as simple as you may have thought! Returning to your pipeline build, change the pipeline "definition" to "Pipeline script from SCM". This basically means that Jenkins will look into our repository (SCM = Source Control Management, so in our case Github) to find a script that represents our pipeline.

my\_first\_pipeline Config [Jenkins] - Mozilla Firefox

my\_first\_pipeline Config | +

10.10.0.253:8080/job/my\_first\_pipeline/configure

Jenkins > my\_first\_pipeline >

General Build Triggers Advanced Project Options Pipeline

Build after other projects are built

Build periodically

GitHub hook trigger for GITScm polling

Poll SCM

Disable this project

Quiet period

Trigger builds remotely (e.g., from scripts)

Advanced Project Options

Advanced...

Pipeline

Definition: Pipeline script from SCM

SCM: None

Script Path: Jenkinsfile

Lightweight checkout:

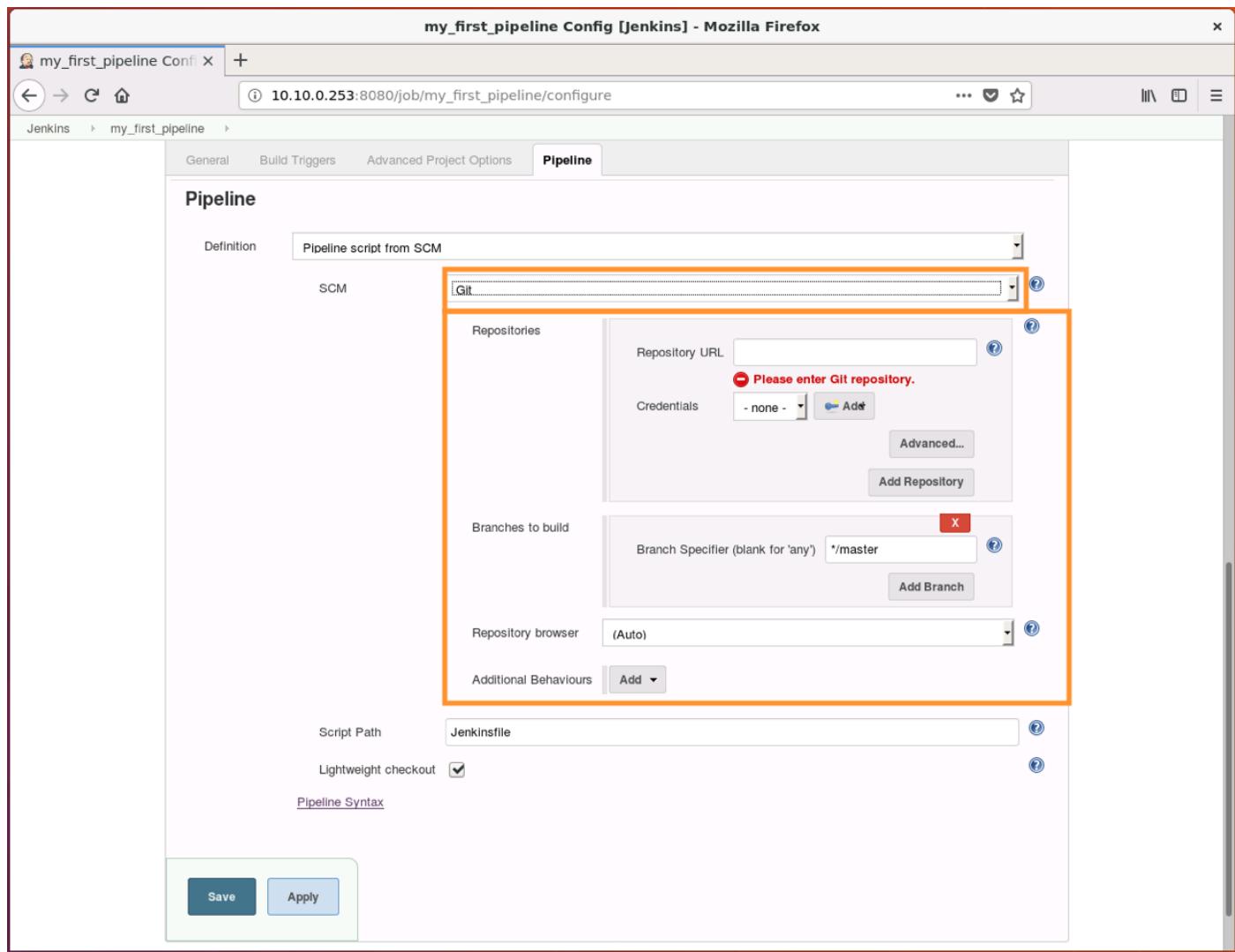
[Pipeline Syntax](#)

Save Apply

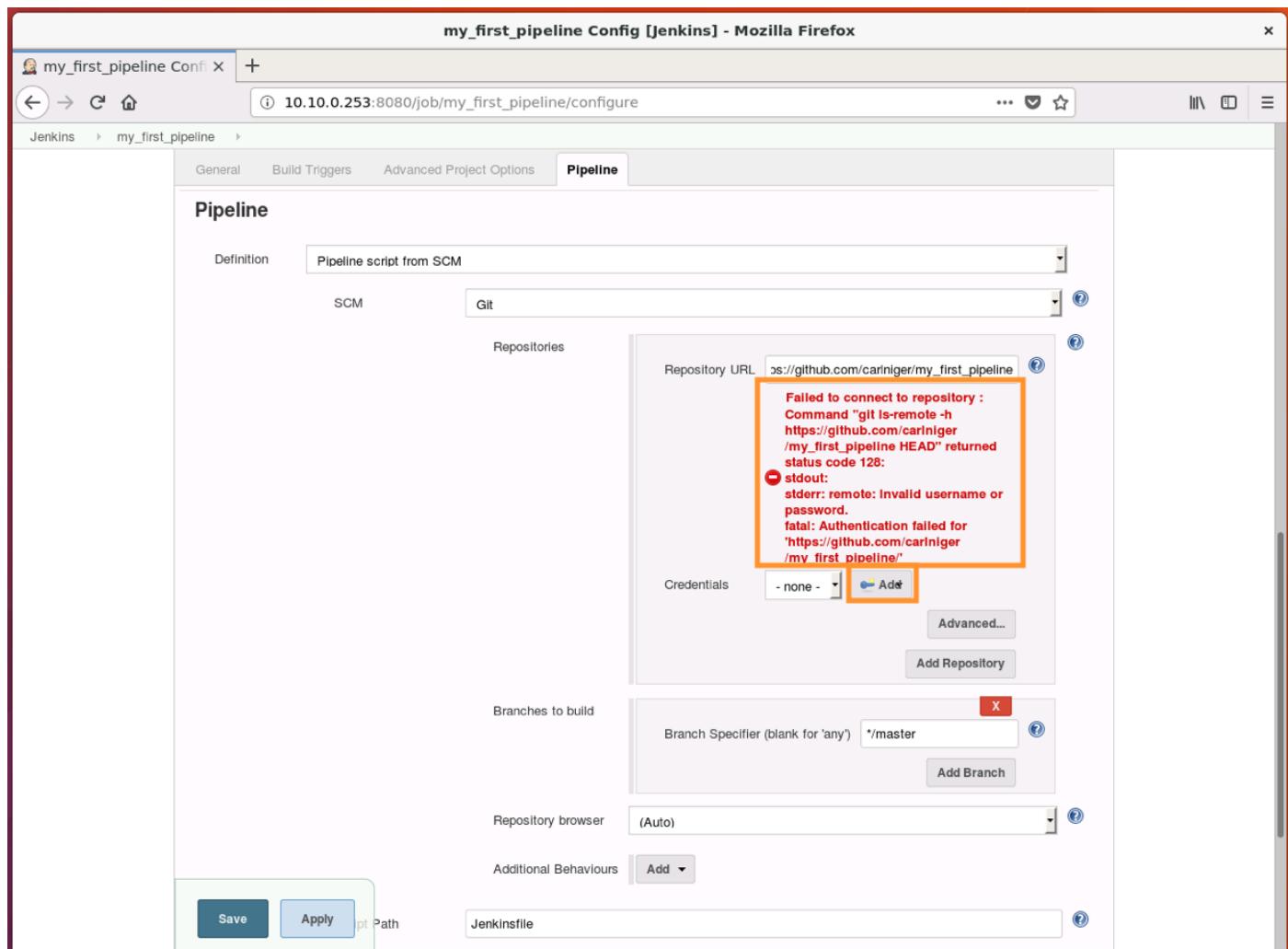
Page generated: May 5, 2018 7:24:34 PM UTC REST API Jenkins ver. 2.107.2

## Connecting to GitHub

Next we need to tell Jenkins that we are in fact going to be using Github (git), so select "Git" from the SCM drop down section.

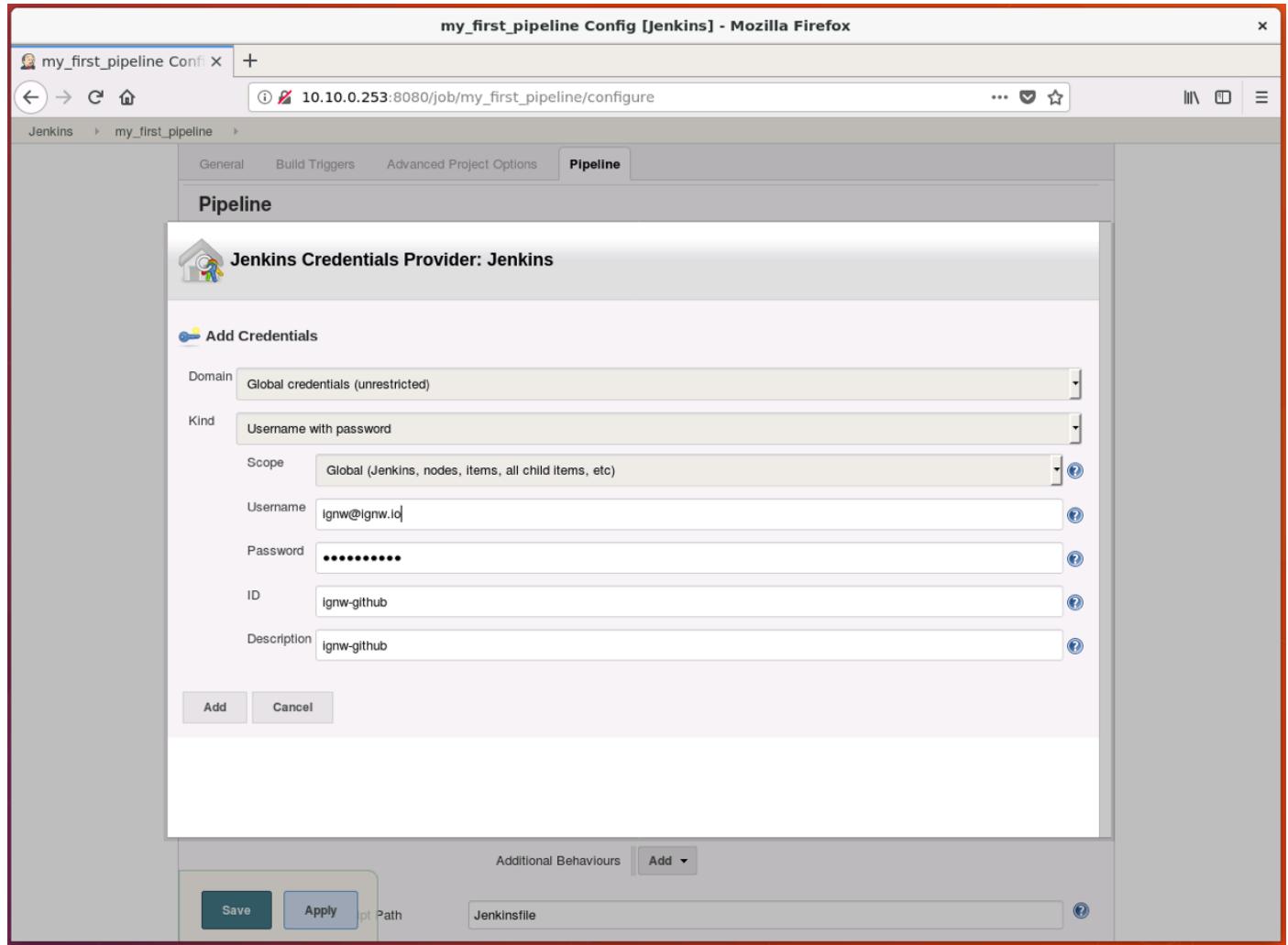


We now need to let Jenkins know where our repository lives, snag the URL of your Github repository and paste it into the "Repository URL" field.



Your repository is (probably) a public repository, but in many cases you'll be working with private repositories, in that case you would see a message that Jenkins "Failed to connect to repository...".

Even if you are working with a public repository, go ahead and click on the "Add" button to add some credentials for Jenkins to connect to Github. In the pop up, enter your Github credentials and click "Add".



Now that Jenkins is connected to Github and can poke our repository we need to let it know which branches to "look" at. The default in Jenkins is to look for the "master" branch. For our purposes this will do just fine.

The screenshot shows the Jenkins 'Advanced Project Options' configuration page for a project named 'my\_first\_pipeline'. The 'Pipeline' tab is selected. In the 'Definition' section, 'Pipeline script from SCM' is chosen. Under 'SCM', 'Git' is selected. The 'Repositories' section contains a repository URL 'https://github.com/carlnlger/my\_first\_pipeline' and a credentials dropdown set to 'ignw@ignw.io/\*\*\*\*(ignw-github)'. The 'Branches to build' section shows a branch specifier '\*/master'. The 'Script Path' is set to 'Jenkinsfile'. At the bottom, there are 'Save' and 'Apply' buttons, with 'Save' being disabled.

Finally, Jenkins needs to know what the "Script Path" is -- in other words where is the file that is outlining the pipeline in your repository. It is a very common convention to simply call this file "Jenkinsfile" and to leave it in the root directory of your repository. The Jenkins default settings reflect this. That will work just fine for us.

Click "Save" to wrap up the initial creation of your pipeline.

my\_first\_pipeline Config [Jenkins] - Mozilla Firefox

my\_my\_first\_pipeline Config | System » Global cred... | +  
10.10.0.253:8080/job/my\_first\_pipeline/configure

Jenkins > my\_first\_pipeline >

General Build Triggers Advanced Project Options Pipeline

Repositories

Repository URL: [https://github.com/carligner/my\\_first\\_pipeline](https://github.com/carligner/my_first_pipeline)

Credentials: carligner@gmail.com/\*\*\*\*\* (ign)

Add Advanced... Add Repository

Branches to build

Branch Specifier (blank for 'any'): \*/master

Add Branch

Repository browser: (Auto)

Additional Behaviours: Add

Script Path: Jenkinsfile

Lightweight checkout:

Pipeline Syntax

Save Apply

Page generated: May 5, 2018 7:24:34 PM UTC REST API Jenkins ver. 2.107.2

The screenshot shows the Jenkins Pipeline configuration interface for a project named 'my\_first\_pipeline'. The 'Pipeline' tab is active. In the 'Repositories' section, a GitHub repository is specified with the URL 'https://github.com/carligner/my\_first\_pipeline' and a credential named 'carligner@gmail.com/\*\*\*\*\* (ign)'. The 'Branches to build' section shows a branch specifier of '\*/master'. The 'Script Path' is set to 'Jenkinsfile'. At the bottom, there are 'Save' and 'Apply' buttons, which are both highlighted with a red box.

## First Build

Jenkins will bring us to what is effectively the "home page" for our newly constructed pipeline. We can see that there isn't a lot going on at this point! The "Stage View" section is where we would normally see activity that has happened in our pipeline, but clearly we can just see that the "Pipeline has not yet run." Let's fix that by clicking the "Build Now" button.

my\_first\_pipeline [Jenkins] - Mozilla Firefox

my\_first\_pipeline [Jen Jenkins System > Global cred +  
10.10.0.253:8080/job/my\_first\_pipeline/ ... Ignw | log out  
Jenkins my\_first\_pipeline ENABLE AUTO REFRESH

Back to Dashboard Status Changes Build Now Delete Pipeline Configure Full Stage View GitHub Pipeline Syntax

Pipeline my\_first\_pipeline

Recent Changes

Stage View

No data available. This Pipeline has not yet run.

Build History trend = find RSS for all RSS for failures

Page generated: May 5, 2018 7:52:13 PM UTC REST API Jenkins ver. 2.107.2

The screenshot shows the Jenkins interface for the 'my\_first\_pipeline' job. The left sidebar contains navigation links: Back to Dashboard, Status, Changes, Build Now (which is highlighted with a red box), Delete Pipeline, Configure, Full Stage View, GitHub, and Pipeline Syntax. The main content area is titled 'Pipeline my\_first\_pipeline'. It features a 'Recent Changes' section with a notebook icon and a 'Stage View' section with an orange border containing the message 'No data available. This Pipeline has not yet run.'. Below the Stage View is a 'Permalinks' section with RSS links for 'RSS for all' and 'RSS for failures'. At the bottom, there's a 'Build History' section with a search bar and a 'trend =' dropdown, along with links for 'RSS for all' and 'RSS for failures'. The footer includes a timestamp 'Page generated: May 5, 2018 7:52:13 PM UTC', a 'REST API' link, and 'Jenkins ver. 2.107.2'.

After a moment Jenkins should refresh and you should now have an entry in the "Build History" section.

my\_first\_pipeline [Jenkins] - Mozilla Firefox

my\_first\_pipeline [Jenl] +  
10.10.0.253:8080/job/my\_first\_pipeline/

Jenkins | log out

Back to Dashboard | Status | Changes | Build Now | Delete Pipeline | Configure | Full Stage View | GitHub | Pipeline Syntax

## Pipeline my\_first\_pipeline

Recent Changes

Stage View

No data available. This Pipeline has not yet run.

Build History

find

#1 May 5, 2018 7:52 PM

RSS for all RSS for failures

Permalinks

- Last build (#1), 16 sec ago
- Last failed build (#1), 16 sec ago
- Last unsuccessful build (#1), 16 sec ago
- Last completed build (#1), 16 sec ago

Page generated: May 5, 2018 7:53:03 PM UTC REST API Jenkins ver. 2.107.2

Red is probably not a good sign though! Click on your build in the "Build History" section to get some more details. On the build details page, click on the "Console Output" to see if we can determine what is going on:

The screenshot shows a Jenkins job named "my\_first\_pipeline" with build number #1. The "Console Output" link in the left sidebar is highlighted with a red box. The main content area displays the console output, which starts with an exception stack trace:

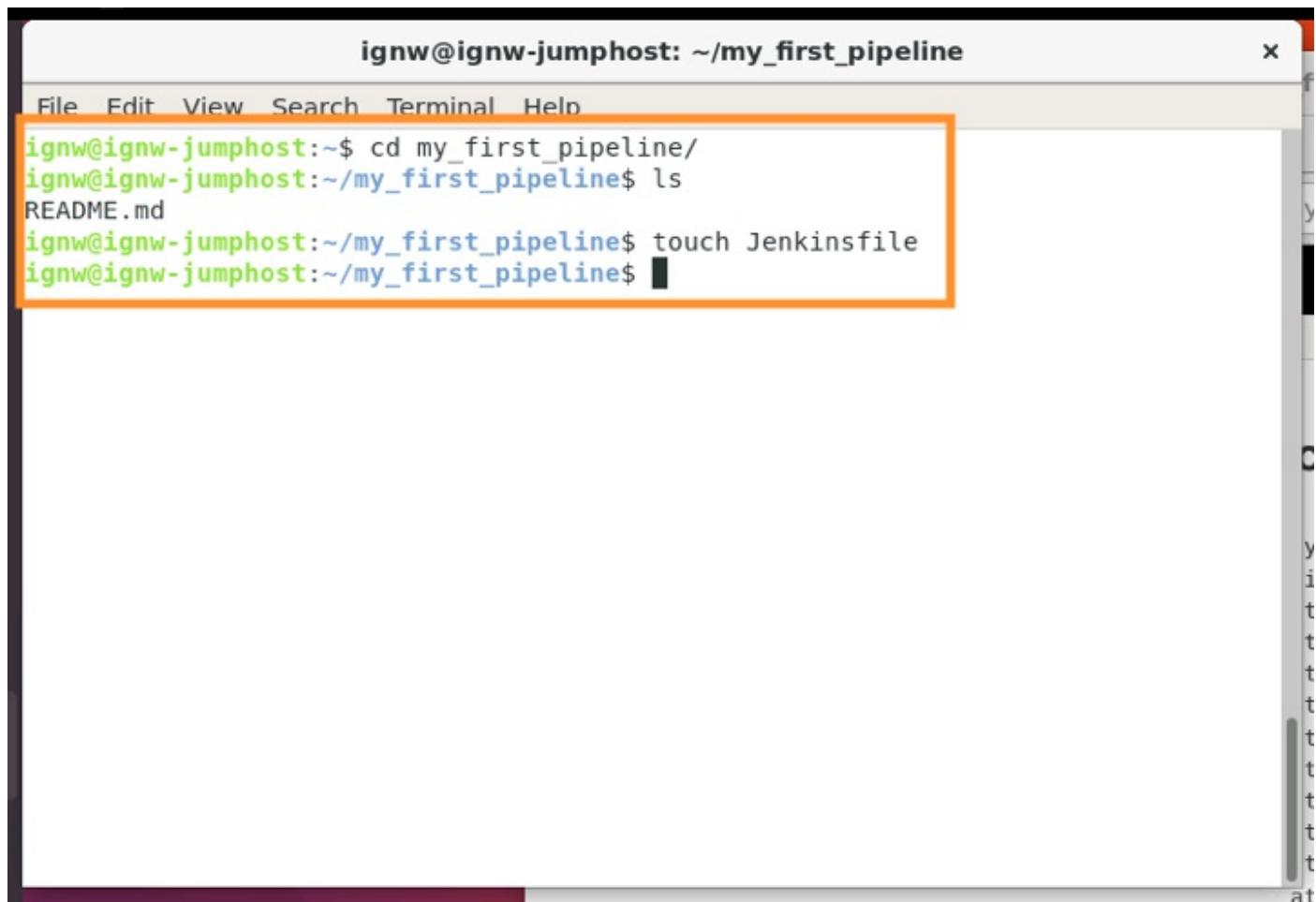
```
Started by user ignw
java.io.FileNotFoundException
    at jenkins.plugins.git.GitSCMFile$3.invoke(GitSCMFile.java:167)
    at jenkins.plugins.git.GitSCMFile$3.invoke(GitSCMFile.java:159)
    at jenkins.plugins.git.GitSCMfileSystem$3.invoke(GitSCMfileSystem.java:193)
    at org.jenkinsci.plugins.gitclient.AbstractGitAPIImpl.withRepository(AbstractGitAPIImpl.java:29)
    at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.withRepository(CliGitAPIImpl.java:72)
    at jenkins.plugins.git.GitSCMfileSystem.invoke(GitSCMfileSystem.java:189)
    at jenkins.plugins.git.GitSCMFile.content(GitSCMFile.java:159)
    at jenkins.scm.api.SCMFile.contentAsString(SCMFile.java:338)
    at org.jenkinsci.plugins.workflow.cps.CpsScmFlowDefinition.create(CpsScmFlowDefinition.java:110)
    at org.jenkinsci.plugins.workflow.cps.CpsScmFlowDefinition.create(CpsScmFlowDefinition.java:67)
    at org.jenkinsci.plugins.workflow.job.WorkflowRun.run(WorkflowRun.java:298)
    at hudson.model.ResourceController.execute(ResourceController.java:97)
    at hudson.model.Executor.run(Executor.java:429)
Finished: FAILURE
```

At the bottom of the page, there is a footer note: "Page generated: May 5, 2018 7:53:33 PM UTC REST API Jenkins ver. 2.107.2".

We can pretty clearly see that our build finished in a "FAILURE" state, but why? What do we know about how we setup our build? Jenkins is trying to look in our repository for a "Jenkinsfile", but we've yet to create one. That would certainly jive with the "FileNotFoundException" that we see at the top of the console output.

## The Jenkinsfile

Open up a terminal window and change directory to your git repository. At this point there should only be the README.md file that we created when initializing the repository. Create a new file in your repository called "Jenkinsfile"



A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/my\_first\_pipeline". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal content is as follows:

```
ignw@ignw-jumphost:~$ cd my_first_pipeline/
ignw@ignw-jumphost:~/my_first_pipeline$ ls
README.md
ignw@ignw-jumphost:~/my_first_pipeline$ touch Jenkinsfile
ignw@ignw-jumphost:~/my_first_pipeline$
```

The Jenkinsfile follows a pretty straight forward format, and the Jenkins website contains some great starting examples from which we can steal! Head on over to the [Jenkins examples page here](#) to see some examples of Jenkinsfiles.

The "Artifactory Gradle Build" example, while having nothing really to do with what we are trying to accomplish, is a nice example that we can use as the basis for our Jenkinsfile.

```
node {  
    // Get Artifactory server instance, defined in the Artifactory Plugin administration page.  
    def server = Artifactory.server "SERVER_ID"  
    // Create an Artifactory Gradle instance.  
    def rtGradle = Artifactory.newGradleBuild()  
    def buildInfo  
  
    stage('Clone sources') {  
        git url: 'https://github.com/jfrogdev/project-examples.git'  
    }  
  
    stage('Artifactory configuration') {  
        // Tool name from Jenkins configuration  
        rtGradle.tool = "Gradle-2.4"  
        // Set Artifactory repositories for dependencies resolution and artifacts deployment.  
        rtGradle.deployer repo:'ext-release-local', server: server  
        rtGradle.resolver repo:'remote-repos', server: server  
    }  
  
    stage('Gradle build') {  
        buildInfo = rtGradle.run rootDir: "gradle-examples/4/gradle-example-ci-server/", buildFile: 'build.gradle'  
    }  
  
    stage('Publish build info') {  
        server.publishBuildInfo buildInfo  
    }  
}
```

## Artifactory Maven Build

One of the first things we see from this example is that the Jenkinsfile starts with a "node" declaration:

```
node {  
    STUFF GOES HERE  
}
```

It also looks like within the node, stages are defined:

```
stage( 'My Stage '){  
    STUFF GOES HERE  
}
```

We also know about some of the things that we can put in the Jenkinsfile thanks to the built in Jenkinsfile syntax checker that we looked at earlier. We'll create a very simple Jenkinsfile that has only a single stage and simply echos some text at the execution of that stage:

```
node {
    stage ('stage1') {
        sh 'echo "automate that &!%#& !!!"'
    }
}
```

ignw@ignw-jumphost: ~/my\_first\_pipeline

File Edit View Search Terminal Help

```
node {
    stage ('stage1') {
        sh 'echo "automate that &!%#& !!"'
    }
}
```

"Jenkinsfile" 7 lines, 76 characters

Now that we've created our Jenkinsfile we need to ensure that it is being tracked by Git, and then ultimately push it up to Github. First, we'll track it with the "add" command, then we can commit our changes and add a simple description to the commit.

```
git add Jenkinsfile  
git commit -m "adding Jenkinsfile"
```

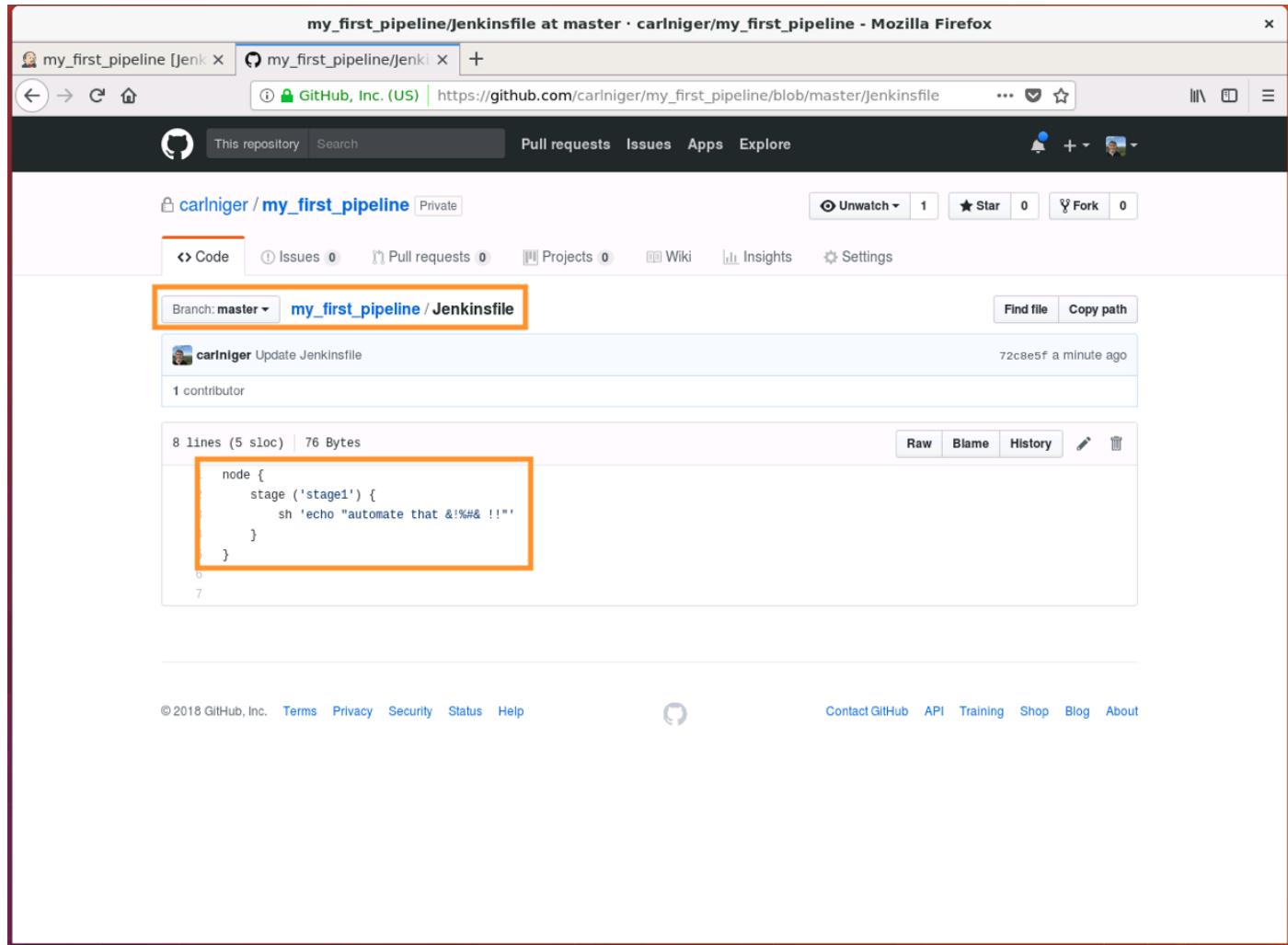
```
ignw@ignw-jumphost: ~/my_first_pipeline
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_first_pipeline$ vi Jenkinsfile
ignw@ignw-jumphost:~/my_first_pipeline$ git add Jenkinsfile
ignw@ignw-jumphost:~/my_first_pipeline$ git commit -m "adding Jenkinsfile"
[master 976b8e8] adding Jenkinsfile
 1 file changed, 5 insertions(+)
 create mode 100644 Jenkinsfile
ignw@ignw-jumphost:~/my_first_pipeline$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 343 bytes | 343.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:carlniger/my_first_pipeline
 8b2e946..976b8e8  master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_first_pipeline$ █
```

Next we can go ahead and push our changes up to our master branch on Github:

```
git push -u origin master
```

```
ignw@ignw-jumphost: ~/my_first_pipeline
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_first_pipeline$ vi Jenkinsfile
ignw@ignw-jumphost:~/my_first_pipeline$ git add Jenkinsfile
ignw@ignw-jumphost:~/my_first_pipeline$ git commit -m "adding Jenkinsfile"
[master 976b8e8] adding Jenkinsfile
 1 file changed, 5 insertions(+)
 create mode 100644 Jenkinsfile
ignw@ignw-jumphost:~/my_first_pipeline$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 343 bytes | 343.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:carlniger/my_first_pipeline
 8b2e946..976b8e8  master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_first_pipeline$ █
```

Now if you check out your Github repository, our newly created Jenkinsfile should be safely in Github!



The screenshot shows a GitHub repository page for 'carlniger/my\_first\_pipeline'. The 'Jenkinsfile' is displayed, showing the following Jenkins pipeline script:

```
node {
    stage ('stage1') {
        sh 'echo "automate that &!%## !!"'
    }
}
```

## Build Output

Back in the Jenkins server, click on "my\_first\_pipeline" from the menu bar to go back to the "home page" for our pipeline.

my\_first\_pipeline #1 Console [Jenkins] - Mozilla Firefox

my\_first\_pipeline #1 my\_first\_pipeline/jenki ... +  
10.10.0.253:8080/job/my\_first\_pipeline/1/console

Jenkins my\_first\_pipeline #1

Back to Project Status Changes Console Output View as plain text Edit Build Information Delete Build Replay Pipeline Steps

## Console Output

```
Started by user ignw
java.io.FileNotFoundException
    at jenkins.plugins.git.GitSCMFile$3.invoke(GitSCMFile.java:167)
    at jenkins.plugins.git.GitSCMFile$3.invoke(GitSCMFile.java:159)
    at jenkins.plugins.git.GitSCMFileSystem$3.invoke(GitSCMFileSystem.java:193)
    at org.jenkinsci.plugins.gitclient.AbstractGitAPIImpl.withRepository(AbstractGitAPIImpl.java:29)
    at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.withRepository(CliGitAPIImpl.java:72)
    at jenkins.plugins.git.GitSCMFileSystem.invoke(GitSCMFileSystem.java:189)
    at jenkins.plugins.git.GitSCMFile.content(GitSCMFile.java:159)
    at jenkins.scm.api.SCMFile.contentAsString(SCMFile.java:338)
    at org.jenkinsci.plugins.workflow.cps.CpsScmFlowDefinition.create(CpsScmFlowDefinition.java:110)
    at org.jenkinsci.plugins.workflow.cps.CpsScmFlowDefinition.create(CpsScmFlowDefinition.java:67)
    at org.jenkinsci.plugins.workflow.job.WorkflowRun.run(WorkflowRun.java:298)
    at hudson.model.ResourceController.execute(ResourceController.java:97)
    at hudson.model.Executor.run(Executor.java:429)
Finished: FAILURE
```

Page generated: May 5, 2018 7:53:33 PM UTC REST API Jenkins ver. 2.107.2

Click on "Build Now" to try to run your build again now that we have a Jenkinsfile in our repository.

my\_first\_pipeline [Jenkins] - Mozilla Firefox

my\_first\_pipeline [Jenkins] my\_first\_pipeline/jenki... +  
10.10.0.253:8080/job/my\_first\_pipeline/

Jenkins my\_first\_pipeline

Back to Dashboard Status Changes **Build Now** Delete Pipeline Configure Full Stage View GitHub Pipeline Syntax

Pipeline my\_first\_pipeline

Recent Changes

Stage View

No data available. This Pipeline has not yet run.

Build History trend = find #1 May 5, 2018 7:52 PM RSS for all RSS for failures

Permalinks

- Last build (#1), 10 min ago
- Last failed build (#1), 10 min ago
- Last unsuccessful build (#1), 10 min ago
- Last completed build (#1), 10 min ago

Page generated: May 5, 2018 8:02:51 PM UTC REST API Jenkins ver. 2.107.2

The screenshot shows the Jenkins interface for the 'my\_first\_pipeline'. The left sidebar contains links like Back to Dashboard, Status, Changes, Build Now (which is highlighted with a red box), Delete Pipeline, Configure, Full Stage View, GitHub, and Pipeline Syntax. The main area is titled 'Pipeline my\_first\_pipeline' and shows a 'Stage View' section with the message 'No data available. This Pipeline has not yet run.' Below it is a 'Build History' section with a table showing four builds, all marked with a blue icon and labeled '#1'. The table includes columns for status (green), build number (#1), date (May 5, 2018 7:52 PM), and RSS links. At the bottom, there's a 'Permalinks' section with a list of four build links. The footer indicates the page was generated on May 5, 2018, at 8:02:51 PM UTC, and shows Jenkins version 2.107.2.

That looks a lot better! It looks like our stage "stage1" completed in 307ms, and we have a blue icon instead of the failed red icon on our build history! Click on the "#2" item in the Build History section.

Click on the "Console Output" so we can see what Jenkins is doing during our build execution.

my\_first\_pipeline #2 [Jenkins] - Mozilla Firefox

my\_first\_pipeline #2 | my\_first\_pipeline/jenkins | +  
10.10.0.253:8080/job/my\_first\_pipeline/5/

Jenkins | log out | search | ... | enable auto refresh

Back to Project | Status | Changes | **Console Output** (highlighted with a red box) | Edit Build Information | Delete Build | Replay | Pipeline Steps | Previous Build

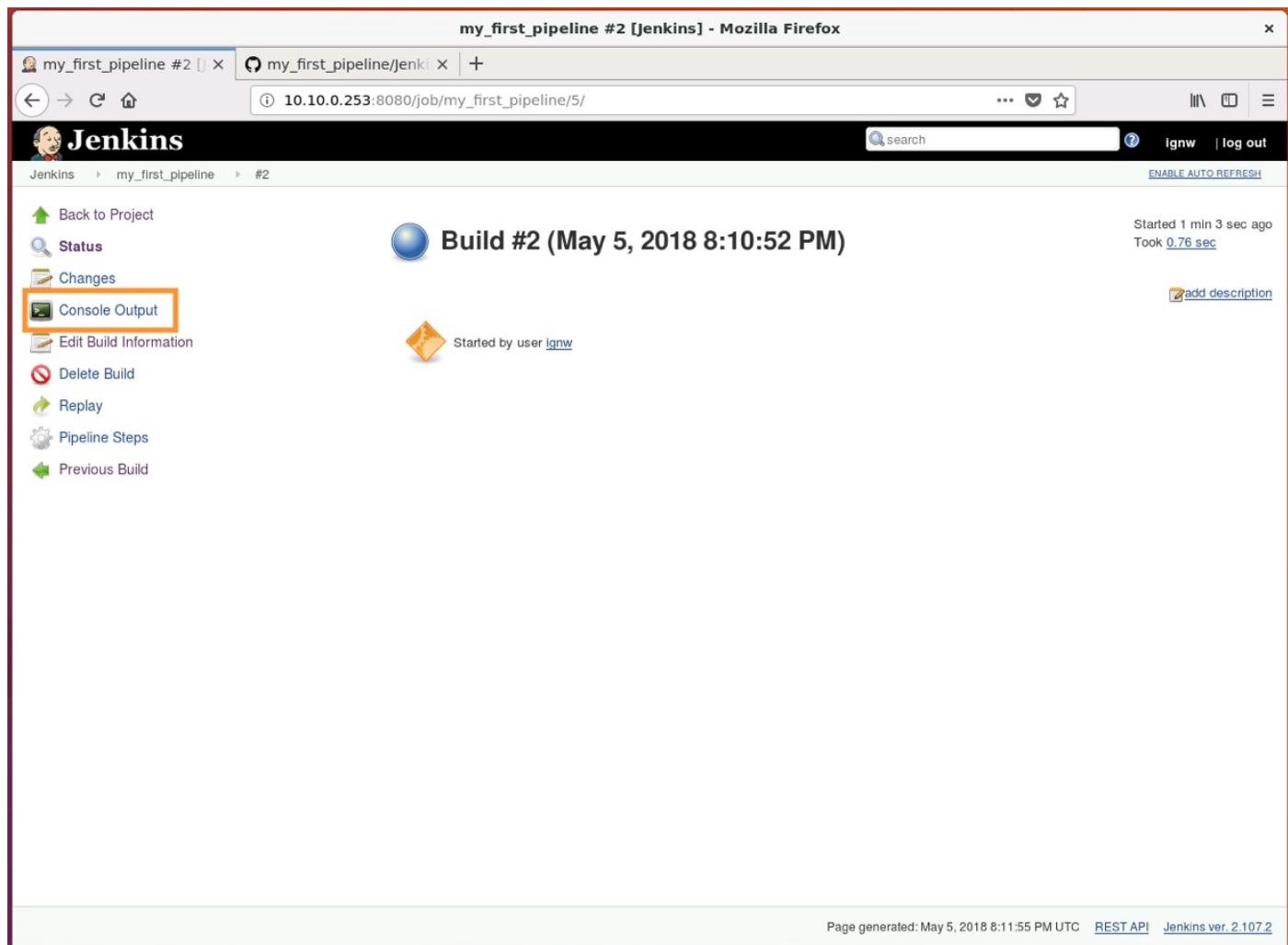
Build #2 (May 5, 2018 8:10:52 PM)

Started by user lgnw

Started 1 min 3 sec ago  
Took 0.76 sec

add description

Page generated: May 5, 2018 8:11:55 PM UTC REST API Jenkins ver. 2.107.2



Once again, blue is much better than red, and no FAILURE messages also seems pretty good!! It even looks like Jenkins printed our "automate that &!%#& !!" message to the console!

my\_first\_pipeline #2 Console [Jenkins] - Mozilla Firefox

my\_first\_pipeline #2 C X my\_first\_pipeline/jenki... +  
10.10.0.253:8080/job/my\_first\_pipeline/5/console

Jenkins

Back to Project Status Changes Console Output View as plain text Edit Build Information Delete Build Replay Pipeline Steps Previous Build

## Console Output

```
Started by user ignw
Obtained Jenkinsfile from git https://github.com/carlniger/my\_first\_pipeline
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/my_first_pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (stage1)
[Pipeline] sh
[my_first_pipeline] Running shell script
+ echo automate that &!%#& !!
automate that &!%#& !!
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Page generated: May 5, 2018 8:12:28 PM UTC [REST API](#) [Jenkins ver. 2.107.2](#)

Not too shabby for our first pipeline! Pipelines *usually* contain more than one stage though, so let's take it just one step further and try to add two more stages to our pipeline. For now we can simply echo more things to the console output, but in future tasks we'll be making good use of each stage!

Edit your Jenkins file to contain two more stages; echo whatever messages you'd like at each stage. Commit your code to Github, and then try to run your build again. Does it work?

my\_first\_pipeline #14 Console [Jenkins] - Mozilla Firefox

my\_first\_pipeline #14 +  
10.10.0.253:8080/job/my\_first\_pipeline/14/console

Jenkins | log out

Back to Project | Status | Changes | **Console Output** | View as plain text | Edit Build Information | Delete Build | Replay | Pipeline Steps | Previous Build

## Console Output

```
Started by user ignw
Obtained Jenkinsfile from git https://github.com/carlniger/my\_first\_pipeline
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/my_first_pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (stage1)
[Pipeline] sh
[my_first_pipeline] Running shell script
+ echo automate that &!%#& !!
automate that &!%#& !!
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] { (stage2)
[Pipeline] sh
[my_first_pipeline] Running shell script
+ echo hey, this is stage two folks!
hey, this is stage two folks!
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (stage3)
[Pipeline] sh
[my_first_pipeline] Running shell script
+ echo enough with the echoing lets do some cool stuff!
enough with the echoing lets do some cool stuff!
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

```
ignw@ignw-jumphost: ~/my_first_pipeline
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_first_pipeline$ cat Jenkinsfile
node {
    stage ('stage1') {
        sh 'echo "automate that &!%#& !!"'
    }
    stage ('stage2') {
        sh 'echo "hey, this is stage two folks!"'
    }
    stage ('stage3') {
        sh 'echo "enough with the echoing lets do some cool stuff!"'
    }
}
ignw@ignw-jumphost:~/my_first_pipeline$
```

# Network Configurations in Git

## Overview and Objectives

In this lab you will create your migrate "flat" network configurations into a GitHub repository. In later labs we will be heavily using GitHub as part of our NetOps process, so this is the first step on that journey. You will also write a simple Python script introducing you to the NAPALM module. We will be using NAPALM much more as we continue (through Ansible) for configuration diffs/merges/replacement.

Objectives:

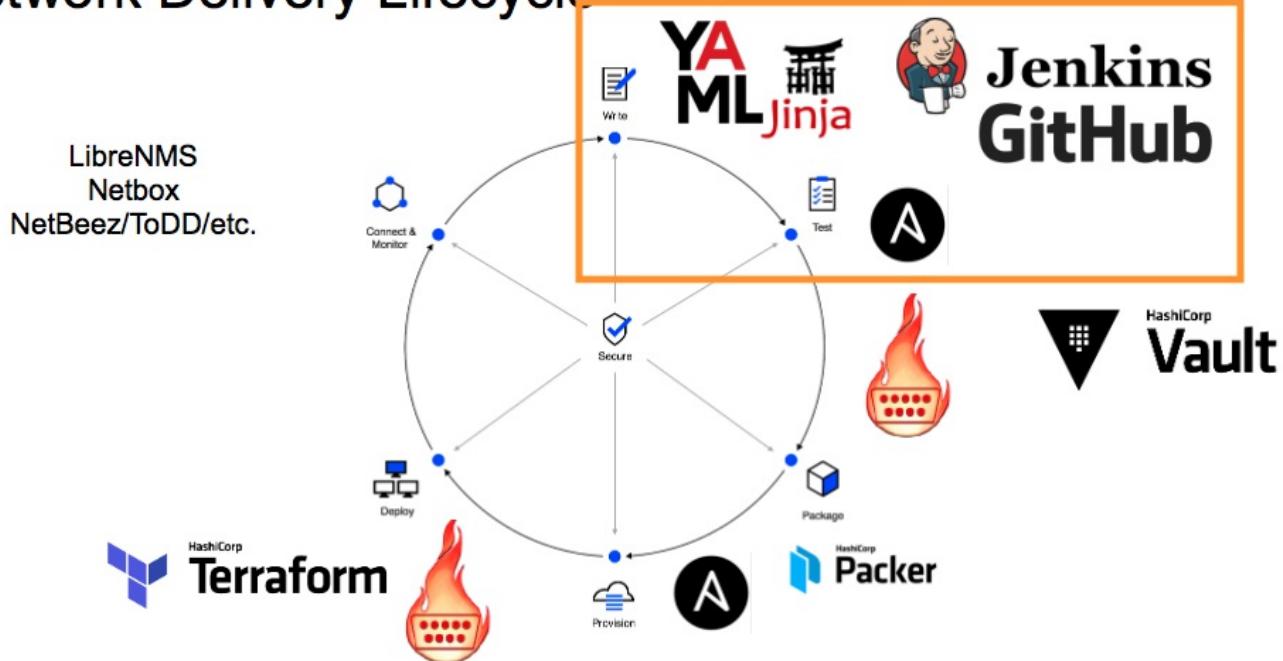
- Setup your GitHub repository for "my\_network\_as\_code"
- Push configuration files into GitHub
- Write a simple Python script to get acquainted with NAPALM

## Network Delivery Lifecycle Overview

We're starting at the beginning (generally a good place to start)! Before we can automate and orchestrate our CI/CD Pipeline for our network we need to get our configurations into version control. In this section we will do just that, as well as take a quick preview at how we can programmatically push/merge/diff our configurations.

As you can see below, we will be primarily operating in the "write" and "test" phases of the Network Delivery Lifecycle.

## Network Delivery Lifecycle



## Managing Configurations in General

Before we begin to integrate "network-y" things into our Jenkins pipeline, we first need to get a handle on how we want to represent our network configurations as code. There are probably an unlimited number of ways to

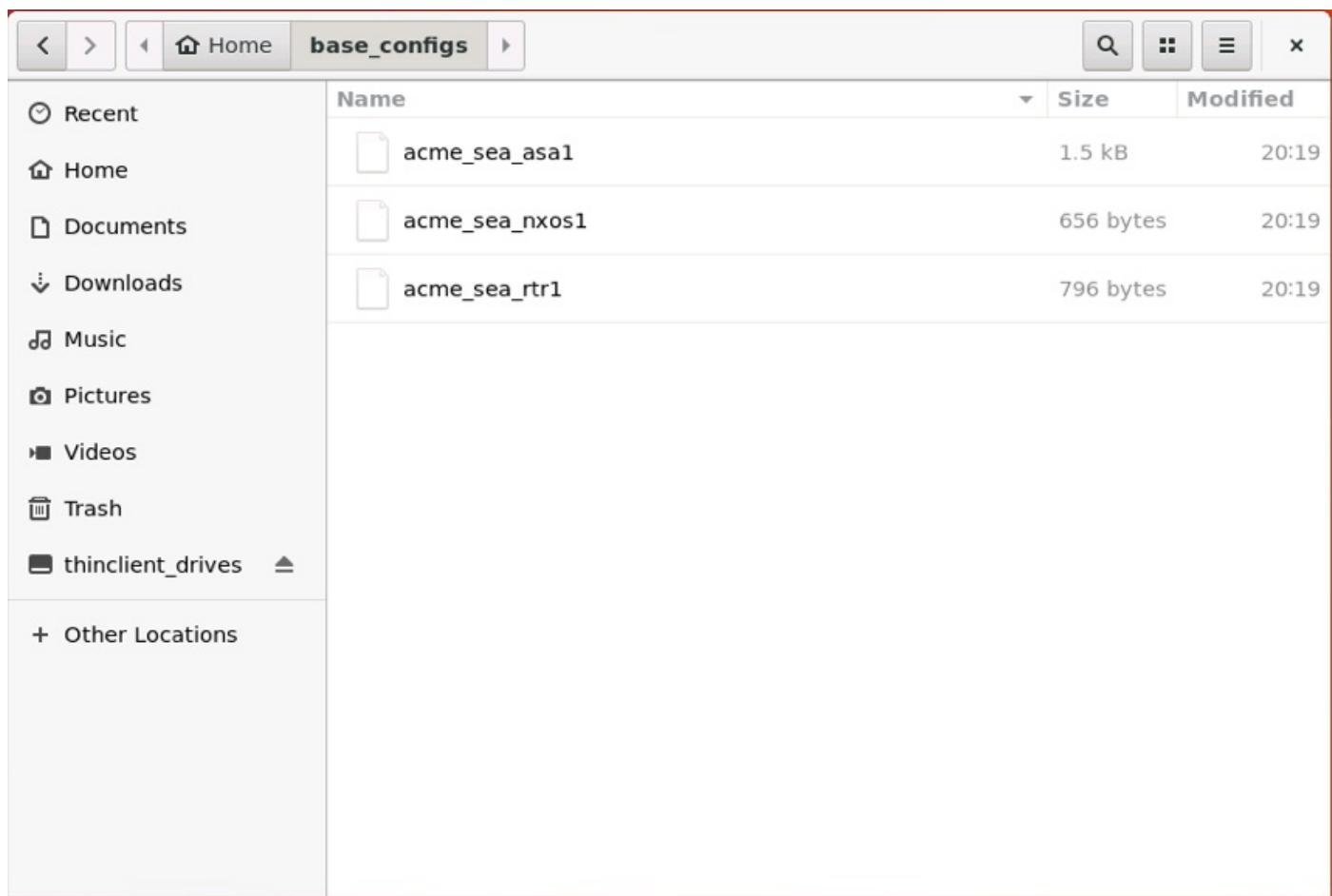
deal with this particular challenge, a few of the common options are outlined below:

- Flat configurations stored in a VCS (GitHub for example)
- Ansible Playbooks
- Ansible Playbooks stored in a VCS
- Proprietary NMS type Platforms such as Cisco Prime, Solarwinds Orion, etc.
- Custom built Python setups leveraging Netmiko, Napalm, Brigade, etc., etc.
- Custom built something else entirely!

For the purposes of this lab, and to keep things relatively simple, we will start with storing our configurations as flat configuration files in GitHub. This is a good first step on any "Infrastructure as Code" journey! By simply having configurations in a repository that is friendly to multiple contributors, tracks version history, and allows for open communication about changes you will already reap tons of benefits!

## Gitting the Configs in Git

The baseline configurations for your devices have been stored in your user folder in the "base\_configs" directory.

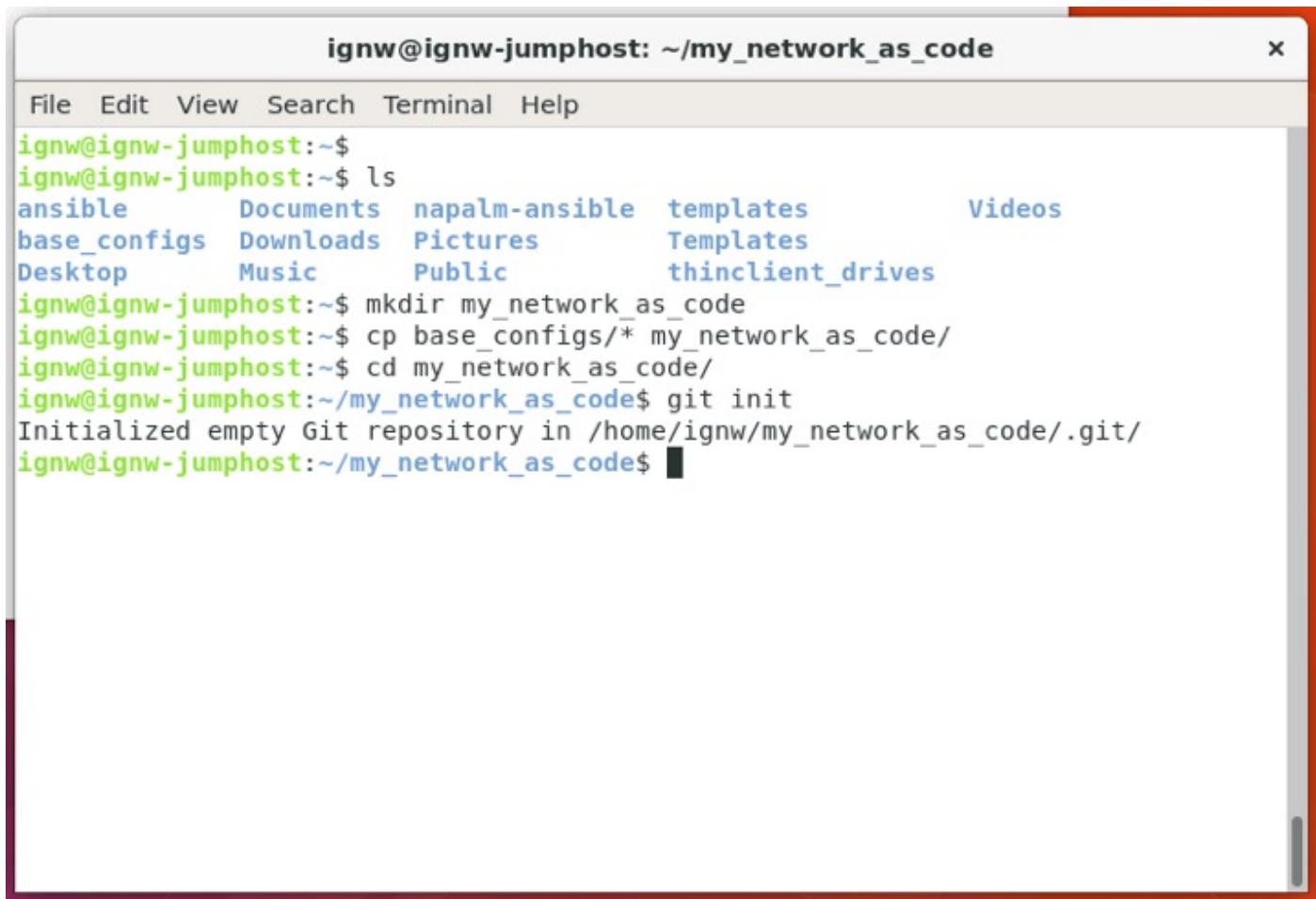


A screenshot of a file explorer window titled "base\_configs". The left sidebar shows standard folder icons for Recent, Home, Documents, Downloads, Music, Pictures, Videos, Trash, and thinclient\_drives. A plus sign indicates Other Locations. The main pane displays a list of three files: acme\_sea\_asa1, acme\_sea\_nxos1, and acme\_sea\_rtr1. The columns are labeled Name, Size, and Modified. The table has a header row with these labels and a footer row with sorting arrows.

Name	Size	Modified
acme_sea_asa1	1.5 kB	20:19
acme_sea_nxos1	656 bytes	20:19
acme_sea_rtr1	796 bytes	20:19

Copy this directory into a new directory called "my\_network\_as\_code". Move into that directory and git initialize it.

```
mkdir my_network_as_code  
cp base_configs/* my_network_as_code  
cd my_network_as_code  
git init
```



A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/my\_network\_as\_code". The window has a standard OS X-style interface with a menu bar at the top. The terminal session shows the following commands being run:

```
ignw@ignw-jumphost:~$  
ignw@ignw-jumphost:~$ ls  
ansible      Documents  napalm-ansible  templates      Videos  
base_configs  Downloads  Pictures       Templates  
Desktop       Music     Public        thinclient_drives  
ignw@ignw-jumphost:~$ mkdir my_network_as_code  
ignw@ignw-jumphost:~$ cp base_configs/* my_network_as_code/  
ignw@ignw-jumphost:~$ cd my_network_as_code/  
ignw@ignw-jumphost:~/my_network_as_code$ git init  
Initialized empty Git repository in /home/ignw/my_network_as_code/.git/  
ignw@ignw-jumphost:~/my_network_as_code$ █
```

Just as we've done before we now need to let Git know which files we would like to track. `git add` all of the configuration files in this directory.

```
git add acme-sea-rtr1  
git add acme-sea-asal  
git add acme-sea-nxos1
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ ls
ansible      Documents  my_first_pipeline  Templates
base_configs Downloads  Pictures           thinclient_drives
Desktop       Music     Public            Videos
ignw@ignw-jumphost:~$ cp -r base_configs/ my_network_as_code
ignw@ignw-jumphost:~$ cd my_network_as_code/
ignw@ignw-jumphost:~/my_network_as_code$ git init
Initialized empty Git repository in /home/ignw/my_network_as_code/.git/
ignw@ignw-jumphost:~/my_network_as_code$ git add acme_sea_rtr1
ignw@ignw-jumphost:~/my_network_as_code$ git add acme_sea_asa1
ignw@ignw-jumphost:~/my_network_as_code$ git add acme_sea_nxos1
ignw@ignw-jumphost:~/my_network_as_code$ 
```

With that out of the way, pop over to your Github page and create a new repository called "my\_network\_as\_code". Make sure to *not* check the "Initialize this repository with a README" box.

Create a New Repository - Mozilla Firefox

Create a New Repository x +

GitHub, Inc. (US) https://github.com/new

Search GitHub Pull requests Issues Apps Explore

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner carlniger / Repository name my\_network\_as\_code ✓

Great repository names are short and memorable. Need inspiration? How about [special-winner](#).

Description (optional)

Public Anyone can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with a README This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None Add a license: None

Create repository

© 2018 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About

Transferring data from assets-cdn.github.com...

Back on the Jumphost, make your initial commit to git with a helpful commit message.

```
git commit -m "initializing my_network_as_code"
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
Initialized empty Git repository in /home/ignw/my_network_as_code/.git/
ignw@ignw-jumphost:~/my_network_as_code$ git add acme_sea_rtr1
ignw@ignw-jumphost:~/my_network_as_code$ git add acme_sea_asal
ignw@ignw-jumphost:~/my_network_as_code$ git add acme_sea_nxos1
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m "initializing my_network_as_code"
[master (root-commit) de3a05c] initializing my_network_as_code
 3 files changed, 143 insertions(+)
 create mode 100644 acme_sea_asal
 create mode 100644 acme_sea_nxos1
 create mode 100644 acme_sea_rtr1
ignw@ignw-jumphost:~/my_network_as_code$ git remote add origin git@github.com:carlniger/my_network_as_code
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 1.70 KiB | 1.70 MiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To github.com:carlniger/my_network_as_code
 * [new branch] master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_network_as_code$
```

Next add the remote origin of your Github repository, and finally, push your configs to Github.

```
git remote add origin git@github.com:[your github username]/my_network_as_code
git push -u origin master
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
Initialized empty Git repository in /home/ignw/my_network_as_code/.git/
ignw@ignw-jumphost:~/my_network_as_code$ git add acme_sea_rtr1
ignw@ignw-jumphost:~/my_network_as_code$ git add acme_sea_asa1
ignw@ignw-jumphost:~/my_network_as_code$ git add acme_sea_nxos1
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m "initializing my_network_as_code"
[master (root-commit) de3a05c] initializing my_network_as_code
 3 files changed, 143 insertions(+)
 create mode 100644 acme_sea_asa1
 create mode 100644 acme_sea_nxos1
 create mode 100644 acme_sea_rtr1
ignw@ignw-jumphost:~/my_network_as_code$ git remote add origin git@github.com:carlniger/my_network_as_code
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 1.70 KiB | 1.70 MiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To github.com:carlniger/my_network_as_code
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
ianw@ianw-jumphost:~/my network as code$
```

## Now what?

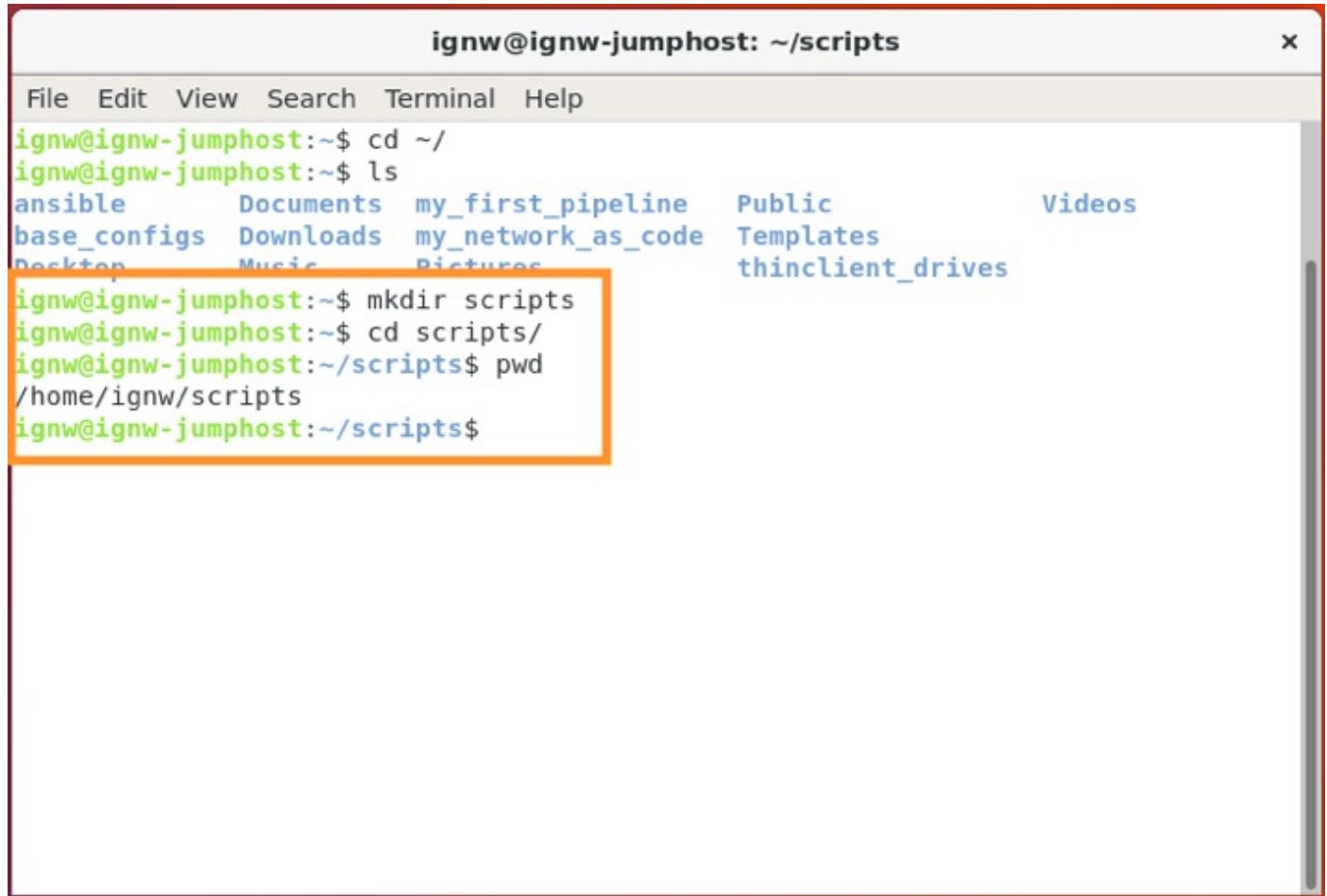
At this point, assuming the proper controls and more importantly culture/business practices, are in place, you have properly versionable configurations! By itself, this is nice, but many organizations have already had versionable configurations by storing configurations in something like Sharepoint(!?!?!), Dropbox, or some other versionable storage system. We need to take this one step further: deploying these configurations in an automated fashion.

As noted previously, there are a myriad of ways to go about this, but we'll start small and simple with the Python module NAPALM (Network Automation and Programmability Abstraction Layer with Multi-vendor support). NAPALM is an open-source library that abstracts (as the name implies) the configuration of multi-vendor platforms by providing a consistent interface in Python.

For our purposes we don't need to know too terribly much about the under-pinnings of NAPALM, but the piece that we care about for now is that NAPALM supports the ability to merge or replace supported device configurations. Let's begin by creating a simple script to output the changes that a merge event would have on our "acme\_sea\_rtr1" device.

Create a new directory in your home folder simply called "scripts", then move into that directory:

```
cd ~/  
mkdir scripts  
cd scripts
```



The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~/scripts". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal session shows the following commands and output:

```
ignw@ignw-jumphost:~$ cd ~/  
ignw@ignw-jumphost:~$ ls  
ansible      Documents   my_first_pipeline    Public          Videos  
base_configs  Downloads   my_network_as_code  Templates  
decktop       Music      Pictures           thinclient_drives  
ignw@ignw-jumphost:~$ mkdir scripts  
ignw@ignw-jumphost:~$ cd scripts/  
ignw@ignw-jumphost:~/scripts$ pwd  
/home/ignw/scripts  
ignw@ignw-jumphost:~/scripts$
```

The command "mkdir scripts" and its resulting directory structure are highlighted with an orange rectangle.

## Getting Started with NAPALM

Create a new file called "csr\_merge.py".

```
touch csr_merge.py
```

*Note:* Feel free to install an editor of your choice on the Ubuntu Jumphost, use Geddit, use vim, or edit the file on your local system. This lab guide will use vi for documentation purposes.

The very first thing we will need to do is import the NAPALM module -- this is because this is a "third-party" module (not included in the standard library). It is common convention -- and a good practice -- to perform all import tasks at the beginning of our script, so that's where we will put our import.

```
import napalm
```

Next, think about the data you would normally need in order to log into a device -- you'll of course need the device IP address or hostname if it is DNS resolvable, and you'll need to know the username and password too. Let's go ahead and capture these inputs and store them in variables so that we can use them throughout our script as needed. By storing these inputs as variables we can ensure that if a device input (user/pass/IP) changes we only have to modify it in a single location as opposed to every instance of the field in our script.

Longer term, we will move away from storing credentials in our Python files natively, but for the purposes of getting some forward momentum and testing how NAPALM behaves we can just stash these in our script for now.

```
username = 'ignw'  
password = 'ignw'  
device_ip = '10.0.0.5'
```

With the preamble bits out of the way we can go about setting up a NAPALM connection to our "dev" router. As is always a good practice when working with open source projects, consulting the documentation is a good place to start to figure out what we can do, and how we can do it. The NAPALM project has very good documentation, including a very simple example for how to get our first connection setup. You can find the

documentation [here](#).

Per the documentation our first step is to select which "napalm\_driver" we need to use. NAPALM as stated previously is an abstraction layer, but it of course does not provide an abstraction for all platforms, and it still does need to know what type of end device we are interacting with so it can behave accordingly. Lucky for us, "ios" (Cisco IOS) is one of the well supported platforms within NAPALM and the underlying Netmiko library.

Once we've selected a driver, we create a device object from the driver, passing in our previously created variables for the hostname, username, and password respectively.

```
napalm_driver = napalm.get_network_driver('ios')
device = napalm_driver(hostname=device_ip,
                       username=username,
                       password=password)
```



The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~/scripts". The window contains a Python script. The code imports the napalm module and defines variables for the username ('ignw'), password ('ignw'), and device IP ('10.0.0.5'). It then creates a device object using the napalm.get\_network\_driver('ios') method and passes the device IP, username, and password as arguments. The terminal window also shows a file icon and a scroll bar. At the bottom, it displays the message "'csr\_merge.py' 11 lines, 224 characters written".

```
File Edit View Search Terminal Help
import napalm

username = 'ignw'
password = 'ignw'
device_ip = '10.0.0.5'

napalm_driver = napalm.get_network_driver('ios')
device = napalm_driver(hostname=device_ip,
                       username=username,
                       password=password)

"csr_merge.py" 11 lines, 224 characters written
```

We are now ready to "open" (connect to) our device with the "open" method of our device object.

```
device.open()
```

Save your script and run it. Make sure you are using "python3", there is no Python 2.7 on this system

(NAPALM supports legacy Python, it is just not presently installed).

If you didn't get any errors, then your script is executing properly. If you saw any netmiko errors you've likely got an incorrect IP, username, and/or password.

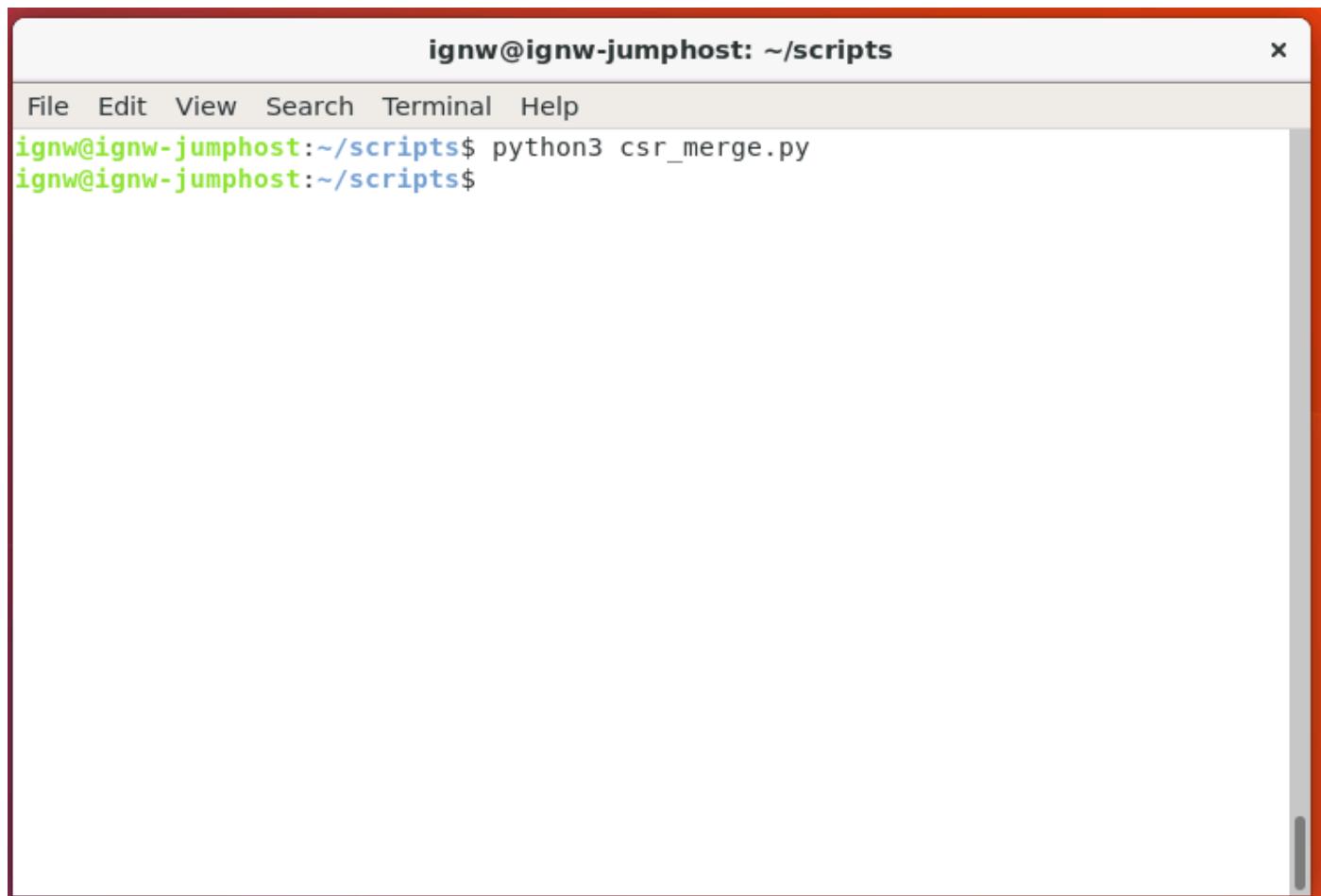
# Merge Candidate and Diffs

Now that we have established a connection to our router, we need to try to figure out how we can merge in our configurations from our git repository. We can see from the [documentation](#) that configuration replace and merge operations are fairly straight forward with NAPALM.

The first thing we need to do is to pass our device object a candidate configuration to load. We can do this with the "load\_merge\_candidate" method, passing in a path to our file. In our case we will use the relative path to keep things simple and (hopefully) a bit more portable.

```
device.load_merge_candidate(filename='..../my_network_as_code/acme-sea-rtr1')
```

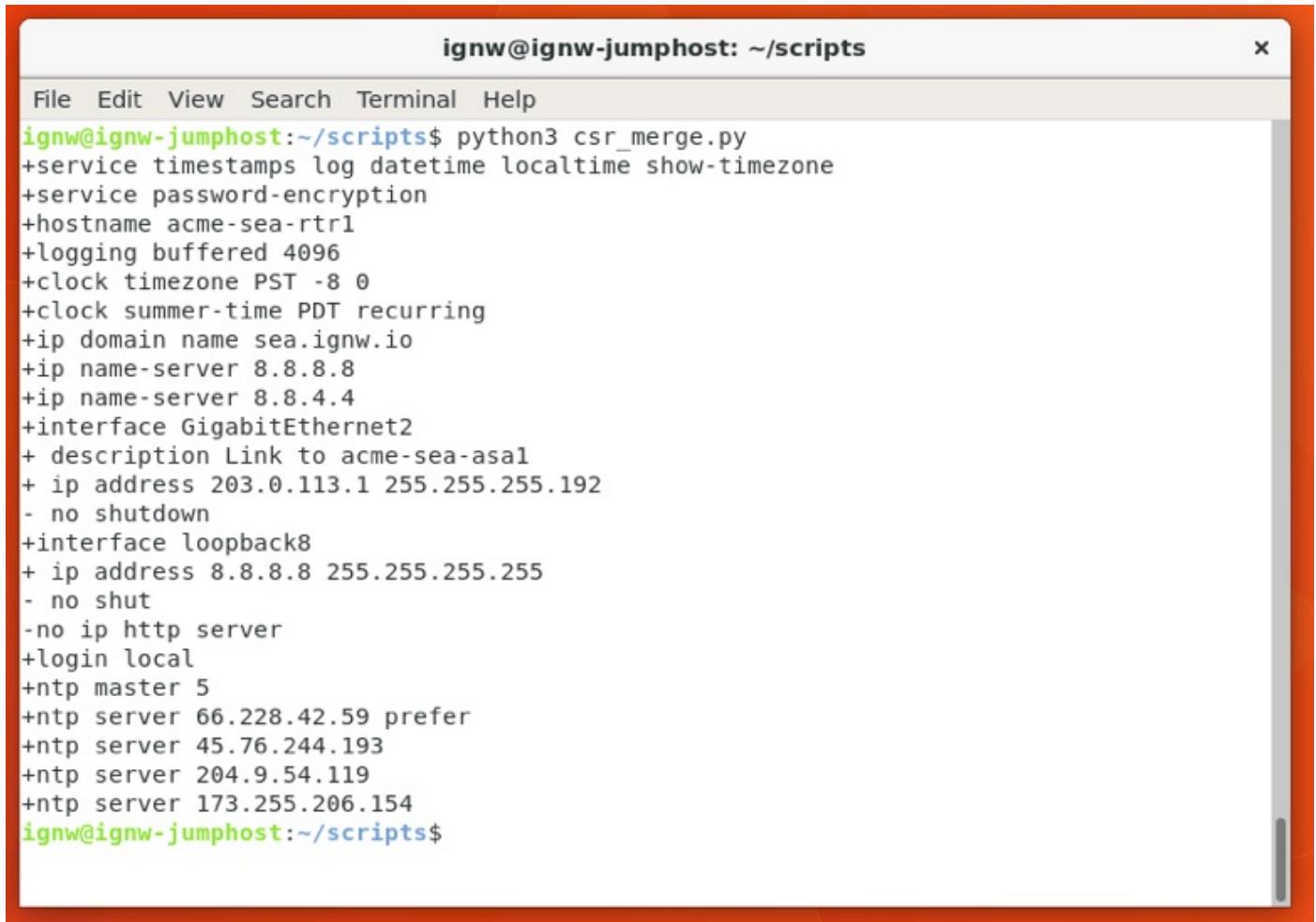
Run your script once more to make sure that the path is correct and NAPALM does not raise any exceptions.

A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/scripts". The window has a standard OS X-style title bar with a close button. Below the title bar is a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area of the terminal shows the command "python3 csr\_merge.py" being run at the prompt "ignw@ignw-jumphost:~/scripts\$". The terminal has a vertical scroll bar on the right side.

```
ignw@ignw-jumphost:~/scripts$ python3 csr_merge.py
```

NAPALM hasn't been very verbose thus far! Let's change that by printing out the delta between the current device configuration and the configuration in our git repository. We can do this with the "compare\_config()" method of our device object.

```
print(device.compare_config())
```

A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/scripts". The window has a menu bar with File, Edit, View, Search, Terminal, and Help. The terminal content shows the command "python3 csr\_merge.py" being run, followed by a large block of configuration differences. The configuration includes various service settings like timestamps, password encryption, and DNS servers, along with interface and loopback definitions.

```
ignw@ignw-jumphost:~/scripts$ python3 csr_merge.py
+service timestamps log datetime localtime show-timezone
+service password-encryption
+hostname acme-sea-rtr1
+logging buffered 4096
+clock timezone PST -8 0
+clock summer-time PDT recurring
+ip domain name sea.ignw.io
+ip name-server 8.8.8.8
+ip name-server 8.8.4.4
+interface GigabitEthernet2
+ description Link to acme-sea-asal
+ ip address 203.0.113.1 255.255.255.192
- no shutdown
+interface loopback8
+ ip address 8.8.8.8 255.255.255.255
- no shut
-no ip http server
+login local
+ntp master 5
+ntp server 66.228.42.59 prefer
+ntp server 45.76.244.193
+ntp server 204.9.54.119
+ntp server 173.255.206.154
ignw@ignw-jumphost:~/scripts$
```

Finally, some output! Does anything about this output strike you as a bit odd? Most (but not all) of this configuration is already on the router. Why would the configuration diff output anything?

## Templates, Diffs, and Idempotency

As you may have guessed by looking at the output some of these are default configurations that do not show up in the running configuration; "logging buffered 4096" for example is the default setting. The DNS server configurations in the running configuration are stored on a single line for both servers. As "service password-encryption" is configured on the router the password is encrypted, and thus does not match the password (in clear text) in our git repository.

This is one of the first challenges in the "Infrastructure as Code" journey: idempotency. An "official" definition may or may not help to clear up what exactly this is: "denoting an element of a set that is unchanged in value when multiplied or otherwise operated on by itself." That's quite a definition! A perhaps simpler, and certainly more applicable definition could be: "for an operation (or service call) to be idempotent, clients can make that same call repeatedly while producing the same result" -- if you've got a few minutes take a look at this video to learn a bit more about [idempotence](#).

While technically our *configurations* are idempotent (for things like the DNS servers) -- resulting in the same configuration end state, our script does not necessarily behave, strictly speaking, in an idempotent fashion. This is because our script sees that a change needs to occur. This is no fault of NAPALM, but actually a combination of issues with how we've structured the flat base configurations file, and how network devices (CLIs) act. We'll continue to run into this challenge as we continue to move forward, but we'll do our best to

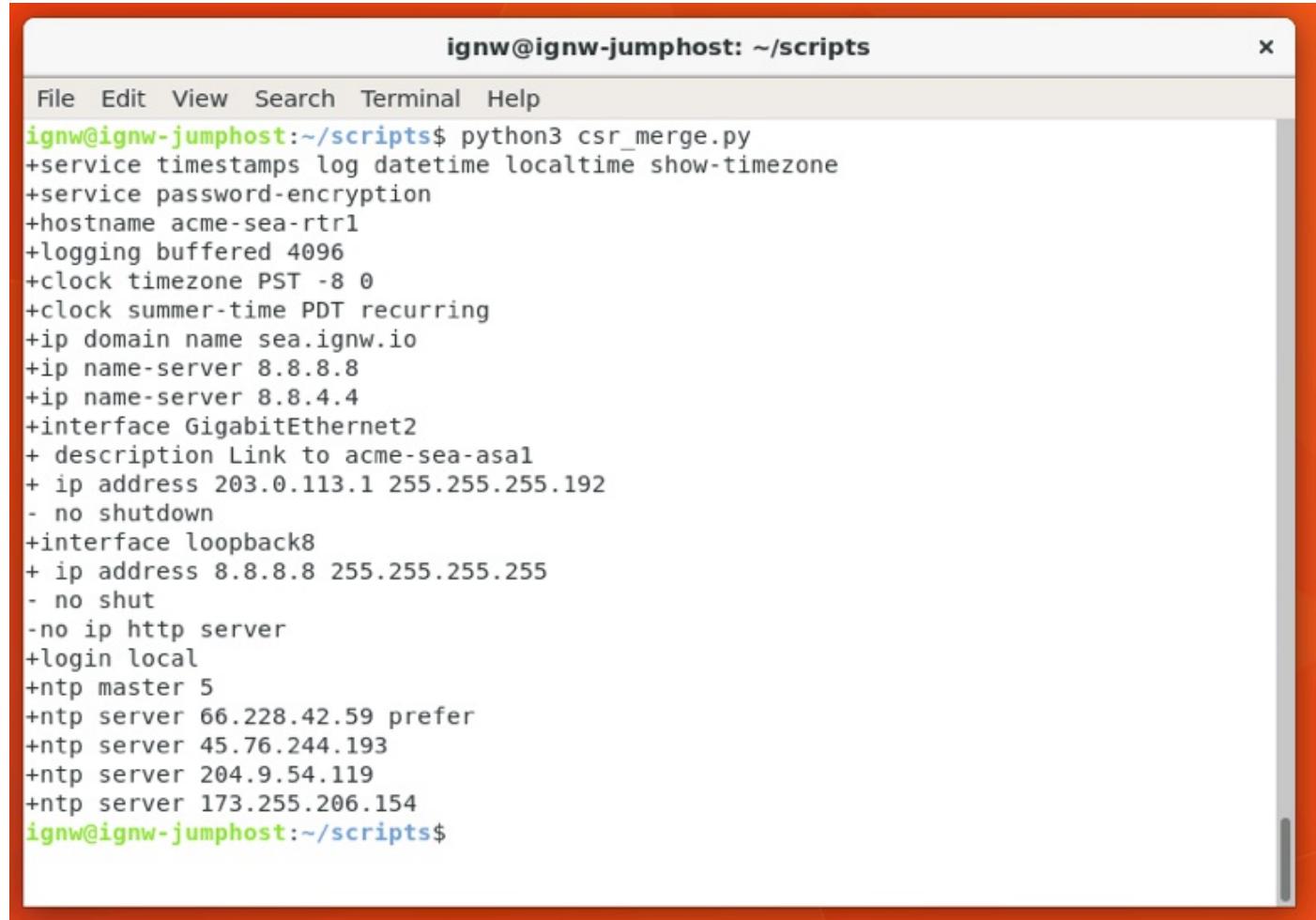
manage it!

One last thing before we move on -- it is a good idea to explicitly close the connection our script created. In a lab, it is probably no big deal to leave connections hanging open, but tying up VTY lines in production is likely a bit bigger of a deal!

Just like the other things we've done, our device object has a method expressly for closing the connection -- it is even aptly named "close".

```
device.close()
```

Run your script one last time to make sure things are working as expected.



The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~/scripts". The window contains a command-line interface with the following text:

```
ignw@ignw-jumphost:~/scripts$ python3 csr_merge.py
+service timestamps log datetime localtime show-timezone
+service password-encryption
+hostname acme-sea-rtr1
+logging buffered 4096
+clock timezone PST -8 0
+clock summer-time PDT recurring
+ip domain name sea.ignw.io
+ip name-server 8.8.8.8
+ip name-server 8.8.4.4
+interface GigabitEthernet2
+ description Link to acme-sea-asal
+ ip address 203.0.113.1 255.255.255.192
- no shutdown
+interface loopback8
+ ip address 8.8.8.8 255.255.255.255
- no shut
-no ip http server
+login local
+ntp master 5
+ntp server 66.228.42.59 prefer
+ntp server 45.76.244.193
+ntp server 204.9.54.119
+ntp server 173.255.206.154
ignw@ignw-jumphost:~/scripts$
```

As you can see there was no change in the behavior of the script (at least visually), but it is a good idea to try to build good habits!

In this lab we've made the first step toward managing our network Infrastructure as Code, but we still have a long way to go...

# Templatizing Configurations with Ansible

## Overview and Objectives

In this lab you will templatize your configurations. The goal here is to create templates that are flexible enough, yet as simple as possible. Once that is done configurations can be managed entirely by variable files -- thus abstracting away syntax differences between platforms or vendors, hopefully eliminating human error as much as possible.

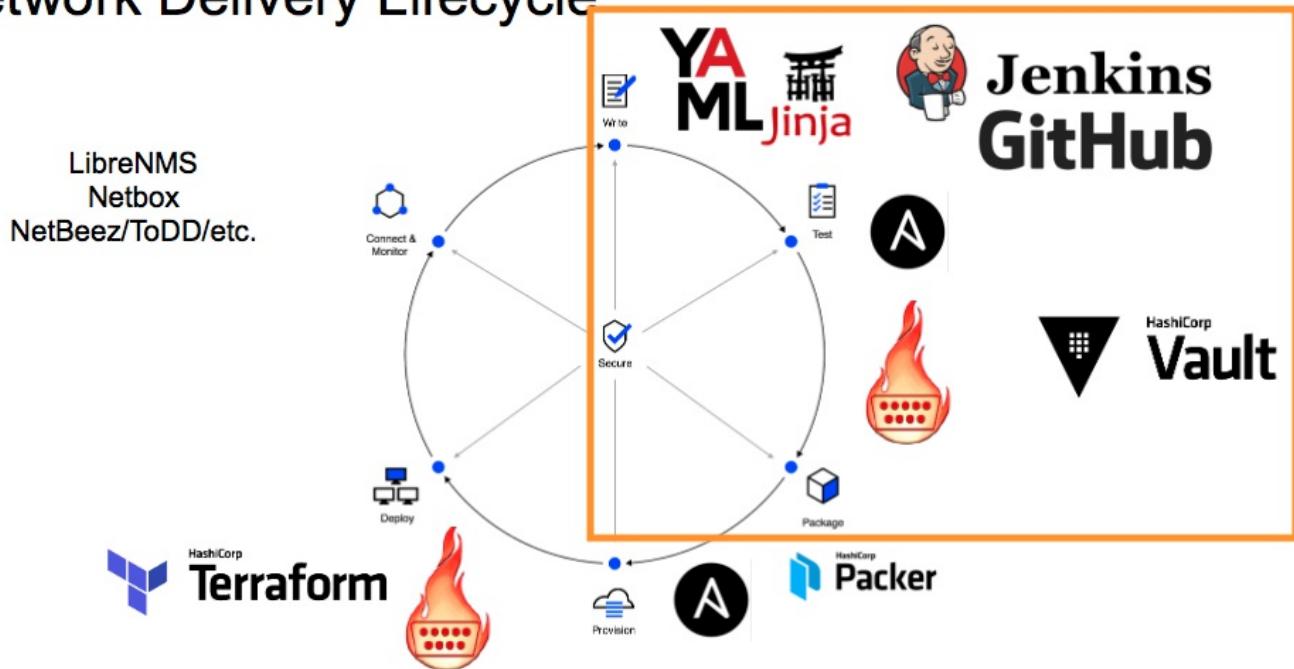
Objectives:

- Walk through step-by-step templatizing of a Cisco IOS-XE configuration file.
- Copy and review ASA and NXOS templatized configuration files
- Write an Ansible Playbook to "render" configuration files from templates
- Push all of your work to GitHub

## Network Delivery Lifecycle Overview

As you can see below, we are still operating primarily in the "write" phase, however we're going a bit further than that by actually rendering our configurations as well taking us a bit into the "packaging" phase. There won't always be a perfect mapping between the HashiCorp Application Lifecycle map and our Network Delivery Lifecycle map, and this is one of those times where the lines are tad bit blurry!

## Network Delivery Lifecycle



## Why and how to Template Configurations

Having our configurations in Git, as discussed, is obviously a good start... but we still need to manage configurations manually, sure we have pull requests and opportunities for peer-review built in (yay!), but humans are still have plenty of opportunities to make typos!

In most organizations it is very unlikely to be able to *fully* eliminate human error (at least for now!), but we certainly should make a concerted standardize and automate where we can.

One way to attempt to eliminate the need for humans typing is to templatize configurations. Network professionals have of course been doing this for just about as long as routers and switches have been around (to varying degrees of success!), so this concept should be nothing new. Lucky for us there are tons and tons of tools available to help templatize configurations, as well as add logic into our templates so that they can be more flexible and more powerful.

In this lab section we'll use Ansible and the template module to templatize configurations. Ansible is ultimately utilizing the very popular Python Jinja2 templating language under the covers. Depending on your environment you may elect to utilize pure Python and Jinja2 without the additional layer of Ansible, however Ansible is an established tool that provides a well thought out inventory and variable structure which we can put to work right away.

Before we jump into templates and variables it is a good idea to think about how to structure your inventory. An inventory (and variable) system that works well in your environment is perhaps the most crucial component to a successful infrastructure as code deployment. In this case, in a small lab, we won't have to contend with a huge amount of devices, groups, vendors, or any of the other challenges that you would likely encounter in a production environment. We'll keep the inventory and variables very simple and in line with a "normal" small(ish) scale Ansible deployment.

## Getting Started

To start, move into the "my\_network\_as\_code" directory. Create a new folder called "legacy\_configs", and move all of the base configurations there.

```
cd ~/my_network_as_code
mkdir legacy_configs
mv acme* legacy_configs
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ cd ~/my_network_as_code/
ignw@ignw-jumphost:~/my_network_as_code$ mkdir legacy_configs
ignw@ignw-jumphost:~/my_network_as_code$ mv acme* legacy_configs/
ignw@ignw-jumphost:~/my_network_as_code$ ls
legacy_configs
ignw@ignw-jumphost:~/my_network_as_code$
```

We'll need an inventory file as well. Ansible allows for INI-like or YAML inventory files. We'll use a simple INI-like file as our inventory. Create an inventory file, simply called "inventory" and open the file for editing.

```
touch inventory
vi inventory
```

We'll need to consider how we want to structure our groups in our inventory file. We'll try to keep it simple, and create a group for "sea" (our site of Seattle), and a group for each of the device types that we have in our environment: cisco-ios-xe-routers, cisco-asa, and cisco-nxos.

In an INI like inventory file, square brackets represent groups, create the four groups outlined above.

The screenshot shows a terminal window with a red border. The title bar reads "ignw@ignw-jumphost: ~/my\_network\_as\_code". The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". Below the menu is a list of inventory groups:

```
[sea]
[cisco-ios-xe-routers]
[cisco-asa]
[cisco-nxos]
```

The "[cisco-nxos]" group is currently selected, indicated by a black selection bar. The terminal window has a vertical scroll bar on the right side.

Great, so we have "groups" now what do we put in them!? Our devices of course! Add the three devices all to the Seattle (sea) group, and to the corresponding group for the platform type.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
[sea]
acme-sea-rtr1
acme-sea-asal
acme-sea-nxos1

[cisco-ios-xe-routers]
acme-sea-rtr1

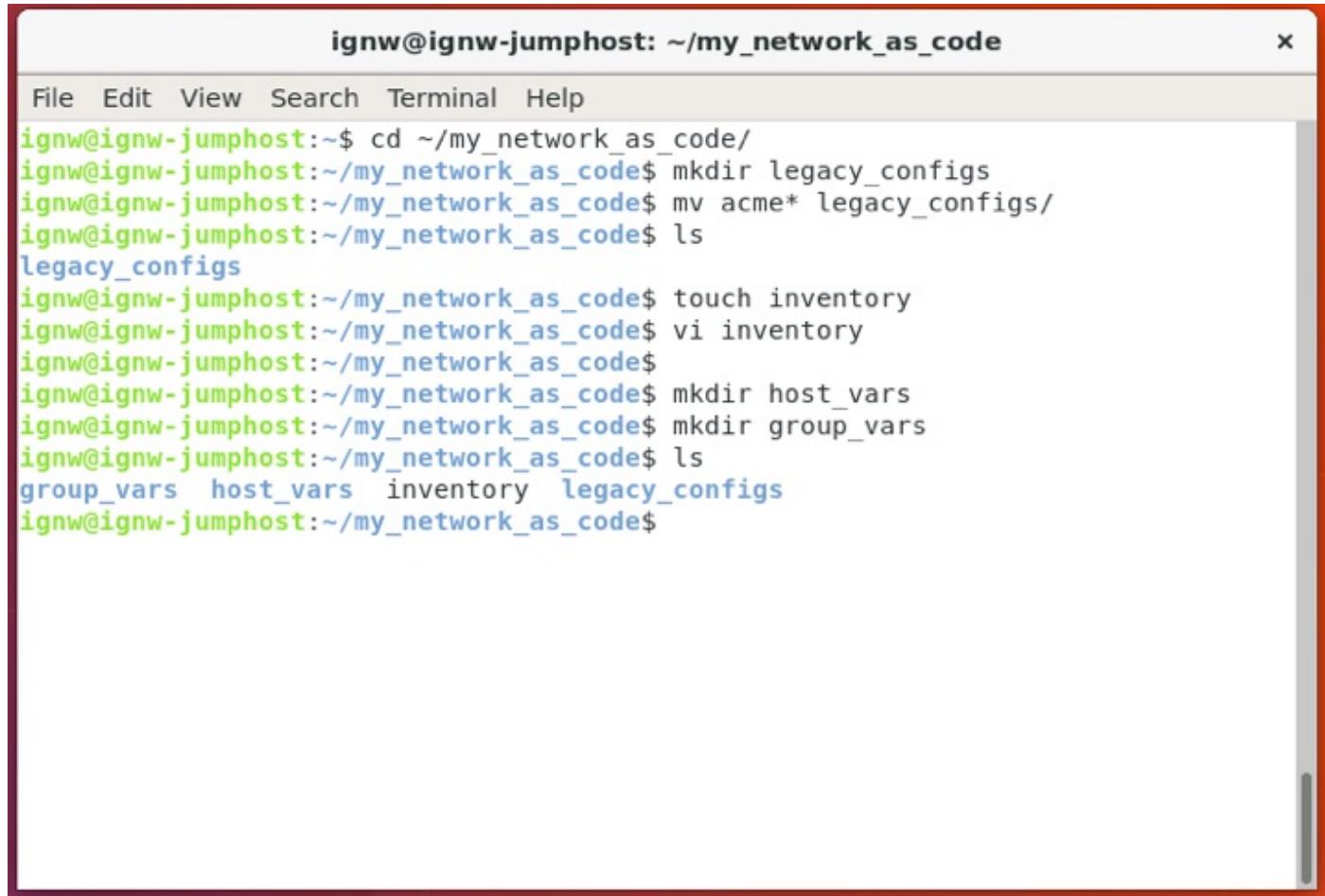
[cisco-asa]
acme-sea-asal

[cisco-nxos]
acme-sea-nxos1

"inventory" 13 lines, 143 characters written
```

Next, we can create our host and group variable folders.

```
mkdir host_vars  
mkdir group_vars
```

A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/my\_network\_as\_code". The window has a standard OS X-style title bar with icons for close, minimize, and zoom. The main area shows a command-line session:

```
ignw@ignw-jumphost:~$ cd ~/my_network_as_code/
ignw@ignw-jumphost:~/my_network_as_code$ mkdir legacy_configs
ignw@ignw-jumphost:~/my_network_as_code$ mv acme* legacy_configs/
ignw@ignw-jumphost:~/my_network_as_code$ ls
legacy_configs
ignw@ignw-jumphost:~/my_network_as_code$ touch inventory
ignw@ignw-jumphost:~/my_network_as_code$ vi inventory
ignw@ignw-jumphost:~/my_network_as_code$ 
ignw@ignw-jumphost:~/my_network_as_code$ mkdir host_vars
ignw@ignw-jumphost:~/my_network_as_code$ mkdir group_vars
ignw@ignw-jumphost:~/my_network_as_code$ ls
group_vars  host_vars  inventory  legacy_configs
ignw@ignw-jumphost:~/my_network_as_code$
```

For now we can leave these folders empty, but we will be making use of them very soon!

Since we know we want to have templates, we will also need somewhere to stash them, let's create a folder simply called "templates".

```
mkdir templates
```

## First Template

Like we did in the last lab, let's start working with the CSR router. Create a file in the templates directory called "cisco\_ios\_xe\_router.j2" -- the ".j2" extension is of course for the Jinja2 templating language.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ls
group_vars host_vars inventory legacy_configs
ignw@ignw-jumphost:~/my_network_as_code$ mkdir templates
ignw@ignw-jumphost:~/my_network_as_code$ touch templates/cisco_ios_xe_router.j2
ignw@ignw-jumphost:~/my_network_as_code$ tree
.
├── group_vars
├── host_vars
└── inventory
    └── legacy_configs
        ├── acme_sea_asal
        ├── acme_sea_nxos1
        └── acme_sea_rtr1
└── templates
    └── cisco_ios_xe_router.j2

4 directories, 5 files
ignw@ignw-jumphost:~/my_network_as_code$
```

Copy the contents of the "acme\_sea\_rtr1" base configuration into our new .j2 file; we'll use this to start crafting our template.

```
cat legacy_configs/acme-sea-rtr1 >> templates/cisco_ios_xe_router.j2
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ tree
.
├── group_vars
├── host_vars
└── inventory
    └── legacy_configs
        ├── acme_sea_asal
        ├── acme_sea_nxos1
        └── acme_sea_rtr1
    └── templates
        └── cisco_ios_xe_router.j2

4 directories, 5 files
ignw@ignw-jumphost:~/my_network_as_code$ cat legacy_configs/acme_sea_rtr1 >> templates/cisco_ios_xe_router.j2
ignw@ignw-jumphost:~/my_network_as_code$ cat templates/cisco_ios_xe_router.j2
service timestamps debug datetime msec
service timestamps log datetime localtime show-timezone
service password-encryption
!
hostname acme-sea-rtr1
!
logging buffered 4096
!
```

Consider the base configuration of the "acme-sea-rtr1" device. What in this configuration could be templated? What configurations will be common across multiple devices (even devices of different operating systems)?

```

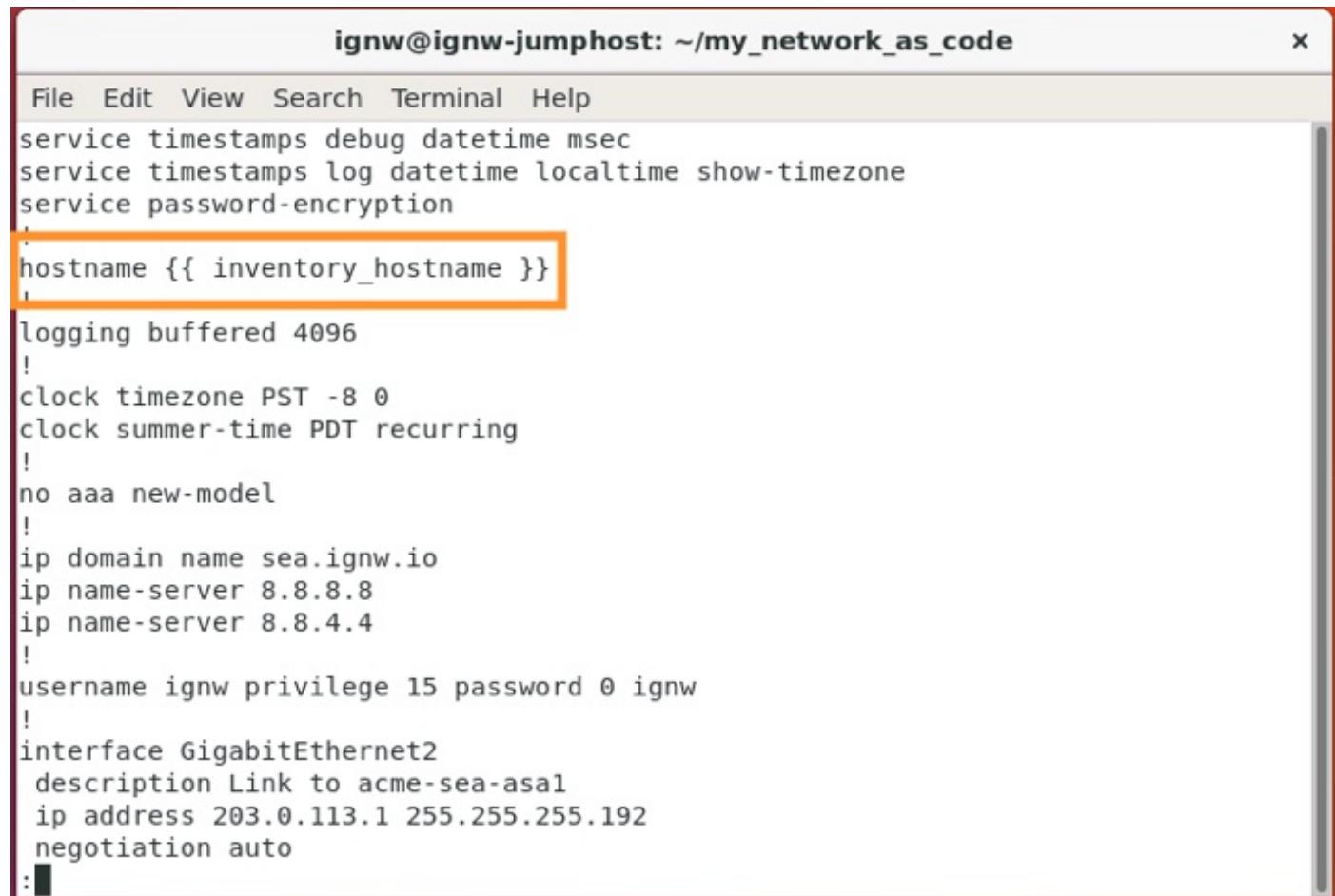
service timestamps debug datetime msec
service timestamps log datetime localtime show-timezone
service password-encryption
!
hostname acme-sea-rtr1
!
logging buffered 4096
!
clock timezone PST -8 0
clock summer-time PDT recurring
!
no aaa new-model
!
ip domain name sea.ignw.io
ip name-server 8.8.8.8
ip name-server 8.8.4.4
!
username ignw privilege 15 password 0 ignw
!
interface GigabitEthernet2
description Link to acme-sea-asal
ip address 203.0.113.1 255.255.255.192
negotiation auto
no shutdown
!
interface loopback8
ip address 8.8.8.8 255.255.255.255
no shut
!
no ip http server
ip scp server enable
!
ip ssh version 2
!
line vty 0 4
login local
!
ntp master 5
ntp server 66.228.42.59 prefer
ntp server 45.76.244.193
ntp server 204.9.54.119
ntp server 173.255.206.154

```

Certainly we know that every device will need a hostname, so maybe that is a good place to start! In the Jinja2

templating language, variables are represented by the double squiggly braces: "{{ my\_variable\_here }}". We know we want to "variablize" the hostname, but what variable do we use to represent it? If we were leveraging Jinja2 without Ansible we would need to determine the best way to handle input of variables, and then we would properly pass those to the templating engine via a Python script.

In this case since we have elected to utilize Ansible we can simply rely on traditional Ansible variables/variable naming. Ansible already has a built in variable for exactly this: "inventory\_hostname". Update the "cisco\_ios\_xe\_router.j2" file replacing the hostname with our j2 variable.



```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
service timestamps debug datetime msec
service timestamps log datetime localtime show-timezone
service password-encryption
!
hostname {{ inventory_hostname }}
!
logging buffered 4096
!
clock timezone PST -8 0
clock summer-time PDT recurring
!
no aaa new-model
!
ip domain name sea.ignw.io
ip name-server 8.8.8.8
ip name-server 8.8.4.4
!
username ignw privilege 15 password 0 ignw
!
interface GigabitEthernet2
description Link to acme-sea-asal
ip address 203.0.113.1 255.255.255.192
negotiation auto
:
```

Before we continue trying to templatize all the things(!!), we should probably validate that our templating is working as we expect/intend it to. To do that we need to get some basic Ansible configurations taken care of.

Create an "ansible.cfg" file in your "my\_network\_as\_code" directory with the following settings:

```
[defaults]
ansible_python_interpreter=/usr/bin/python3
host_key_checking=False
host_key_auto_add=True
retry_files_enabled=False
inventory=inventory
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ cat templates/cisco_ios_xe_router.j2 | less
ignw@ignw-jumphost:~/my_network_as_code$ vi ansible.cfg
ignw@ignw-jumphost:~/my_network_as_code$ cat ansible.cfg
[defaults]
ansible_python_interpreter=/usr/bin/python3
host_key_checking=False
host_key_auto_add=True
retry_files_enabled=False
inventory=inventory
ignw@ignw-jumphost:~/my_network_as_code$ █
```

## Generating Templates

Next we need to create a Playbook that we can run to generate our templates. Create a new file called "generate\_configurations.yaml", and open it for editing.

Enter the following into your Playbook file:

```
---
- name: Generate IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  tasks:
    - template:
        src=templates/cisco_ios_xe_router.j2
        dest=
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
---
- name: Generate IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  tasks:
    - template:
        src=templates/cisco_ios_xe_router.j2
        dest=
```

The above configurations creates a Play named "Generate IOS-XE Router Configurations", using the connection of "local". This basically means Ansible will "connect" to the host it is being ran on to execute this playbook (as opposed to connecting to a remote host to run the Play(s)). We don't need to gather any facts since we are just generating text templates, and we would like to execute this Play against the "cisco-ios-xe-routers" hosts group which of course contains our "acme-sea-rtr1" device.

Our only task will be to use the "template" module to generate our template. This module obviously needs to know which source template we would like to use, and where to output the resulting file. We should probably create a new directory where we can store generated configurations. Create a directory called "configs".

```
mkdir configs
```

Reopen the Playbook for editing. We will want to export the configurations to the configs folder, but we will need some way to keep track of which configuration is which (rather than just having "config", "config1" or something), so we can use the "inventory\_hostname" variable once more to ensure that our output configurations match the hostname of the device they correspond to.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
---
- name: Generate IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  tasks:
    - template:
        src=templates/cisco_ios_xe_router.j2
        dest=configs/{{ inventory_hostname }}]
```

Let's try to execute our Playbook to see what happens!

```
ansible-playbook generate_configurations.yaml
```

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+interface loopback8"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ ip address 8.8.8.8 255.255.255.255"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "- no shutdown"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "-no ip http server"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+login local"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp master 5"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp server 66.228.42.59 prefer"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp server 45.76.244.193"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp server 204.9.54.119"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp server 173.255.206.154"
}

PLAY RECAP ****
acme-sea-rtr1 : ok=2    changed=1    unreachable=0    failed=0
ignw@ignw-jumphost:~/my_network_as_code$ 

```

Uhoh! "/usr/bin/python" not found?! Well that does make some sense as legacy Python (Python 2) is not installed on this system. BUT... didn't we set the Python interpreter in the Ansible configuration? We certainly did, but there is a bit of a catch. Ansible has historically supported Python 2.7 (due to its immense install base), when we set the interpreter in the ansible.cfg file we set it for the version of Python that would be executed on remote hosts, as we are executing all of the Playbook tasks locally this wouldn't really apply. There are two (admittedly not ideal) ways to quickly address this, we can pass the interpreter we would like to use in as a variable at execution using the "-e" ("EXTRA\_VARS") flag, or we can add an "[all:vars]" entry to our inventory file that specifies the appropriate Python interpreter. Let's do the latter:

```

[all:vars]
ansible_python_interpreter=/usr/bin/python3

```

Execute your Playbook once again to see if that did the trick!

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook generate_configurations.yaml

PLAY [Generate IOS-XE Router Configurations] ****
TASK [template] ****
changed: [acme-sea-rtr1]

PLAY RECAP ****
acme-sea-rtr1 : ok=1    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$
```

What does the output file look like? Did the hostname get rendered properly?

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
PLAY RECAP ****
acme-sea-rtr1 : ok=1    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$ cat configs/acme-sea-rtr1
service timestamps debug datetime msec
service timestamps log datetime localtime show-timezone
service password-encryption
!
hostname acme-sea-rtr1
!
logging buffered 4096
!
clock timezone PST -8 0
clock summer-time PDT recurring
!
no aaa new-model
!
ip domain name sea.ignw.io
ip name-server 8.8.8.8
ip name-server 8.8.4.4
!
username ignw privilege 15 password 0 ignw
!
interface GigabitEthernet2
```

Now that we have a basic templating engine up and running we need to start putting it to work!

## Host and Group Variables

The next step is to pull variables in from our inventory/variable structure and properly render them in our configurations. Let's start with another low hanging fruit item -- the logging buffer size. This *could* (your mileage may vary) be a standard configuration item across your whole environment. Let's set our logging buffer size to 8192 for ALL of our devices.

Ansible will always create an "all" group for us, that's how we were able to set the "all:vars" before without even having to create an "all" group. We can take advantage of this and simply create an "all.yaml" file in the "group\_vars" directory where we store any variables we want to apply universally. We can even move the Python interpreter variable there to make the inventory file a bit tidier.

```
touch group_vars/all.yaml
```

Edit the inventory file to remove the "all:vars" section, copy and paste the "ansible\_python\_interpreter" variable into the newly created "all.yaml" file. Also add our new logging variable:

```
ansible_python_interpreter: /usr/bin/python3  
logging_buff_size: 8192
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ansible_python_interpreter: /usr/bin/python3
logging_buff_size: 8192
"group_vars/all.yaml" 2 lines, 69 characters written
```

With that out of the way, let's variablize our logging buffer string our configuration template:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
service timestamps debug datetime msec
service timestamps log datetime localtime show-timezone
service password-encryption
!
hostname {{ inventory_hostname }}
!
logging buffered {{ logging_buff_size }}
!
clock timezone PST -8 0
clock summer-time PDT recurring
!
no aaa new-model
!
ip domain name sea.ignw.io
ip name-server 8.8.8.8
ip name-server 8.8.4.4
!
username ignw privilege 15 password 0 ignw
!
interface GigabitEthernet2
  description Link to acme-sea-asal
  ip address 203.0.113.1 255.255.255.192
  negotiation auto
"templates/cisco_ios_xe_router.j2" 43 lines, 847 characters
```

Run the Playbook again to see if Ansible notices a change (it should since we modified the buffer size from 4096 -> 8192).

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
x

ignw@ignw-jumphost:~/my_network_as_code$ 
ignw@ignw-jumphost:~/my_network_as_code$ 
ignw@ignw-jumphost:~/my_network_as_code$ 
ignw@ignw-jumphost:~/my_network_as_code$ 
ignw@ignw-jumphost:~/my_network_as_code$ vi templates/cisco_ios_xe_router.j2
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook generate_configurations.yaml

PLAY [Generate IOS-XE Router Configurations] ****
TASK [template] ****
changed: [acme-sea-rtr1]

PLAY RECAP ****
acme-sea-rtr1 : ok=1    changed=1    unreachable=0    failed=0
ignw@ignw-jumphost:~/my_network_as_code$
```

Awesome, we're on the right track!

Create a "sea.yaml" for variables that will be specific to the Seattle group. Include the following (you can steal the values from the base config):

- timezone
- timezone\_offset
- timezone\_summer
- domain\_name
- name\_server\_1
- name\_server\_2

With the variables file prepared, modify the .j2 template file to accept the new variables.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
service timestamps debug datetime msec
service timestamps log datetime localtime show-timezone
service password-encryption
!
hostname {{ inventory_hostname }}
!
logging buffered {{ logging_buff_size }}
!
clock timezone {{ timezone }} {{ timezone_offset }} 0
clock summer-time {{ timezone_summer }} recurring
!
no aaa new-model
!
ip domain name {{ domain_name }}
ip name-server {{ name_server_1 }}
ip name-server {{ name_server_2 }}
!
username ignw privilege 15 password 0 ignw
!
interface GigabitEthernet2
description Link to acme-sea-asal
ip address 203.0.113.1 255.255.255.192
negotiation auto
no shutdown
"templates/cisco_ios_xe_router.j2" 44 lines, 926 characters
```

Run your Playbook once again to make sure things are still working correctly:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
TASK [template] *****
changed: [acme-sea-rtr1]

PLAY RECAP *****
acme-sea-rtr1 : ok=1    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$ cat configs/acme-sea-rtr1
service timestamps debug datetime msec
service timestamps log datetime localtime show-timezone
service password-encryption
!
hostname acme-sea-rtr1
!
logging buffered 8192
!
clock timezone PST -8 0
clock summer-time PDT recurring
!
no aaa new-model
!
ip domain name sea.acme.io
ip name-server 8.8.8.8
ip name-server 8.8.4.4
!
```

You could very easily spend some more time "templatizing" away even more configurations from even the short router base configuration provided, however it is important to remember that there is a point of diminishing returns!

For example, it would be very easy to add a variable to control the "http server" functionality of the router. Doing so would be a simple variable and would be one more "widget" that could be controlled if so desired. That said, it is very uncommon to see http server enabled in production environments (obviously you would know if this applies to your environment or not!), as such it is likely far simpler to just leave this in a "flat" text file section of the configuration (meaning do not templatize it).

With the low hanging fruit of the simple to templatize/variabilize configurations out of the way, it is time to turn to some that are a bit more complex. Let's start by tackling users.

As the configuration sits currently, there is only the single "ignw" user. This is well and good, but it is pretty common to have more than a single user, so we should ensure that our templating can support that.

## Logic in Jinja2

Lucky for us Jinja2 supports basic logic that will allow us to work with lists and dictionaries in our template. We also have a flexible data structure in our YAML variable files.

Open up the "sea.yaml" file and add the following:

```
users: {
    'ignw': {
        'privilege': 15,
        'password': 'ignw',
    }
}
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
timezone: PST
timezone_offset: -8
timezone_summer: PDT

domain_name: sea.acme.io
name_server_1: 8.8.8.8
name_server_2: 8.8.4.4

users: {
    'ignw': {
        'privilege': 15,
        'password': 'ignw',
    }
}

"group_vars/sea.yaml" 14 lines, 212 characters written
```

We've created a "users" dictionary object. Within our dictionary, we've created an "ignw" dictionary object that contains the attributes that we want to assign to our user.

Next we need to figure out how we can iterate over this data structure in Jinja2 to appropriately render our configuration file, thankfully Jinja2 has some great [documentation](#) that can help us along the way.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>My Webpage</title>
</head>
<body>
    <ul id="navigation">
        {% for item in navigation %}
            <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
        {% endfor %}
    </ul>

    <h1>My Webpage</h1>
    {{ a_variable }}

    {# a comment #}
</body>
</html>

```

Take a look at the "for" loop in the above screen shot. Of course we aren't dealing with HTML, but that's OK, the general syntax is fairly clear:

```

{% for item in list %}
    {{ item }}
{% endfor %}

```

We happen to have a dictionary object that we need to iterate over, however the logic is fundamentally the same. Replace the "username ignw ...." section of the template file with the following:

```

{% for user, data in users.items() %}
username {{ user }} privilege {{ data.privilege }} password 0 {{ data.password }}
}
{% endfor %}

```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
service timestamps log datetime localtime show-timezone
service password-encryption
!
hostname {{ inventory_hostname }}
!
logging buffered {{ logging_buff_size }}
!
clock timezone {{ timezone }} {{ timezone_offset }} 0
clock summer-time {{ timezone_summer }} recurring
!
no aaa new-model
!
ip domain name {{ domain_name }}
ip name-server {{ name_server_1 }}
ip name-server {{ name_server_2 }}
!
{% for user, data in users.items() %}
username {{ user }} privilege {{ data.privilege }} password 0 {{ data.password }}
{% endfor %}
!
interface GigabitEthernet2
description Link to acme-sea-asal
ip address 203.0.113.1 255.255.255.192
negotiation auto
```

Notice that we can use the ".items()" just like in "regular" Python when accessing a dictionary. In the above snippet we are iterating through each key/value pair of our dictionary and assigning the key to the variable "user" (which corresponds to the username "ignw" in our YAML file), and the value of that user to the "data" variable. "data" in this case is the nested dictionary that contains the attributes that we created for our user. We can then access the key/value pairs of our nested dictionary using the "." notation as depicted.

Execute your Playbook again; did Ansible detect any changes? Should it have?

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ vi group_vars/sea.yaml
ignw@ignw-jumphost:~/my_network_as_code$ vi group_vars/sea.yaml
ignw@ignw-jumphost:~/my_network_as_code$ vi templates/cisco_ios_xe_router.j2
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook generate_configurations.yaml

PLAY [Generate IOS-XE Router Configurations] ****
TASK [template] ****
ok: [acme-sea-rtr1]

PLAY RECAP ****
acme-sea-rtr1 : ok=1    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$
```

Ansible thinks (correctly so) that nothing changed... because nothing changed! This is a good thing that means our logic worked! If you feel like some concrete proof, feel free to delete the output file and execute your Playbook one more time to be safe!

Now we need to try to tackle the NTP server configuration. We can use much of the same logic that we applied to the username, but this time we have one additional challenge -- how do we deal with marking the appropriate name server as "preferred"?

The simplest answer would likely to be defining the ntp server with "preferred" in the string, something like this:

```
ntp_server: 66.228.42.59 prefer
```

This would of course work, but what about other options that you may need to configure in the future?

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ssh -l ignw 10.0.0.5
Password:

acme-sea-rtr1#conf t
Enter configuration commands, one per line. End with CNTL/Z.
acme-sea-rtr1(config)#ntp server 1.1.1.1 ?
  burst    Send a burst when peer is reachable (Default)
  iburst   Send a burst when peer is unreachable (Default)
  key      Configure peer authentication key
  maxpoll  Maximum poll interval
  minpoll  Minimum poll interval
  prefer   Prefer this peer when possible
  source   Interface for source address
  version  Configure NTP version
<cr>     <cr>

acme-sea-rtr1(config)#ntp server 1.1.1.1
```

Something like "source" could come in handy later to ensure that ntp is sourced from a loopback for example. To handle that we can set things up very similar to how we handled the users. Add the following to your "sea.yaml" file:

```
ntp: {
  '66.228.42.59': {
    'prefer': true
  },
  '45.76.244.193': {
    'prefer': false
  },
  '204.9.54.119': {
    'prefer': false
  },
  '173.255.206.154': {
    'prefer': false
  }
}
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
timezone: PST
timezone_offset: -8
timezone_summer: PDT

domain_name: sea.acme.io
name_server_1: 8.8.8.8
name_server_2: 8.8.4.4

users: {
    'ignw': {
        'privilege': 15,
        'password': 'ignw',
    }
}

ntp: {
    '66.228.42.59': {
        'prefer': true
    },
    '45.76.244.193': {
        'prefer': false
    },
    '204.9.54.119': {
        'prefer': false
    },
    '173.255.206.154': {
        'prefer': false
    }
}
```

Notice that we can use booleans for values. This will come in handy in a moment!

In your template file, add the following to iterate over the dictionary, rendering the ntp servers:

```
{% for server, data in ntp.items() %}
ntp server {{ server }}
{% endfor %}
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
!
interface GigabitEthernet2
description Link to acme-sea-asal
ip address 203.0.113.1 255.255.255.192
negotiation auto
no shutdown
!
interface loopback8
ip address 8.8.8.8 255.255.255.255
no shut
!
no ip http server
ip scp server enable
!
ip ssh version 2
!
line vty 0 4
login local
!
ntp master 5
{% for server, data in ntp.items() %}
ntp server {{ server }}{% if data['prefer'] == true %} prefer {% endif %}
{%
}
"templates/cisco_ios_xe_router.j2" 44 lines, 948 characters written
```

This is working well, *but* we need to add in some logic to identify which ntp server we want to be the preferred server. The good news is we don't just have loops in Jinja2, we can also work with conditionals. Modify the previously added block to the following:

```
{% for server, data in ntp.items() %}
ntp server {{ server }}{% if data['prefer'] == true %} prefer {% endif %}

{%
}
```

Now our ntp configurations should look the same as the original base config!

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
no aaa new-model
!
ip domain name sea.acme.io
ip name-server 8.8.8.8
ip name-server 8.8.4.4
!
username ignw privilege 15 password 0 ignw
!
interface GigabitEthernet2
description to acme-sea-asal
ip redirects
 ip address 203.0.113.1 255.255.255.192
negotiation auto
no shutdown
!
interface loopback8
ip address 8.8.8.8 255.255.255.255
no shutdown
!
no ip http server
ip scp server enable
!
ip ssh version 2
!
line vty 0 4
login local
!
ntp master 5
ntp server 66.228.42.59 prefer
ntp server 45.76.244.193
ntp server 204.9.54.119
ntp server 173.255.206.154
!
end
ignw@ignw-jumphost:~/my_network_as_code$
```

We're in the home stretch now, all that is left to variableize is the interface level configurations, and we can more or less copy/paste the same types of logic that we've already crafted to get these working well for us.

Interface level configuration is of course device-specific though, so now instead of storing our variable information in the "group\_vars" directory, we will store this information in "host\_vars". Create a file called "acme\_sea\_rtr1" in the "host\_vars" directory. Note that just like the group variables, the host variable files must match the name of the host -- this is how Ansible ties the variables back to the appropriate host/group.

```
touch host_vars/acme-sea-rtr1
```

Add the following to the newly created file:

```
loopback_interfaces: {
  8: {
    'ip': '8.8.8.8',
    'mask': '255.255.255.255'
  }
}
```

At this point this should look fairly familiar. We're creating a dictionary called (and for) "loopback\_interfaces", with keys that happen to be integers. Normally integers as keys would maybe not be the best idea, however for

loopbacks we know that they will be "named" by the integer ID assigned, so this works OK in this scenario. The value of the loopback ID is a dictionary with the data needed to configure it, just like the users and ntp sections.

Finally, update the template file to reflect the new variables:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ip domain name {{ domain_name }}
ip name-server {{ name_server_1 }}
ip name-server {{ name_server_2 }}
!
{% for user, data in users.items() %}
username {{ user }} privilege {{ data.privilege }} password 0 {{ data.password }}
{% endfor %}
!
interface GigabitEthernet2
description Link to acme-sea-asal
ip address 203.0.113.1 255.255.255.192
negotiation auto
no shutdown
!
{% for interface, data in loopback_interfaces.items() %}
interface loopback{{ interface }}
 ip address {{ data.ip }} {{ data.mask }}
 no shut
{% endfor %}
!
no ip http server
ip scp server enable
!
ip ssh version 2
!
"templates/cisco_ios_xe_router.j2" 47 lines, 1089 characters written
```

Once again, re-running the Playbook should show no changes.

Next, do the same thing for the Ethernet interfaces.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
clock summer-time {{timezone_summer }} recurring
!
no aaa new-model
!
ip domain name {{ domain_name }}
ip name-server {{ name_server_1 }}
ip name-server {{ name_server_2 }}
!
{% for user, data in users.items() %}
username {{ user }} privilege {{ data.privilege }} password 0 {{ data.password }}
{% endfor %}
!
{% for interface, data in ethernet_interfaces.items() %}
interface {{ interface }}
 ip address {{ data.ip }} {{ data.mask }}
 negotiation auto
 no shutdown
{% endfor %}
!
{% for interface, data in loopback_interfaces.items() %}
interface loopback{{ interface }}
 ip address {{ data.ip }} {{ data.mask }}
 no shutdown
{% endfor %}
!
no ip http server
ip scp server enable
!
ip ssh version 2
```

Almost there... one thing that is not ideal about templatizing things is that if our templates contain some field, they expect to receive that value in order to appropriately render the data. This is OK because, as we've seen we can add conditional logic to only apply a configuration *if* it exists in the variables for that device or group.

## Jinaj2 Include

One extra neat thing about Jinja2 is that we can include templates in other templates. This is a handy way to keep things a bit more tidy by keeping configuration sections in their own sections and then including them all into a "master" template.

To round out the router configurations, let's create one more template called "generic\_l3\_interface\_configs.j2", and stash some basic interface-level configuration templates in there for things like descriptions, proxy-arp, ip redirects, etc..

```
touch templates/generic_l3_interface_configs.j2
```

Add the following to the new file:

```
{% if 'description' in data.keys() %}
description {{ data['description'] }}

{% endif %}

{% if 'redirects' in data.keys() %}
{% if data['redirects'] == true %}
  ip redirects
{% elif data['redirects'] == false %}
  no ip redirects
{% endif %}
{% endif %}

{% if 'unreachables' in data.keys() %}
{% if data['unreachables'] == true %}
  ip unreachables
{% elif data['unreachables'] == false %}
  no ip unreachables
{% endif %}
{% endif %}

{% if 'proxy-arp' in data.keys() %}
{% if data['proxy-arp'] == true %}
  ip proxy-arp
{% elif data['proxy-arp'] == false %}
  no ip proxy-arp
{% endif %}
{% endif %}
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
{% if 'description' in data.keys() %}
  description {{ data['description'] }}
{% endif %}
{% if 'redirects' in data.keys() %}
  {% if data['redirects'] == true %}
    ip redirects
  {% elif data['redirects'] == false %}
    no ip redirects
  {% endif %}
  {% endif %}
  {% if 'unreachables' in data.keys() %}
    {% if data['unreachables'] == true %}
      ip unreachables
    {% elif data['unreachables'] == false %}
      no ip unreachables
    {% endif %}
    {% endif %}
  {% if 'proxy-arp' in data.keys() %}
    {% if data['proxy-arp'] == true %}
      ip proxy-arp
    {% elif data['proxy-arp'] == false %}
      no ip proxy-arp
    {% endif %}
    {% endif %}
  {% endif %}
~"templates/generic_l3_interface_configs.j2" 24 lines, 596 characters written
```

As outlined, this section will see "if" a particular value is in the "data" for any particular interface. To take advantage of this we will also need to update our main template file:

```
{% for interface, data in ethernet_interfaces.items() %}
interface {{ interface }}
  {% include 'generic_l3_interface_configs.j2' ignore missing %}
    ip address {{ data.ip }} {{ data.mask }}
    negotiation auto
    no shutdown
  {% endfor %}
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
!
no aaa new-model
!
ip domain name {{ domain_name }}
ip name-server {{ name_server_1 }}
ip name-server {{ name_server_2 }}
!
{% for user, data in users.items() %}
username {{ user }} privilege {{ data.privilege }} password 0 {{ data.password }}
{% endfor %}
!
{% for interface, data in ethernet_interfaces.items() %}
interface {{ interface }}
{% include 'generic_l3_interface_configs.j2' ignore missing %}
  ip address {{ data.ip }} {{ data.mask }}
  negotiation auto
  no shutdown
{% endfor %}
!
{% for interface, data in loopback_interfaces.items() %}
interface loopback{{ interface }}
  ip address {{ data.ip }} {{ data.mask }}
  no shutdown
{% endfor %}
!
no ip http server
ip scp server enable
!
ip ssh version 2
!
```

Notice how we are using the "include" statement to bring in the other template to this file. We also have the handy option to "ignore missing" templates.

Finally, add a setting for ip redirects and a description to the host\_vars file for your router to take advantage of the new functionality.

## ASA and NX-OS Templates

Rather than going through this entire exercise again for the ASA and NX-OS templates, copy from the `~/templates` directory the "cisco\_asa.j2" and "cisco\_nxos.j2" files into your templates directory (`~/my_network_as_code/templates`). Also copy the "generic\_I2\_interface\_configs.j2" file over.

```
cd ~/templates  
cp *.j2 ~/my_network_as_code/templates/
```

```
ignw@ignw-jumphost: ~/templates
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/templates$ 
ignw@ignw-jumphost:~/templates$ 
ignw@ignw-jumphost:~/templates$ ls
cisco_asa.j2 cisco_nxos.j2 generic_l2_interface_configs.j2
ignw@ignw-jumphost:~/templates$ ls ~/my_network_as_code/templates/
ignw@ignw-jumphost:~/templates$ cp *.j2 ~/my_network_as_code/templates/
ignw@ignw-jumphost:~/templates$ ls ~/my_network_as_code/templates/
cisco_asa.j2 cisco_nxos.j2 generic_l2_interface_configs.j2
ignw@ignw-jumphost:~/templates$ 
```

Take a peak at the freshly copied templates. They should be more or less similar to the template we've already built -- taking advantage of the same logic we've already been working with.

One thing worth calling out is the creation of users; this varies slightly across the three platforms. Take a look at the three "flavors" of user creation:

```
IOS-XE: username ignw privilege 15 password 0 ignw
ASA: username ignw password $sha512$5000$laGin1QPyEq2nB/7SjQK9w==$EZxnp1UPcv0zP+
Is/zUTpw== pbkdf2 privilege 15
NX-OS: username ignw password 5 $5$5h0TyPZ6$Bf2cvl7dMDTrCSStEbZqbVDaN0J0rMatzzC1
4W7uFQ3  role
e network-admin
```

The most obvious difference is simply the password encryption (or lack thereof), but there are a few other differences: the NX-OS flavor has a "role" instead of a "privilege", and the order of how configurations show up are different between the IOS-XE and ASA configurations.

We've already addressed these challenges in the templates that you copied over:

```
ignw@ignw-jumphost: ~/my_network_as_code/templates
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code/templates$ cat cisco_nxos.j2
hostname {{ inventory_hostname }}
!
{% for feature in features %}
feature {{ feature }}
{% endfor %}
!
no password strength check
{% for user, data in users.items() %}
username {{ user }} password 0 {{ data.password }} role {{ data.role }}
{% endfor %}
!
ip domain-lookup
ip domain-name {{ domain_name }}
!
ip route 0.0.0.0/0 10.255.255.1
!
{% for context, data in routes.items() %}
{% if context == 'default' %}
{% for prefix, data in data.items() %}
ip route {{ prefix }} {{ data.destination }}
{% endfor %}
{% endif %}
{% endfor %}
vrf context {{ context }}
{% for prefix, data in data.items() %}
  ip route {{ prefix }} {{ data.destination }}
{% endfor %}
!
{% for vlan, data in l2_vlans.items() %}
vlan{{ vlan }}
{% if data['name'] %}
```

How do you suppose we can update our group (sea) variables to reflect adding in the additional "data.role" element?

Modify your "sea.yaml" group\_vars file to add the "role" field to your users dictionary:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
timezone: PST
timezone_offset: -8
timezone_summer: PDT

domain_name: sea.acme.io
name_server_1: 8.8.8.8
name_server_2: 8.8.4.4

users: {
    'ignw': {
        'privilege': 15,
        'password': 'ignw',
        'role': 'network-admin'
    }
}

ntp: {
    '66.228.42.59': {
        'prefer': true
    },
    '45.76.244.193': {
        'prefer': false
    },
    '204.9.54.119': {
        'prefer': false
    },
    '173.255.206.154': {
        'prefer': false
    }
}
"group_vars/sea.yaml" 30 lines, 468 characters
```

The NX-OS template also has a few other simple differences -- support for enabling features, dealing with layer 2 VLANs, and handling routing/VRFs in a different way from IOS-XE.

Copy the following into a new host\_vars file for the NX-OS device -- remember that the host\_vars file name must match the name of the ansible inventory name exactly.

```

features:
  - interface-vlan
  - scp-server
  - nxapi

l2_vlans: {
  '1000': {
    'name': 'routing_to_acme_sea_asa1'
  }
}

l2_ethernet_interfaces: {
  'Ethernet1/2': {
    'mode': 'trunk',
    'native_vlan': 1000,
    'description': 'to acme-sea-asa1'
  }
}

svi_interfaces: {
  '1000': {
    'ip': '10.255.255.2',
    'mask': '255.255.255.240',
    'redirects': true,
    'description': 'routing to acme-sea-asa1'
  }
}

```

From the snippet above we can see that we have a list of features to enable, and we have created dictionary objects for "l2\_vlans", "l2\_ethernet\_interfaces", and "svi\_interfaces" just as we have done before. Take a look at the template to make sure you understand how all the variables are being rendered by Jinja.

Add one more section to the "acme-sea-nxos1.yaml" file you just created:

```

routes: {
    'default': {
        '0.0.0.0 0.0.0.0': {
            'destination': '10.255.255.1'
        }
    },
    'management': {
        '1.1.1.1 255.255.255.255': {
            'destination': 'null0'
        }
    }
}

```

As you can see this is the dictionary object containing the routing information for the the acme-sea-nxos1 device. In this case, due to the way Nexus handles static routing we have to be a little bit careful! Routes in the "default" VRF (or what would be the global routing table on IOS/IOS-XE devices) are configured flatly in the configuration like so:

```
ip route 0.0.0.0 0.0.0.0 10.255.255.1
```

For any route in a VRF other than default the configuration is slightly different:

```
vrf context management
ip route 1.1.1.1 255.255.255.255 null0
```

*Note:* In general in these labs we will be completely leaving the management access of our devices alone, however in order to showcase this difference the above route will be created.

To cope with this the template for routing is a bit more complex than what we've created thus far:

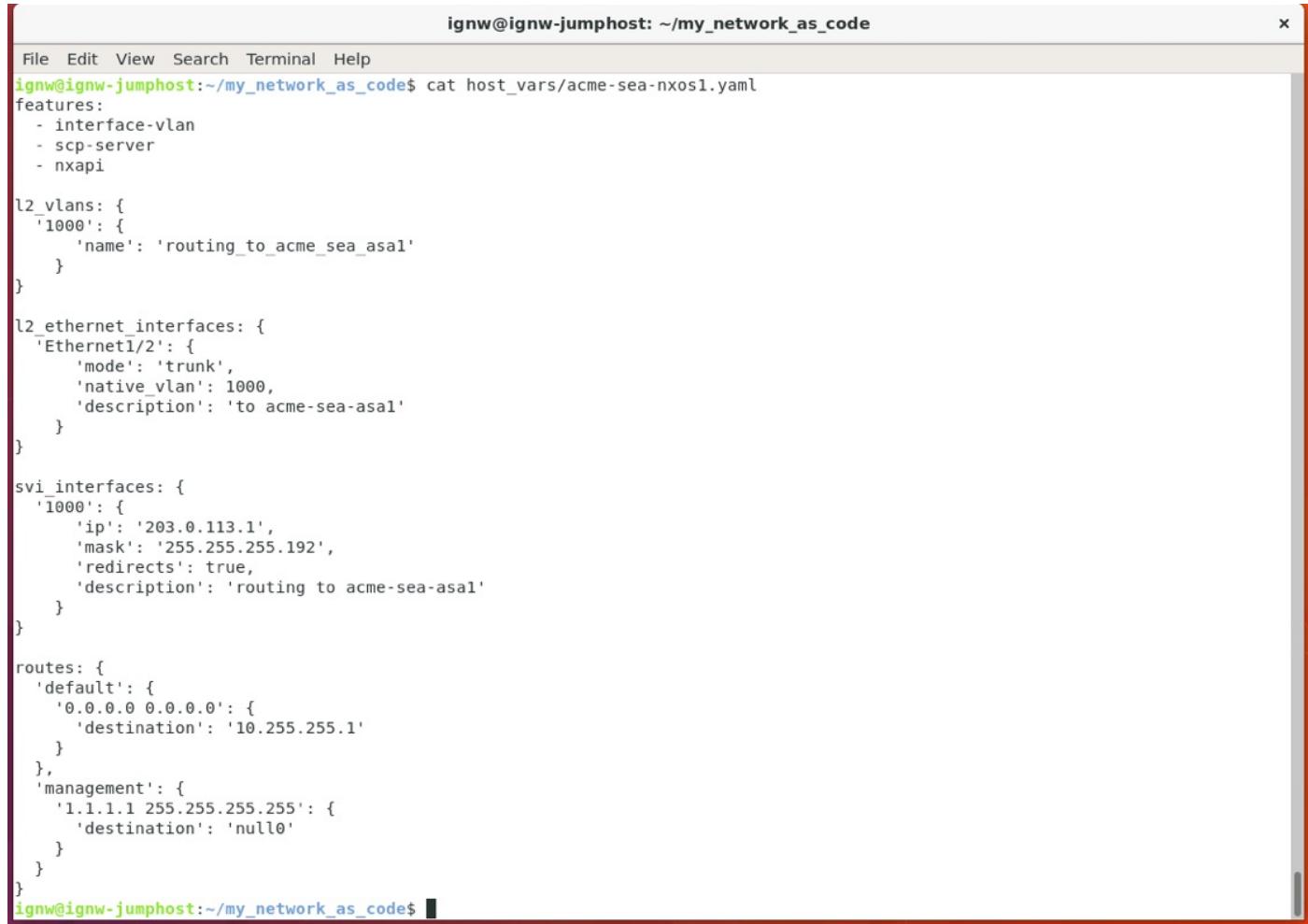
```

{% for context, data in routes.items() %}
{% if context == 'default' %}
{% for prefix, data in data.items() %}
ip route {{ prefix }} {{ data.destination }}
{% endfor %}
{% else %}
vrf context {{ context }}
{% for prefix, data in data.items() %}
    ip route {{ prefix }} {{ data.destination }}
{% endfor %}
{% endif %}
{% endfor %}

```

If the context is "default" then we will render the route in the "global" section of the configuration file; "else" we will enter the "vrf context {{ context }}" section of the configuration to program the routes.

When you're done, your "acme-sea-nxos1.yaml" host\_vars file should look similar to the following:



```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ cat host_vars/acme-sea-nxos1.yaml
features:
- interface-vlan
- scp-server
- nxapi

l2_vlans: {
'1000': {
'name': 'routing_to_acme_sea_asal'
}
}

l2_ethernet_interfaces: {
'Ethernet1/2': {
'mode': 'trunk',
'native_vlan': 1000,
'description': 'to acme-sea-asal'
}
}

svi_interfaces: {
'1000': {
'ip': '203.0.113.1',
'mask': '255.255.255.192',
'redirects': true,
'description': 'routing to acme-sea-asal'
}
}

routes: {
'default': {
'0.0.0.0 0.0.0.0': {
'destination': '10.255.255.1'
}
},
'management': {
'1.1.1.1 255.255.255.255': {
'destination': 'null0'
}
}
}
ignw@ignw-jumphost:~/my_network_as_code$
```

Finally, create the host\_vars file for the ASA ("acme-sea-asa1.yaml") and enter the following:

```

ethernet_interfaces: {
    'GigabitEthernet0/0': {
        'ip': '203.0.113.2',
        'mask': '255.255.255.192',
        'nameif': 'outside',
        'security_level': 0,
        'description': 'to acme-sea-rtr1'
    },
    'GigabitEthernet0/1': {
        'ip': '10.255.255.1',
        'mask': '255.255.255.240',
        'nameif': 'inside',
        'security_level': 90,
        'description': 'to acme-sea-nxos1'
    }
}

routes: {
    '0.0.0.0 0.0.0.0': {
        'nameif': 'outside',
        'destination': '203.0.113.1'
    },
    '10.0.0.0 255.0.0.0': {
        'nameif': 'inside',
        'destination': '10.255.255.2'
    }
}

```

Again, this should look fairly similar to what we've built already. Routes are handled slightly differently due to the operating system differences, but fundamentally nothing has changed. There are additional interface level configurations to reflect the security settings on the firewall (nameif, security-level), but otherwise the interfaces are similar.

When complete your "acme-sea-asa1.yaml" file should look similar to the following:

```
ignw@ignw-jumphost: ~/my_network_as_code/host_vars
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code/host_vars$ cat acme-sea-asal.yaml
ethernet_interfaces: {
    'GigabitEthernet0/0': {
        'ip': '203.0.113.2',
        'mask': '255.255.255.192',
        'nameif': 'outside',
        'security_level': 0,
        'description': 'to acme-sea-rtr1'
    },
    'GigabitEthernet0/1': {
        'ip': '10.255.255.1',
        'mask': '255.255.255.240',
        'nameif': 'inside',
        'security_level': 90,
        'description': 'to acme-sea-nxos1'
    }
}

routes: {
    '0.0.0.0 0.0.0.0': {
        'nameif': 'outside',
        'destination': '203.0.113.1'
    },
    '10.0.0.0 255.0.0.0': {
        'nameif': 'outside',
        'destination': '10.255.255.2'
    }
}
ignw@ignw-jumphost:~/my_network_as_code/host_vars$
```

All that is left is to update our Playbook to generate the configurations for the new devices. For now we will simply add a new Play that is nearly identical to our previous Play:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
---
- name: Generate IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  tasks:
    - template:
        src=templates/cisco_ios_xe_router.j2
        dest=configs/{{ inventory_hostname }}
```

```
- name: Generate NX-OS Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: no
  tasks:
    - template:
        src=templates/cisco_nxos.j2
        dest=configs/{{ inventory_hostname }}
```

```
~
```

Executing your Playbook should now generate the "acme-sea-nxos1" base configuration.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook generate_configurations.yaml
PLAY [Generate IOS-XE Router Configurations] ****
TASK [template] ****
ok: [acme-sea-rtr1]
PLAY [Generate NX-OS Configurations] ****
TASK [template] ****
changed: [acme-sea-nxos1]
PLAY RECAP ****
acme-sea-nxos1      : ok=1    changed=1    unreachable=0    failed=0
acme-sea-rtr1       : ok=1    changed=0    unreachable=0    failed=0
ignw@ignw-jumphost:~/my_network_as_code$
```

Create one last Play for the ASA configurations and re-execute your Playbook:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook generate_configurations.yaml
PLAY [Generate IOS-XE Router Configurations] ****
TASK [template] ****
ok: [acme-sea-rtr1]
PLAY [Generate NX-OS Configurations] ****
TASK [template] ****
ok: [acme-sea-nxos1]
PLAY [Generate ASA Configurations] ****
TASK [template] ****
changed: [acme-sea-asal]
PLAY RECAP ****
acme-sea-asal      : ok=1    changed=1    unreachable=0    failed=0
acme-sea-nxos1     : ok=1    changed=0    unreachable=0    failed=0
acme-sea-rtr1      : ok=1    changed=0    unreachable=0    failed=0
ignw@ignw-jumphost:~/my_network_as_code$
```

# Deploying Configurations Programmatically

## Overview and Objectives

In this lab you will work with Ansible and the Ansible NAPALM module to deploy configurations to devices via a Playbook. This will ultimately be orchestrated by Jenkins to automate the entire process.

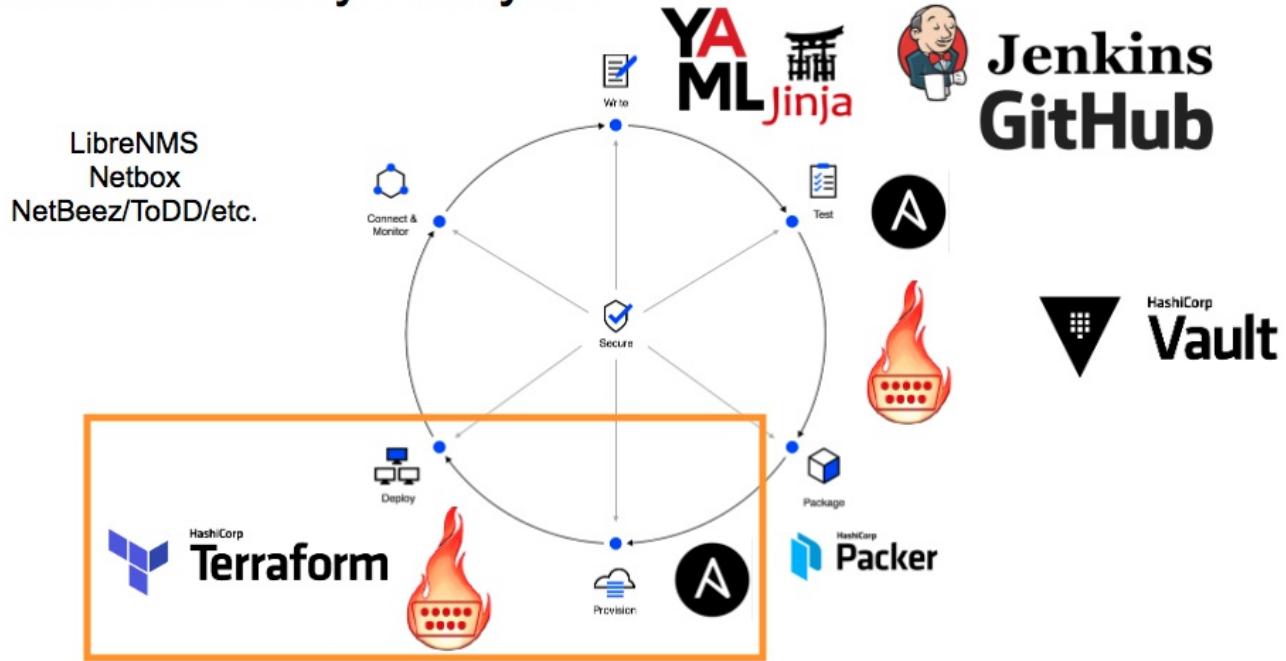
Objectives:

- Create Ansible Plays to deploy configurations to each device type in the lab
- Push configuration files into GitHub

## Network Delivery Lifecycle Overview

Now that we are looking at actually pushing configurations out, we're clearly moving into the "provision" and "deploy" phases of the Network Delivery Lifecycle Overview!

## Network Delivery Lifecycle



## Intro to NAPALM Ansible

Now that we've gone through great lengths to template baseline configurations, and built in the ability to generate functional configurations from simple variable files, we need to figure out how we can "push" those configurations to real network devices.

As with nearly everything we're doing, there are many ways to go about this!! Traditionally many network teams have used tools such as SolarWinds or Cisco Prime. These tools are fine, but are a bit more difficult to tie into an open-source tool chain, or to extend to bring custom functionality that may be required in any given environment.

We've already seen how NAPALM can be used to diff configurations so we know what changes are happening. NAPALM can also be used to *push* those configuration changes via merge or replace operations. This is great, but if we are to use NAPALM directly we will need to be writing pure Python scripts, pulling in inventory information as we go. To keep things relatively simple in the lab environment we'll stick with Ansible as a way to handle our inventory and variables. Happily there is a NAPALM module for Ansible!

The "napalm-ansible" module is not a part of Ansible core, this means it will need to be installed on any system wishing to take advantage of it. This has been done for you, but the general installation process is as follows:

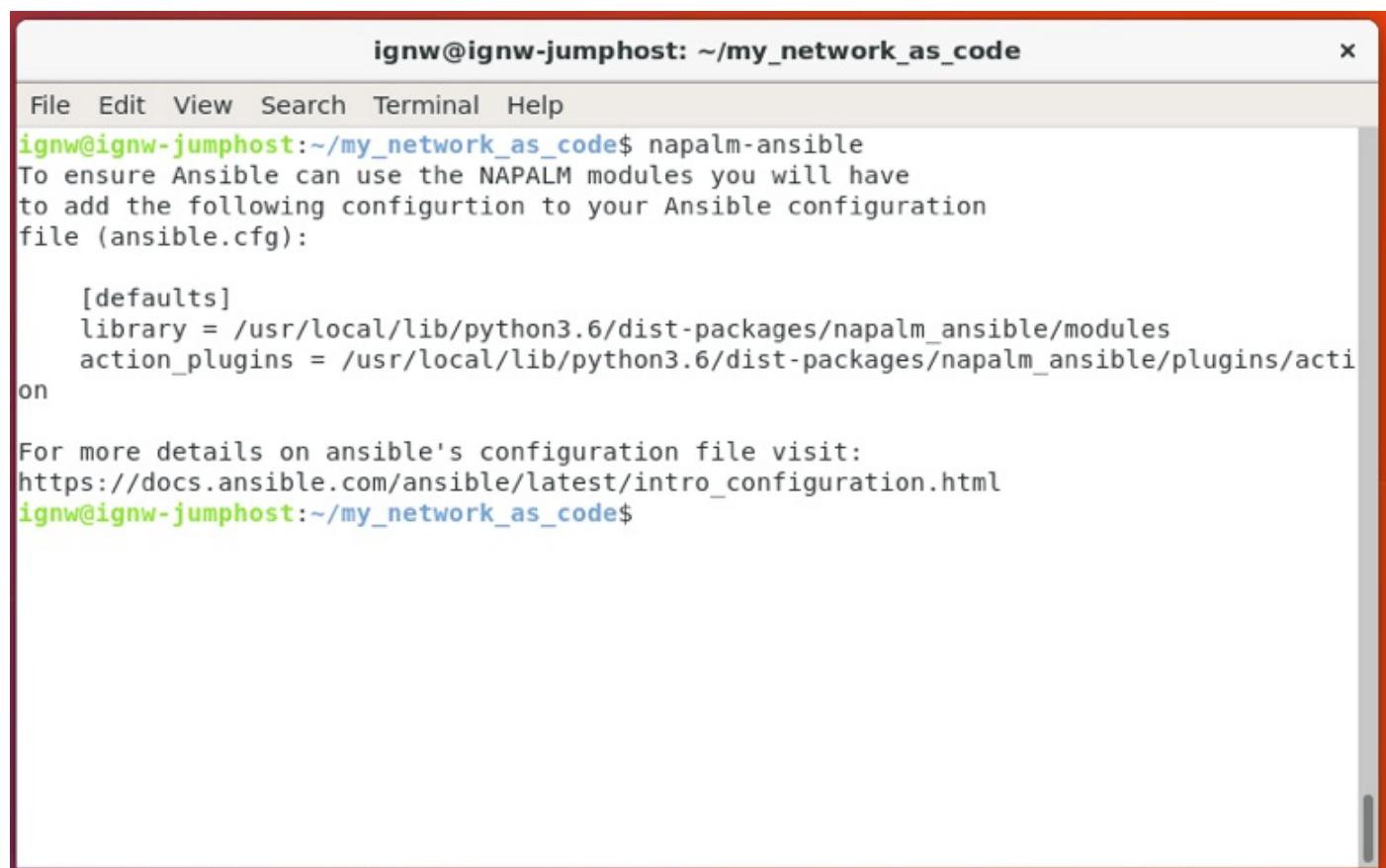
```
pip install napalm-ansible
```

Tough huh?

*Note:* At the time of writing this there are a few open Pull Requests to mitigate a few napalm-ansible Python3 bugs so the lab environment is running a forked version of the module in order to work around these. The fixes will likely be pulled into the master branch shortly (time of writing: April 2018)

## Setting up NAPALM Ansible

As napalm-ansible is *not* part of Ansible core, we need to tell our Ansible configuration where we are storing these third party modules. napalm-ansible has a handy feature to help us know what we need to add to our ansible.cfg file. Simply type "napalm-ansible" in a terminal and napalm-ansible will let us know where things are installed and what we should be entering into our config file.



```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ napalm-ansible
To ensure Ansible can use the NAPALM modules you will have
to add the following configuration to your Ansible configuration
file (ansible.cfg):
[defaults]
library = /usr/local/lib/python3.6/dist-packages/napalm_ansible/modules
action_plugins = /usr/local/lib/python3.6/dist-packages/napalm_ansible/plugins/acti
on
For more details on ansible's configuration file visit:
https://docs.ansible.com/ansible/latest/intro_configuration.html
ignw@ignw-jumphost:~/my_network_as_code$
```

Add the following lines to your "ansible.cfg" file:

```
library = /usr/local/lib/python3.6/dist-packages/napalm_ansible/modules
action_plugins = /usr/local/lib/python3.6/dist-packages/napalm_ansible/plugins/action
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
[defaults]
ansible_python_interpreter=/usr/bin/python3
host_key_checking=False
host_key_auto_add=True
retry_files_enabled=False
inventory=inventory
library = /usr/local/lib/python3.6/dist-packages/napalm_ansible/modules
action_plugins = /usr/local/lib/python3.6/dist-packages/napalm_ansible/plugins/action
"ansible.cfg" 9 lines, 307 characters
```

With that out of the way we can begin crafting a new Playbook to deploy our configurations using napalm-ansible. Create a new file called "deploy\_configurations.yaml" and open it for editing.

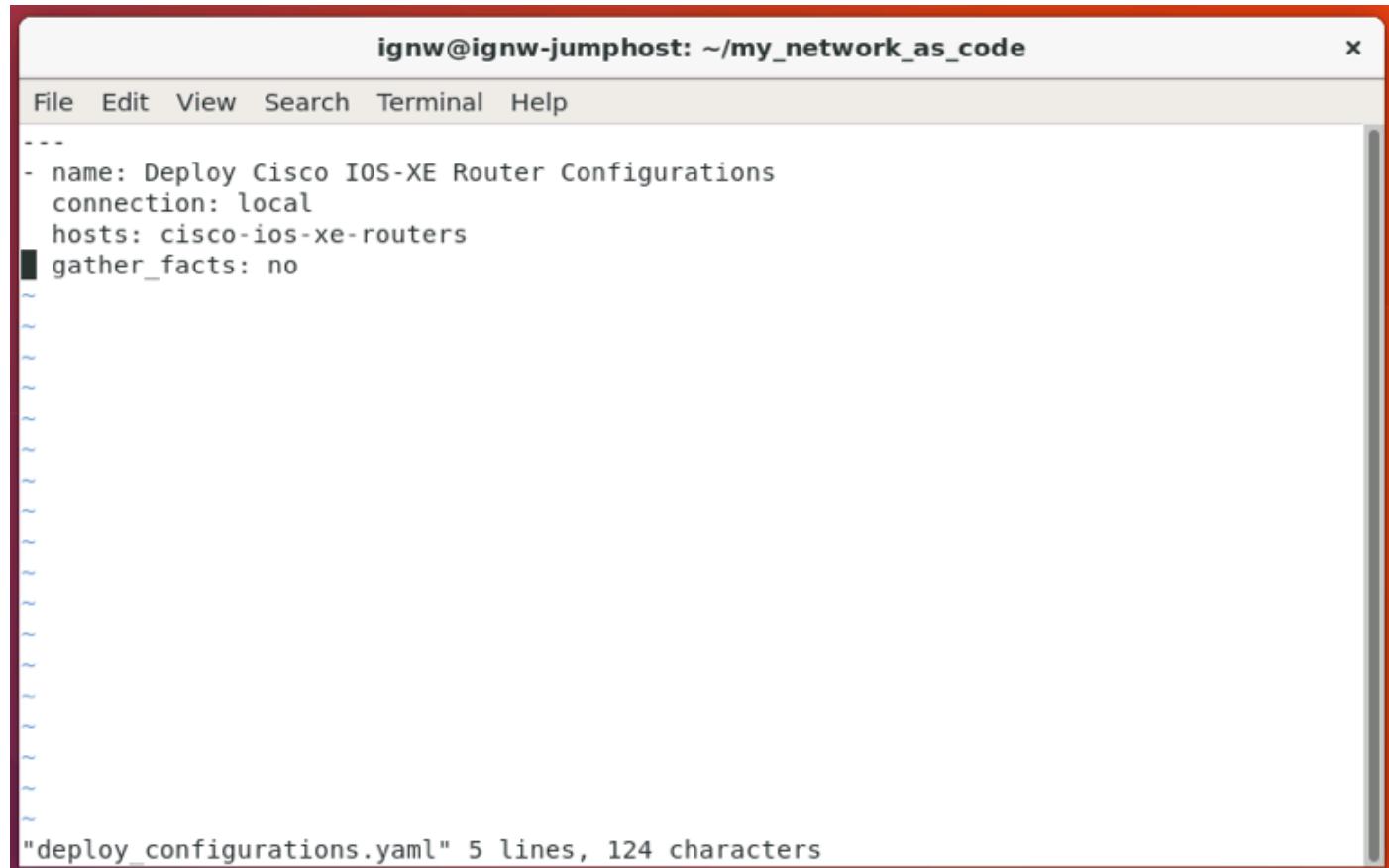
```
touch deploy_configurations.yaml
vi deploy_configurations.yaml
```

## Playbook time -- Getting logged in

Create a Play just like we've done previously:

```
---
- name: Deploy Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
```

Just like before we'll disable "gather\_facts", and set our "connection" to local. For our hosts, we'll start simply with the IOS-XE router.



The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~/my\_network\_as\_code". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area of the terminal displays the following YAML code:

```
---
- name: Deploy Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
```

Below the code, there are several small blue question mark icons. At the bottom of the terminal window, the status message "deploy configurations.yaml" 5 lines, 124 characters" is visible.

Next we'll need to define our credentials and connection parameters for napalm-ansible to connect with our router. The napalm-ansible Github page has a readme file that contains some handy examples for us, you can find that [here](#).

# Examples

Example to retrieve facts from a device

```
- name: get facts from device
  napalm_get_facts:
    hostname={{ inventory_hostname }}
    username={{ user }}
    dev_os={{ os }}
    password={{ passwd }}
    filter='facts,interfaces,bgp_neighbors'
  register: result

- name: print data
  debug: var=result
```

Create a "vars" dictionary that contains the hostname (ip), user, operating system, and password:

```
vars:
  creds:
    hostname: "10.0.0.5"
    username: "ignw"
    password: "ignw"
    dev_os: "ios"
```

For now, to get things going, we'll just stick with static values here, but we will improve upon that shortly!

Next we need to setup a task to have napalm-ansible actually do something (connect to a router) -- and we can start with a simple diff, the way we did with the Python script previously.

Create a task in the Playbook using the "napalm\_install\_config" module. This module requires a "provider" argument which is the dictionary of connection information we previously created.

```
tasks:
  napalm_install_config:
    provider: "{{ creds }}"
```

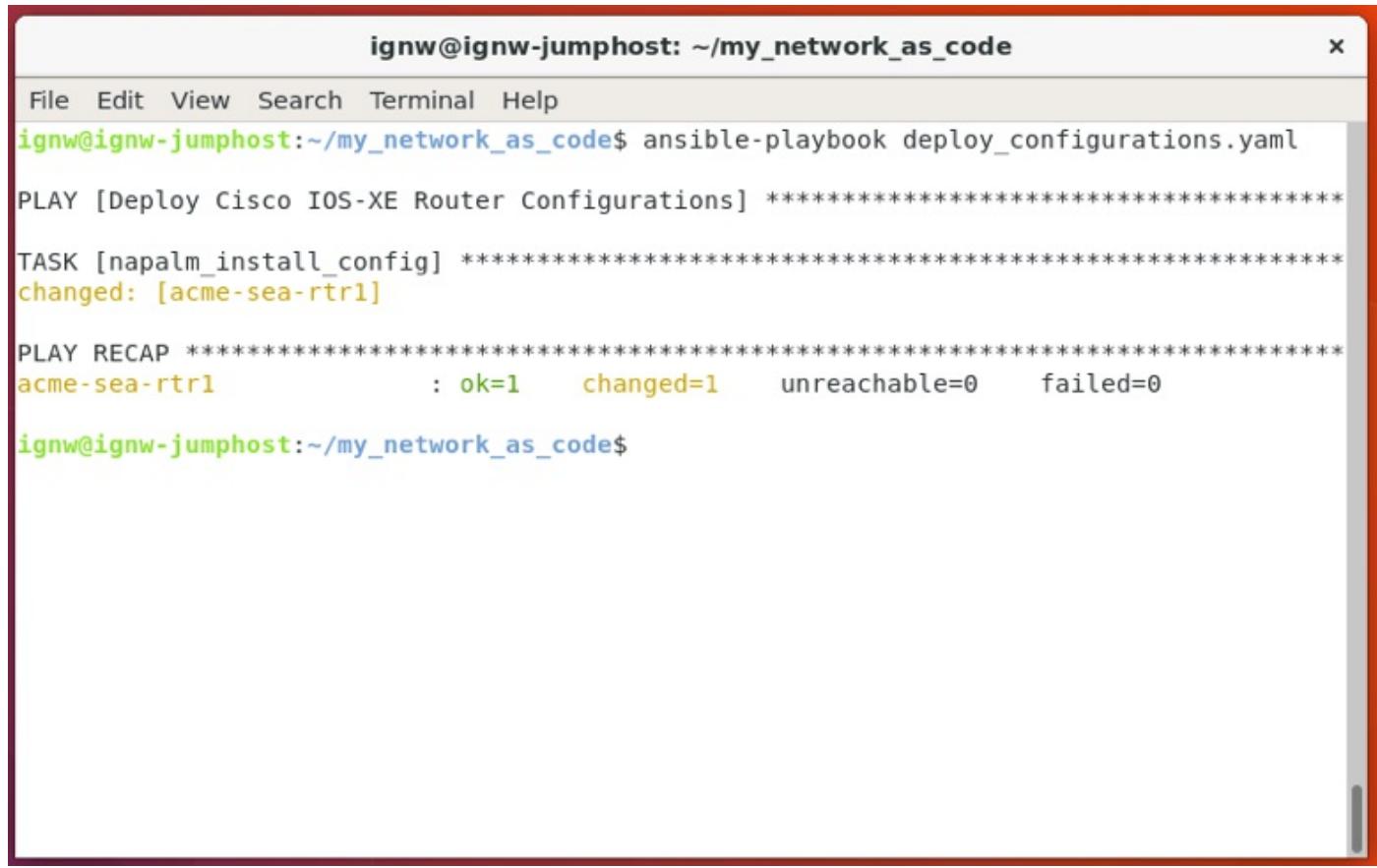
Next we need to provide the configuration file we would like to use. We also have "commit\_changes" and "replace\_config" bools that we can adjust. From the documentation ([here](#)), the "commit\_changes" parameter: "If set to True the configuration will be actually merged or replaced. If set to False, we will not apply the changes, just check and report the diff". So to start, we'll set this to "False" so we are not making any changes just yet.

```
config_file: "configs/{{ inventory_hostname }}"
commit changes: False
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
---
- name: Deploy Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  vars:
    creds:
      hostname: "10.0.0.5"
      username: "ignw"
      password: "ignw"
      dev_os: "ios"
  tasks:
    - napalm_install_config:
        provider: "{{ creds }}"
        config_file: "configs/{{ inventory_hostname }}"
        commit_changes: False
~
```

We're now ready to execute our new Playbook.

```
ansible-playbook deploy configurations.yaml
```



A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/my\_network\_as\_code". The window shows the output of an Ansible playbook named "deploy\_configurations.yaml". The output indicates that a Cisco IOS-XE Router Configuration was deployed to a host named "acme-sea-rtr1". The status summary at the end shows "ok=1" and "changed=1".

```
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook deploy_configurations.yaml
PLAY [Deploy Cisco IOS-XE Router Configurations] ****
TASK [napalm_install_config] ****
changed: [acme-sea-rtr1]

PLAY RECAP ****
acme-sea-rtr1 : ok=1    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$
```

## NAPALM Diffs in Ansible

A good start, but we need to know what will be changing! Just like we did with the "pure" Python version of NAPALM, we need to output our diff. Let's register what the diff would be to an Ansible variable and then iterate over that printing out the changes.

Add the following to your Playbook:

```
register: napalm_diff
- debug:
  msg: "{{ item }}"
  with_items: "{{ napalm_diff.msg.split('\n') }}"
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
---
- name: Deploy Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  vars:
    creds:
      hostname: "10.0.0.5"
      username: "ignw"
      password: "ignw"
      dev_os: "ios"
  tasks:
    - napalm_install_config:
        provider: "{{ creds }}"
        config_file: "configs/{{ inventory_hostname }}"
        commit_changes: False
        register: napalm_diff
    - debug:
        msg: "{{ item }}"
        with_items: "{{ napalm_diff.msg.split('\n') }}"
~
~
~"deploy_configurations.yaml" 20 lines, 512 characters
```

As you can see, we are splitting the "msg" variable returned from the napalm\_install\_config module on new lines, and printing them out as a message.

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+interface loopback8"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ ip address 8.8.8.8 255.255.255.255"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "- no shutdown"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "-no ip http server"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+login local"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp master 5"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp 66.228.42.59 prefer"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp 45.76.244.193"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp 204.9.54.119"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp 173.255.206.154"
}

PLAY RECAP ****
acme-sea-rtr1 : ok=2    changed=1    unreachable=0    failed=0
ignw@ignw-jumphost:~/my_network_as_code$ 

```

At this point we can finally see what changes would be implemented after all of the hard work creating our templates! This should look pretty familiar as it is very similar to the output from NAPALM previously.

## Committing Changes

Now that we know what is happening, modify the Playbook to actually push the configuration to the router. The default behavior of napalm-ansible (for the napalm\_install\_config module) is to merge -- not replace -- the configuration. This is perfect since that is exactly what we want to do. Even though it is the default setting, let's go ahead and "hard code" that anyway... just to be safe!

Change the "commit\_changes" to True, and add the following line to your task:

```
replace_config: False
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
-- 
- name: Deploy Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  vars:
    creds:
      hostname: "10.0.0.5"
      username: "ignw"
      password: "ignw"
      dev_os: "ios"
  tasks:
    - napalm_install_config:
        provider: "{{ creds }}"
        config_file: "configs/{{ inventory_hostname }}"
        commit_changes: True
        replace_config: False
        register: napalm_diff
    - debug:
        msg: "{{ item }}"
        with_items: "{{ napalm_diff.msg.split('\n') }}"
"deploy_configurations.yaml" 21 lines, 542 characters
```

Execute your Playbook.

```
ansible-playbook deploy_configurations.yaml
```

If it seems like its taking a while, like a really long while.... you may have chopped off the "end" line at the very end of the configuration. Cisco IOS expects to see this when using the archive functionality that NAPALM uses. If you see something like the following, check to make sure your config has "end" at the end.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help

ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook deploy_configurations.yaml
PLAY [Deploy Cisco IOS-XE Router Configurations] ****
TASK [napalm_install_config] ****
fatal: [acme-sea-rtr1]: FAILED! => {"changed": false, "msg": "cannot install config: Search pattern never detected in send _command_expect: [>#]\\s*$"}
PLAY RECAP ****
acme-sea-rtr1 : ok=0    changed=0    unreachable=0    failed=1
ignw@ignw-jumphost:~/my_network_as_code$
```

If everything went as planned your output should look similar to that shown below:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help

PLAY RECAP ****
acme-sea-rtr1 : ok=0    changed=0    unreachable=0    failed=1

ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook deploy_configurations.yaml

PLAY [Deploy Cisco IOS-XE Router Configurations] ****
TASK [napalm_install_config] ****
changed: [acme-sea-rtr1]

TASK [debug] ****
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ip name-server 8.8.8.8"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ip name-server 8.8.4.4"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+interface GigabitEthernet2"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "- no shutdown"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+interface loopback8"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "- no shut"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp server 0.us.pool.ntp.org prefer"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp server 1.us.pool.ntp.org"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp server 2.us.pool.ntp.org"
}
ok: [acme-sea-rtr1] => (item=None) => {
    "msg": "+ntp server 3.us.pool.ntp.org"
}

PLAY RECAP ****
acme-sea-rtr1 : ok=2    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$
```

## Moving on to NX-OS

Now that we've got the IOS-XE merge rolling, it is time to tackle the NX-OS configurations. Now would also be a good time to add tags to our Plays so that we can execute only the Plays that we are testing as we work.

Copy the "Deploy Cisco IOS-XE Router Configurations" Play, and modify it to be for the NX-OS group. Add a tag to each play with the corresponding device type:

```

---
- name: Deploy Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  vars:
    creds:
      hostname: "10.0.0.5"
      username: "ignw"
      password: "ignw"
      dev_os: "ios"
  tags: ios-xe
  tasks:
    - napalm_install_config:
        provider: "{{ creds }}"
        config_file: "configs/{{ inventory_hostname }}"
        commit_changes: True
        replace_config: False
        register: napalm_diff
    - debug:
        msg: "{{ item }}"
        with_items: "{{ napalm_diff.msg.split('\n') }}"
- name: Deploy Cisco NX-OS Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: no
  vars:
    creds:
      hostname: "10.0.0.6"
      username: "ignw"
      password: "ignw"
      dev_os: "nxos_ssh"
  tags: nxos
  tasks:
    - napalm_install_config:
        provider: "{{ creds }}"
        config_file: "configs/{{ inventory_hostname }}"
        commit_changes: True
        replace_config: False
        register: napalm_diff
    - debug:
        msg: "{{ item }}"
        with_items: "{{ napalm_diff.msg.split('\n') }}"

```

**Note:** Make sure you change the "dev\_os" to "nxos\_ssh"!

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
---
- name: Deploy Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  vars:
    creds:
      hostname: "10.0.0.5"
      username: "ignw"
      password: "ignw"
      dev_os: "ios"
  tags: ios-xe
  tasks:
    - napalm_install_config:
        provider: "{{ creds }}"
        config_file: "configs/{{ inventory_hostname }}"
        commit_changes: True
        replace_config: False
        register: napalm_diff
    - debug:
        msg: "{{ item }}"
        with_items: "{{ napalm_diff.msg.split('\\n') }}"
- name: Deploy Cisco NX-OS Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: no
  vars:
    creds:
      hostname: "10.0.0.6"
      username: "ignw"
      password: "ignw"
      dev_os: "nxos ssh"
  tags: nxos
  tasks:
    - napalm_install_config:
        provider: "{{ creds }}"
        config_file: "configs/{{ inventory_hostname }}"
        commit_changes: True
        replace_config: False
        register: napalm_diff
    - debug:
        msg: "{{ item }}"
        with_items: "{{ napalm_diff.msg.split('\\n') }}"
```
"deploy_configurations.yaml" 44 lines, 1096 characters

```

One final challenge before executing the NX-OS section of the Playbook. The way NAPALM handles merge configurations with the nxos\_ssh driver is by use of Netmiko. Changing the hostname modifies the prompt that Netmiko expects to see, thus causing timeouts. This is not an issue with config *replacement* but for now we want to simply have merges, so we need to address this. SSH to the device and modify the hostname to match the Ansible inventory.

```

ssh 10.0.0.6
conf t
hostname acme-sea-nxos1
wr

```

Execute just the "Deploy Cisco NX-OS Configurations" Play by running the Playbook with the "--tags" argument:

```
ansible-playbook deploy_configurations.yaml --tags nxos
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook deploy_configurations.yaml --tags nxos
PLAY [Deploy Cisco IOS-XE Router Configurations] ****
PLAY [Deploy Cisco NX-OS Configurations] ****
TASK [napalm_install_config] ****
changed: [acme-sea-nxos1]

TASK [debug] ****
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": "username ***** password 0 ***** role network-admin"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": "ip route 0.0.0.0 0.0.0.0 10.255.255.1"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": " ip route 1.1.1.1 255.255.255.255 null0"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": " name to_acme_sea_asal"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": "interface vlan1000"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": " ip address 203.0.113.1 255.255.255.192"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": " no shutdown"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": " switchport mode trunk"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": " switchport trunk native vlan 1000"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": " no shutdown"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": "boot nxos bootflash:/nxos.7.0.3.I7.3.bin"
}

PLAY RECAP ****
acme-sea-nxos1      : ok=2    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$ ignw@ignw-jumphost:~/my_network_as_code$
```

## ASA NAPALM Woes

Excellent! Now we need to get the configuration pushed to the firewall. There is a minor issue with this... NAPALM does not support the ASA operating system. Fortunately we can fall back to the "regular" (Core) Ansible module "asa\_config". Check out the documentation [here](#), what do you think is the best way to approach getting our templated configuration on the firewall?

Generally, "asa\_config" is used to manage configurations that are stored in tasks within Plays, however, since we took the route of templating we need to do things a bit differently...

The "src" option allows us to pass a configuration file to the asa\_config module. This sounds like just the thing we need!

|                  |                                      |                                                                                                                                                                                                                                                                                                                                                |
|------------------|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>src</code> | <b>Default:</b><br><code>None</code> | Specifies the source path to the file that contains the configuration or configuration template to load. The path to the source file can either be the full path on the Ansible control host or a relative path from the playbook or role root directory. This argument is mutually exclusive with <code>lines</code> , <code>parents</code> . |
|------------------|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Once again, copy the previous Play in your Playbook, and paste it in. We'll modify this to fit our needs.

```
- name: Deploy Cisco ASA Configurations
  connection: local
  hosts: cisco-asa
  gather_facts: no
  vars:
    creds:
      host: "10.0.0.8"
      username: "ignw"
      password: "ignw"
  tags: asa
  tasks:
    - asa_config:
        provider: "{{ creds }}"
        src: "configs/{{ inventory_hostname }}"
```

Since we are not using the NAPALM module for the ASA (because that doesn't exist) this Play will look a bit different. As you can see we are still naming our Play, using connection local, selecting a host group, and disabling fact gathering. We also still need the device information (ip/hostname, and credentials) of course. Note that the `asa_config` module expects "host" instead of "hostname" though!

Our task is going to be using the "`asa_config`" module, which, just like the NAPALM module, expects the credentials/host information in the "`providers`" argument. Next we need to provide our templated configuration using the "`src`" argument.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
tags: ios-xe
tasks:
- napalm_install_config:
  provider: "{{ creds }}"
  config_file: "configs/{{ inventory_hostname }}"
  commit_changes: True
  replace_config: False
  register: napalm_diff
- debug:
  msg: "{{ item }}"
  with_items: "{{ napalm_diff.msg.split('\n') }}"
- name: Deploy Cisco NX-OS Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: no
  vars:
    creds:
      hostname: "10.0.0.6"
      username: "ignw"
      password: "ignw"
      dev_os: "nxos_ssh"
  tags: nxos
  tasks:
    - napalm_install_config:
      provider: "{{ creds }}"
      config_file: "configs/{{ inventory_hostname }}"
      commit_changes: True
      replace_config: False
      register: napalm_diff
    - debug:
      msg: "{{ item }}"
      with_items: "{{ napalm_diff.msg.split('\n') }}"
- name: Deploy Cisco ASA Configurations
  connection: local
  hosts: cisco-asa
  gather_facts: no
  vars:
    creds:
      host: "10.0.0.8"
      username: "ignw"
      password: "ignw"
      authorize: yes
      auth_pass: "ignw"
  tags: asa
  tasks:
    - asa_config:
      provider: "{{ creds }}"
      src: "configs/{{ inventory_hostname }}"
```

Execute your Playbook using the "asa" tag so we only run the appropriate task:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
x

ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook deploy_configurations.yaml --tags asa
PLAY [Deploy Cisco IOS-XE Router Configurations] ****
PLAY [Deploy Cisco NX-OS Configurations] ****
PLAY [Deploy Cisco ASA Configurations] ****
TASK [asa_config] ****
acme-sea-asa1>ccurred during task execution. To see the full traceback, use -vvv. The error was:
fatal: [acme-sea-asa1]: FAILED! => {"changed": false, "module_stderr": "Traceback (most recent call last):\n  File \"/tmp/ansible_0th55h_c/ansible_modlib.zip/ansible/module_utils/network/asa/asa.py\", line 142, in get_config\n    KeyError: 'show running-config'\n\nDuring handling of the above exception, another exception occurred:\n\n  Traceback (most recent call last):\n    File \"/tmp/ansible_0th55h_c/ansible_module_asa_config.py\", line 308, in <module>\n        main()\n      File \"/tmp/ansible_0th55h_c/ansible_module_asa_config.py\", line 223, in run\n        contents = get_config(module)\n      File \"/tmp/ansible_0th55h_c/ansible_modlib.zip/ansible/module_utils/network/asa/asa.py\", line 145, in get_config\n        File \"/tmp/ansible_0th55h_c/ansible_modlib.zip/ansible/module_utils/connection.py\", line 149, in __rpc__\n        ansible.module_utils.connection.ConnectionError: show running-config\n   ^\r\nERROR: % Invalid input detected at '^' marker.\r\nracme-sea-asa1> \n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 1}

PLAY RECAP ****
acme-sea-asa1 : ok=0    changed=0    unreachable=0    failed=1

ignw@ignw-jumphost:~/my_network_as_code$
```

That does not look very promising... run the Playbook again specifying the "-vvvv" flag so we get all of the verbose (very very very verbose!) output.

```
ansible-playbook deploy_configurations --tags asa -vvvv
```

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
Using module file /usr/local/lib/python3.6/dist-packages/ansible/modules/network/asa/asa_config.py
<acme-sea-asal> ESTABLISH LOCAL CONNECTION FOR USER: ignw
<acme-sea-asal> EXEC /bin/sh -c 'echo ~ && sleep 0'
<acme-sea-asal> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo /home/ignw/.ansible/tmp/ansible-tmp-1526077537.4911718-276681547035933 `\" && echo ansible
-tmp-1526077537.4911718-276681547035933=\"` echo /home/ignw/.ansible/tmp/ansible-tmp-1526077537.4911718-276681547035933 `\" ) && sleep 0'
<acme-sea-asal> PUT /home/ignw/.ansible/tmp/ansible-local-7947baq9zkvg/tmpce3st5l3 TO /home/ignw/.ansible/tmp/ansible-tmp-1526077537.4911718-276681547035
933/asa_config.py
<acme-sea-asal> EXEC /bin/sh -c 'chmod u+x /home/ignw/.ansible/tmp/ansible-tmp-1526077537.4911718-276681547035933 /home/ignw/.ansible/tmp/ansible-tmp-15
26077537.4911718-276681547035933/asa_config.py && sleep 0'
<acme-sea-asal> EXEC /bin/sh -c '/usr/bin/python3 /home/ignw/.ansible/tmp/ansible-tmp-1526077537.4911718-276681547035933/asa_config.py && sleep 0'
<acme-sea-asal> EXEC /bin/sh -c 'rm -f -r /home/ignw/.ansible/tmp/ansible-tmp-1526077537.4911718-276681547035933/ > /dev/null 2>&1 && sleep 0'
The full traceback is:
Traceback (most recent call last):
  File "/tmp/ansible_ajjgx69t/ansible_modlib.zip/ansible/module_utils/network/asa/asa.py", line 142, in get_config
KeyError: 'show running-config'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/tmp/ansible_ajjgx69t/ansible_module_asa_config.py", line 308, in <module>
    main()
  File "/tmp/ansible_ajjgx69t/ansible_module_asa_config.py", line 302, in main
    run(module, result)
  File "/tmp/ansible_ajjgx69t/ansible_module_asa_config.py", line 223, in run
    contents = get_config(module)
  File "/tmp/ansible_ajjgx69t/ansible_modlib.zip/ansible/module_utils/network/asa/asa.py", line 145, in get_config
  File "/tmp/ansible_ajjgx69t/ansible_modlib.zip/ansible/module_utils/connection.py", line 149, in __rpc__
ansible.module_utils.connection.ConnectionError: show running-config

ERROR: % Invalid input detected at '^' marker.
acme-sea-asal>

fatal: [acme-sea-asal]: FAILED! => {
    "changed": false,
    "module_stderr": "Traceback (most recent call last):\n  File \"/tmp/ansible_ajjgx69t/ansible_modlib.zip/ansible/module_utils/network/asa/asa.py\", line 142, in get_config\nKeyError: 'show running-config'\nDuring handling of the above exception, another exception occurred:\n\nTraceback (most recent call last):\n  File \"/tmp/ansible_ajjgx69t/ansible_module_asa_config.py\", line 308, in <module>\n    main()\n  File \"/tmp/ansible_ajjgx69t/ansible_modlib.zip/ansible/module_utils/as
a_config.py\", line 302, in main\n    run(module, result)\n  File \"/tmp/ansible_ajjgx69t/ansible_modlib.zip/ansible/module_utils/as
a_config.py\", line 223, in run\n    contents = get_config(module)\n  File \"/tmp/ansible_ajjgx69t/ansible_modlib.zip/ansible/module_utils/network/asa/asa.py\", line 145, in get_config\n  File \"/tmp/ansible_ajjgx69t/ansible_modlib.zip/ansible/module_utils/connection.py\", line 149, in __rpc__
ansible.module_utils.connection.ConnectionError: show running-config\n",
    "module_stdout": "",
    "msg": "MODULE FAILURE",
    "rc": 1
}

PLAY RECAP ****
acme-sea-asal : ok=0    changed=0    unreachable=0    failed=1
ignw@ignw-jumphost:~/my_network_as_code$ 
```

Hmm... certainly we can't deploy configurations without even getting past the enable prompt. The documentation outlines the "authorize" and "authorize\_pass" options which will allow us to get fully logged in.

We just need to update the Playbook with the appropriate information:

```

authorize: yes
auth_pass: "ignw"
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Terminal Help
username: "ignw"
password: "ignw"
dev_os: "nxos_ssh"
tags: nxos
tasks:
- napalm_install_config:
  provider: "{{ creds }}"
  config_file: "configs/{{ inventory_hostname }}"
  commit_changes: True
  replace_config: False
  register: napalm_diff
- debug:
  msg: "{{ item }}"
  with_items: "{{ napalm_diff.msg.split('\n') }}"
- name: Deploy Cisco ASA Configurations
  connection: local
  hosts: cisco-asa
  gather_facts: no
  vars:
    creds:
      host: "10.0.0.8"
      username: "ignw"
      password: "ignw"
      authorize: yes
      auth_pass: "ignw"
tags: asa
tasks:
- asa_config:
  provider: "{{ creds }}"
  src: "configs/{{ inventory_hostname }}"
```

Execute the Playbook once more.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook deploy_configurations.yaml --tags asa
PLAY [Deploy Cisco IOS-XE Router Configurations] ****
PLAY [Deploy Cisco NX-OS Configurations] ****
PLAY [Deploy Cisco ASA Configurations] ****
TASK [asa_config] ****
changed: [acme-sea-asa1]
PLAY RECAP ****
acme-sea-asa1 : ok=1    changed=1    unreachable=0    failed=0
ignw@ignw-jumphost:~/my_network_as_code$
```

Unfortunately the `asa_config` module does not provide "diffs" in an easily consumable format like NAPALM, but even without that we now have a way to programmatically render and deploy configurations on the ASA platform (plus IOS-XE and NX-OS!). Note that you can run the Playbook with the `-vvvv` switch to get the diffs if you would like.

## Group and Host Variables

Now that we have the foundations in place we need to go back and make a few minor improvements to our Playbook and variable structure to ensure that we have a solid foundation moving forward.

The first thing we can improve upon is how we handle the NAPALM device types. In this itty bitty lab environment dealing with things like hard coding device types is obviously very simple, but if we began to manage hundreds of devices these hard coded things become very tedious to manage.

Create a new file for each of the groups (named after each group) in the `"group_vars"` folder:

```
touch group_vars/cisco-ios-xe-routers.yaml
touch group_vars/cisco-asa.yaml
touch group_vars/cisco-nxos.yaml
```

Within these new variable files, add the following, where "[DEVICE TYPE]" is the appropriate NAPALM device type (ios, nxos\_ssh):

```
device_type: [DEVICE TYPE]
```

Note: NAPALM also supports the NX-API, for now we're using "nxos\_ssh" instead of "nxos" (the NX-API driver) because both the "nxos\_ssh" and the "ios" are backed by the Netmiko library, this is purely for consistency sake, but this could easily be modified to take advantage of NX-API.

Add the device\_type variable even to the ASA file, we may not need it now (since it is not using NAPALM), but for consistency sake we'll have it anyway.

Now we should be able to modify our Playbook to take advantage of the device\_type variables we've created.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
vars:
  creds:
    hostname: "10.0.0.5"
    username: "ignw"
    password: "ignw"
    dev_os: "{{ device_type }}"
tags: ios-xe
tasks:
  - napalm_install_config:
      provider: "{{ creds }}"
      config_file: "configs/{{ inventory_hostname }}"
      commit_changes: True
      replace_config: False
      register: napalm_diff
  - debug:
      msg: "{{ item }}"
      with_items: "{{ napalm_diff.msg.split('\n') }}"
- name: Deploy Cisco NX-OS Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: no
vars:
  creds:
    hostname: "10.0.0.6"
    username: "ignw"
    password: "ignw"
    dev_os: "{{ device_type }}"
tags: nxos
tasks:
  - napalm_install_config:
"deploy_configurations.yaml" 59 lines, 1469 characters written
```

Great! Next, we want to take a look at how to handle credentials.... this is actually a pretty big topic! In a production environment there are many tools available to help securely handle this (things like HashiCorp Vault, Ansible Vault, Amazon Key management Service, and tons of others), and you should definitely take advantage of these to ensure security... but... this is a lab, so we're going to do this in the open to keep things simple.

Let's assume that each site will have their own credentials; maybe a silly assumption, but remember: its a lab! We can now go ahead and add the username and password variables to the group\_vars for our site "sea":

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ cat group_vars/sea.yaml
timezone: PST
timezone_offset: -8
timezone_summer: PDT

domain_name: sea.acme.io
name_server_1: 8.8.8.8
name_server_2: 8.8.4.4

users: {
  'ignw': {
    'privilege': 15,
    'password': 'ignw',
    'role': 'network-admin'
  }
}

ntp: {
  '66.228.42.59': {
    'prefer': true
  },
  '45.76.244.193': {
    'prefer': false
  },
  '204.9.54.119': {
    'prefer': false
  },
  '173.255.206.154': {
    'prefer': false
  }
}

username: ignw
password: ignw
ignw@ignw-jumphost:~/my_network_as_code$
```

You may have noticed that we actually already have this information in the "sea.yaml" file in the "users" dictionary. That is true, and we could absolutely just use that, however the username/password in this context are what Ansible/NAPALM are going to be using to connect to the device. If we had multiple users in our users dictionary (as you probably would in real life) we would have to ensure we selected the correct one which would be a bit more complex. So we'll continue keeping it simple! Again, in production you would likely be using SSH keys and/or Vault or similar to handle this.

Go back and update the Playbook to use the newly created variables:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
connection: local
hosts: cisco-ios-xe-routers
gather_facts: no
vars:
  creds:
    hostname: "10.0.0.5"
    username: "{{ username }}"
    password: "{{ password }}"
    dev_os: "{{ device_type }}"
tags: ios-xe
tasks:
  - napalm_install_config:
      provider: "{{ creds }}"
      config_file: "configs/{{ inventory_hostname }}"
      commit_changes: True
      replace_config: False
      register: napalm_diff
  - debug:
      msg: "{{ item }}"
      with_items: "{{ napalm_diff.msg.split('\n') }}"
- name: Deploy Cisco NX-OS Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: no
  vars:
    creds:
      hostname: "10.0.0.6"
      username: "{{ username }}"
      password: "{{ password }}"
      dev_os: "{{ device_type }}"
```

Since the ASA has an additional argument for the enable password, go ahead and add that to the "sea.yaml" file as well.

```
echo "enable_password: ignw" >> group_vars/sea.yaml
```

And, finally, update the Playbook again:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
username: "{{ username }}"
password: "{{ password }}"
dev_os: "{{ device_type }}"
tags: nxos
tasks:
- napalm_install_config:
  provider: "{{ creds }}"
  config_file: "configs/{{ inventory_hostname }}"
  commit_changes: True
  replace_config: False
  register: napalm_diff
- debug:
  msg: "{{ item }}"
  with_items: "{{ napalm_diff.msg.split('\n') }}"
- name: Deploy Cisco ASA Configurations
  connection: local
  hosts: cisco-asa
  gather_facts: no
  vars:
    creds:
      host: "10.0.0.8"
      username: "{{ username }}"
      password: "{{ username }}"
      authorize: yes
      auth_pass: "{{ enable_password }}"
tags: sea
tasks:
- asa_config:
  provider: "{{ creds }}"
  src: "configs/{{ inventory_hostname }}"
"deploy_configurations.yaml" 59 lines, 1546 characters written
```

Lastly, we need to address the hard coded IP addresses we've used. This is obviously not going to scale! Due to our decision to use Ansible, we have a very simple way to address this: "ansible\_host". The "ansible\_host" variable is commonly stored in the inventory file and is the IP or FQDN that is used to reach the host. Update your inventory file to add the "ansible\_host=X.X.X.X" variable for each host. Note that this only needs to be done at one of the groups (we'll do it at the "sea" group).

And one again, update the Playbook:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
...
- name: Deploy Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  vars:
    creds:
      hostname: "{{ ansible_host }}"
      username: {{ username }}
      password: "{{ password }}"
      dev_os: "{{ device_type }}"
  tags: ios-xe
  tasks:
    - napalm_install_config:
        provider: "{{ creds }}"
        config_file: "configs/{{ inventory_hostname }}"
        commit_changes: True
        replace_config: False
        register: napalm_diff
    - debug:
        msg: "{{ item }}"
        with_items: "{{ napalm_diff.msg.split('\n') }}"
- name: Deploy Cisco NX-OS Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: no
  vars:
    creds:
      hostname: "{{ ansible_host }}"
      username: {{ username }}
"deploy_configurations.yaml" 59 lines, 1566 characters written
```

Make sure you update the ASA group too!

One your Playbook one last time to make sure none of the improvements broke our functionality!

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
TASK [napalm_install_config] *****
changed: [acme-sea-nxos1]

TASK [debug] *****
ok: [acme-sea-nxos1] => (item=None) =>
  "msg": "username ***** password 0 ***** role network-admin"
}
ok: [acme-sea-nxos1] => (item=None) => {
  "msg": "ip route 0.0.0.0 0.0.0.0 10.255.255.1"
}
ok: [acme-sea-nxos1] => (item=None) => {
  "msg": " ip route 1.1.1.1 255.255.255.255 null0"
}
ok: [acme-sea-nxos1] => (item=None) => {
  "msg": " name to_acme_sea_asal"
}
ok: [acme-sea-nxos1] => (item=None) => {
  "msg": " interface vlan1000"
}
ok: [acme-sea-nxos1] => (item=None) => {
  "msg": " ip address 203.0.113.1 255.255.255.192"
}
ok: [acme-sea-nxos1] => (item=None) => {
  "msg": " no shutdown"
}
ok: [acme-sea-nxos1] => (item=None) => {
  "msg": " switchport mode trunk"
}
ok: [acme-sea-nxos1] => (item=None) => {
  "msg": " switchport trunk native vlan 1000"
}
ok: [acme-sea-nxos1] => (item=None) => {
  "msg": " no shutdown"
}
ok: [acme-sea-nxos1] => (item=None) => {
  "msg": "boot nxos bootflash:/nxos.7.0.3.I7.3.bin"
}

PLAY [Deploy Cisco ASA Configurations] *****
TASK [asa_config] *****
changed: [acme-sea-asal]

PLAY RECAP *****
acme-sea-asal      : ok=1    changed=1    unreachable=0    failed=0
acme-sea-nxos1     : ok=2    changed=1    unreachable=0    failed=0
acme-sea-rtr1      : ok=2    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$
```

There is still plenty of room for improvement within our Playbook (ideally making it idempotent, but that is a rather *serious* undertaking), but we now have a functional, programmatic way to not only generate configurations, but also to deploy them!

# Jenkins and IaC

## Overview and Objectives

In this lab you will create a Pipeline in Jenkins to orchestrate generation of configuration files based on your Ansible inventory/variables as well as actually deploying the configurations. You'll set the stage (literally stages!) for validation and verification of the deployment as well that we will continue to build on throughout the labs.

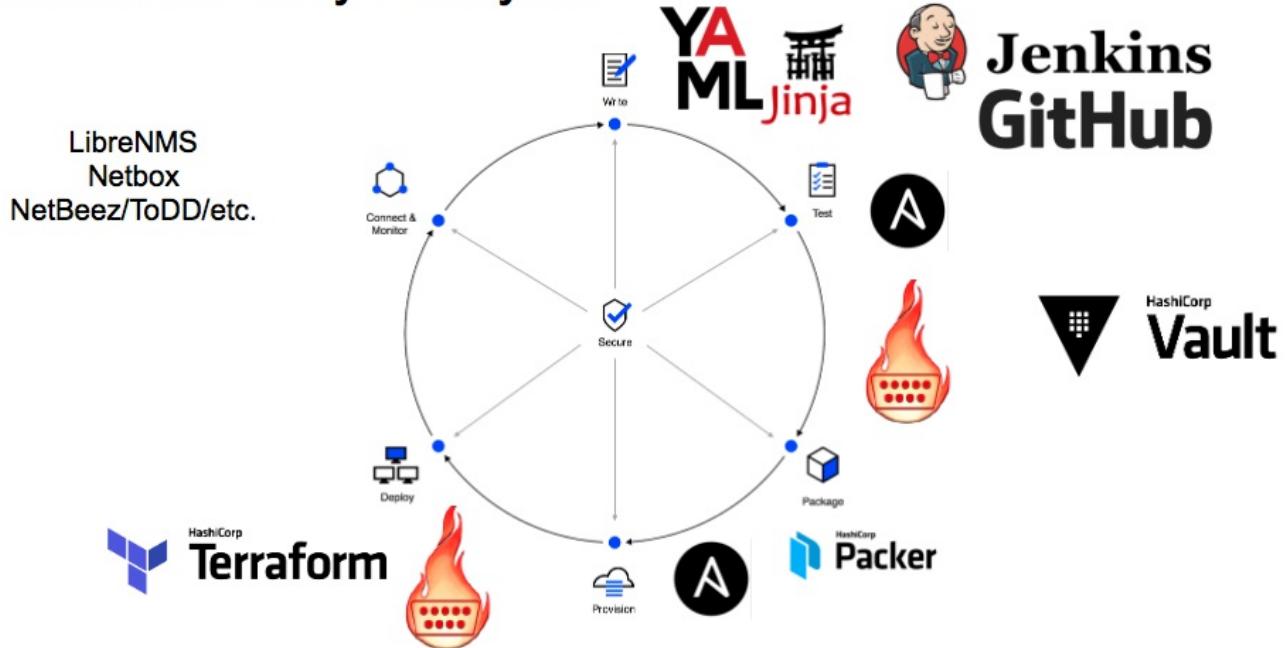
Objectives:

- Create a Jenkins job
- Create a Jenkinsfile to reflect the stages in our CI/CD pipeline
- Push configuration files into GitHub
- Run Jenkins builds to validate your configurations

## Network Delivery Lifecycle Overview

Looking back to the Network Delivery Lifecycle, we are zooming out a bit. We aren't necessarily looking at any of the phases individually, but instead we are looking at automating the phases we've completed thus far.

## Network Delivery Lifecycle



## Getting ready for Jenkins

In general, it would have been a good idea for us to be tracking all of the changes that we made in the previous lab in our version control system. Before we get any further ahead of ourselves, let's do that now.

The first thing we need to do is to let git know that we want to track basically all of our new files, we can do that nice and simply as follows:

```
cd ~/my_network_as_code  
git add *
```

This is all well and good, however, we also used to have our "acme-sea-XYZ" configurations in the root of this directory when we first got rolling with things. We will need to untrack those (since they don't exist in that location anymore).

```
git rm acme-sea-*
```

This is only going to stop tracking those files in the root directory -- our previous "git add \*" will cover those files in our "legacy\_configs" folder.

With that out of the way, let's commit our changes locally.

```
git commit -m "templatized and deployable configurations!"
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
rm 'acme-sea-asal'
rm 'acme-sea-nxos1'
rm 'acme-sea-rtr1'
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m "templatized and deployable configurations!"
[master 5482b18] templatized and deployable configurations!
Committer: ignw <ignw@ignw-jumphost.ignw.io>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

22 files changed, 588 insertions(+)
create mode 100644 ansible.cfg
create mode 100644 configs/acme-sea-asal
create mode 100644 configs/acme-sea-nxos1
create mode 100644 configs/acme-sea-rtr1
create mode 100644 deploy_configurations.yaml
create mode 100644 generate_configurations.yaml
create mode 100644 group_vars/all.yaml
create mode 100644 group_vars/cisco-ios-xe-routers.yaml
create mode 100644 group_vars/cisco-nxos.yaml
create mode 100644 group_vars/sea.yaml
create mode 100644 host_vars/acme-sea-asal.yaml
create mode 100644 host_vars/acme-sea-nxos1.yaml
create mode 100644 host_vars/acme-sea-rtr1.yaml
create mode 100644 inventory
rename acme-sea-asal => legacy_configs/acme-sea-asal (100%)
rename acme-sea-nxos1 => legacy_configs/acme-sea-nxos1 (100%)
rename acme-sea-rtr1 => legacy_configs/acme-sea-rtr1 (100%)
create mode 100644 templates/cisco_asa.j2
create mode 100644 templates/cisco_ios_xe_router.j2
create mode 100644 templates/cisco_nxos.j2
create mode 100644 templates/generic_l2_interface_configs.j2
create mode 100644 templates/generic_l3_interface_configs.j2
ignw@ignw-jumphost:~/my_network_as_code$
```

Great, git is now tracking all of the files locally, but we still need to push this up to our repository.

```
git push -u origin master
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

22 files changed, 588 insertions(+)
create mode 100644 ansible.cfg
create mode 100644 configs/acme-sea-asal
create mode 100644 configs/acme-sea-nxos1
create mode 100644 configs/acme-sea-rtr1
create mode 100644 deploy_configurations.yaml
create mode 100644 generate_configurations.yaml
create mode 100644 group_vars/all.yaml
create mode 100644 group_vars/cisco-ios-xe-routers.yaml
create mode 100644 group_vars/cisco-nxos.yaml
create mode 100644 group_vars/sea.yaml
create mode 100644 host_vars/acme-sea-asal.yaml
create mode 100644 host_vars/acme-sea-nxos1.yaml
create mode 100644 host_vars/acme-sea-rtr1.yaml
create mode 100644 inventory
rename acme-sea-asal => legacy_configs/acme-sea-asal (100%)
rename acme-sea-nxos1 => legacy_configs/acme-sea-nxos1 (100%)
rename acme-sea-rtr1 => legacy_configs/acme-sea-rtr1 (100%)
create mode 100644 templates/cisco_asa.j2
create mode 100644 templates/cisco_ios_xe_router.j2
create mode 100644 templates/cisco_nxos.j2
create mode 100644 templates/generic_l2_interface_configs.j2
create mode 100644 templates/generic_l3_interface_configs.j2
ignw@ignw-jumphost: ~/my_network_as_code$ git push -u origin master
Warning: Permanently added the RSA host key for IP address '192.30.255.113' to the list of known hosts.
Counting objects: 25, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (23/23), done.
Writing objects: 100% (25/25), 5.89 KiB | 1.96 MiB/s, done.
Total 25 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To github.com:carlniger/my_network_as_code
  908f1ae..5482b18  master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost: ~/my_network_as_code$
```

The screenshot shows a GitHub repository page for 'carlniger/my\_network\_as\_code'. The repository is private and has 2 commits, 1 branch, 0 releases, and 0 contributors. The latest commit was made 4 minutes ago. The repository contains files like 'configs', 'group\_vars', 'host\_vars', 'legacy\_configs', 'templates', 'ansible.cfg', 'deploy\_configurations.yaml', 'generate\_configurations.yaml', and 'inventory', all of which are templatized and deployable configurations. There is a note to add a README file.

No description, website, or topics provided.

Add a README

① Firefox automatically sends some data to Mozilla so that we can improve your experience. Choose What I Share X

In general, it would probably be a good idea to add a README.md file to explain what the project is and how it works. For now, let's simply add a README.md with some silly sentence just so that we have one.

```
touch README.md
echo "automate all the things" >> README.md
git add README.md
git commit -m "whoops, forgot a README"
git push -u origin master
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ touch README.md
ignw@ignw-jumphost:~/my_network_as_code$ echo "automate all the things" >> README.md
ignw@ignw-jumphost:~/my_network_as_code$ git add README.md
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m "whoops, forgot a README"
[master b5fca8c] whoops, forgot a README
Committer: ignw <ignw@ignw-jumphost.ignw.io>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 README.md
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 305 bytes | 305.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:carlniger/my_network_as_code
  45ae1ef..b5fca8c  master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_network_as_code$
```

## Enter Jenkins (again, but for real this time!)

Great, now we have an awesome platform to work from. What next? We need to get this code turned into a Pipeline. Jenkins will allow us to automatically regenerate and redeploy configurations every time a commit is made to our repository. It will also give us a platform to write tests -- both unit tests (testing for things like syntax, valid IP addresses, etc.), and functional tests (pinging things, telnet-ing on ports to confirm/deny reachability, etc.). The results of the tests can then be used to determine if the "build" passes and should be deployed to production. Let's not get too far ahead of ourselves just yet though!

We need to create a Jenkinsfile just like we did before so that we can tell Jenkins what our stages are, and what to do in each of them. Create a new Jenkinsfile and open it for editing.

```
touch Jenkinsfile
```

Let's stub out the general idea of what we want the Pipeline to look like. We can create our stages and simply

leave a comment (in a Jenkinsfile this is a double slash "://" as a placeholder for each stage for now.

```
node {  
    stage ('Checkout Repository') {  
        // Get our repo cloned and prepped for action  
    }  
  
    stage ('Render Configurations') {  
        // Generate our configurations with our sweet Playbooks  
    }  
  
    stage ('Unit Testing') {  
        // Do some kind of "linting" on our code to make sure we didn't bugger anything up too badly  
    }  
  
    stage ('Deploy Configurations to Dev') {  
        // Push the configurations out to the dev environment  
    }  
  
    stage ('Functional/Integration Testing') {  
        // Ping stuff and make sure we didn't blow up dev!  
    }  
  
    stage ('Promote Configurations to Production') {  
        // Ping stuff and make sure we didn't blow up dev!  
    }  
  
    stage ('Production Functional/Integration Testing') {  
        // Ping stuff and make sure we didn't blow up prod!  
    }  
}
```

Track the Jenkinsfile in Git, and commit it to your repository:

```
git add Jenkinsfile  
git commit -m 'add jenkinsfile'  
git push -u origin master
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ls
ansible.cfg deploy_configurations.yaml group_vars inventory README.md
configs generate_configurations.yaml host_vars legacy_configs templates
ignw@ignw-jumphost:~/my_network_as_code$ touch Jenkinsfile
ignw@ignw-jumphost:~/my_network_as_code$ vi Jenkinsfile
ignw@ignw-jumphost:~/my_network_as_code$ git add Jenkinsfile
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m 'add jenkinsfile'
[master 1070617] add jenkinsfile
Committer: ignw <ignw@ignw-jumphost.ignw.io>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 30 insertions(+)
create mode 100644 Jenkinsfile
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 572 bytes | 572.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:carlniger/my_network_as_code
  f4d1267..1070617 master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_network_as_code$
```

Now that we've got at least a stubbed out Jenkinsfile, we probably should get a Pipeline created in Jenkins for this repository. Head over to your Jenkins server (10.10.0.253:8080 from the RDP Server), login with the credentials you created, click on the "New Item" button to create a new Pipeline.

Dashboard [Jenkins] - Mozilla Firefox

Dashboard [Jenkins] + 10.10.0.253:8080 ... ? log out ENABLE AUTO REFRESH

# Jenkins

New Item

People

Build History

Project Relationship

Check File Fingerprint

Manage Jenkins

My Views

Credentials

New View

**Build Queue**

No builds in the queue.

**Build Executor Status**

1 Idle  
2 Idle

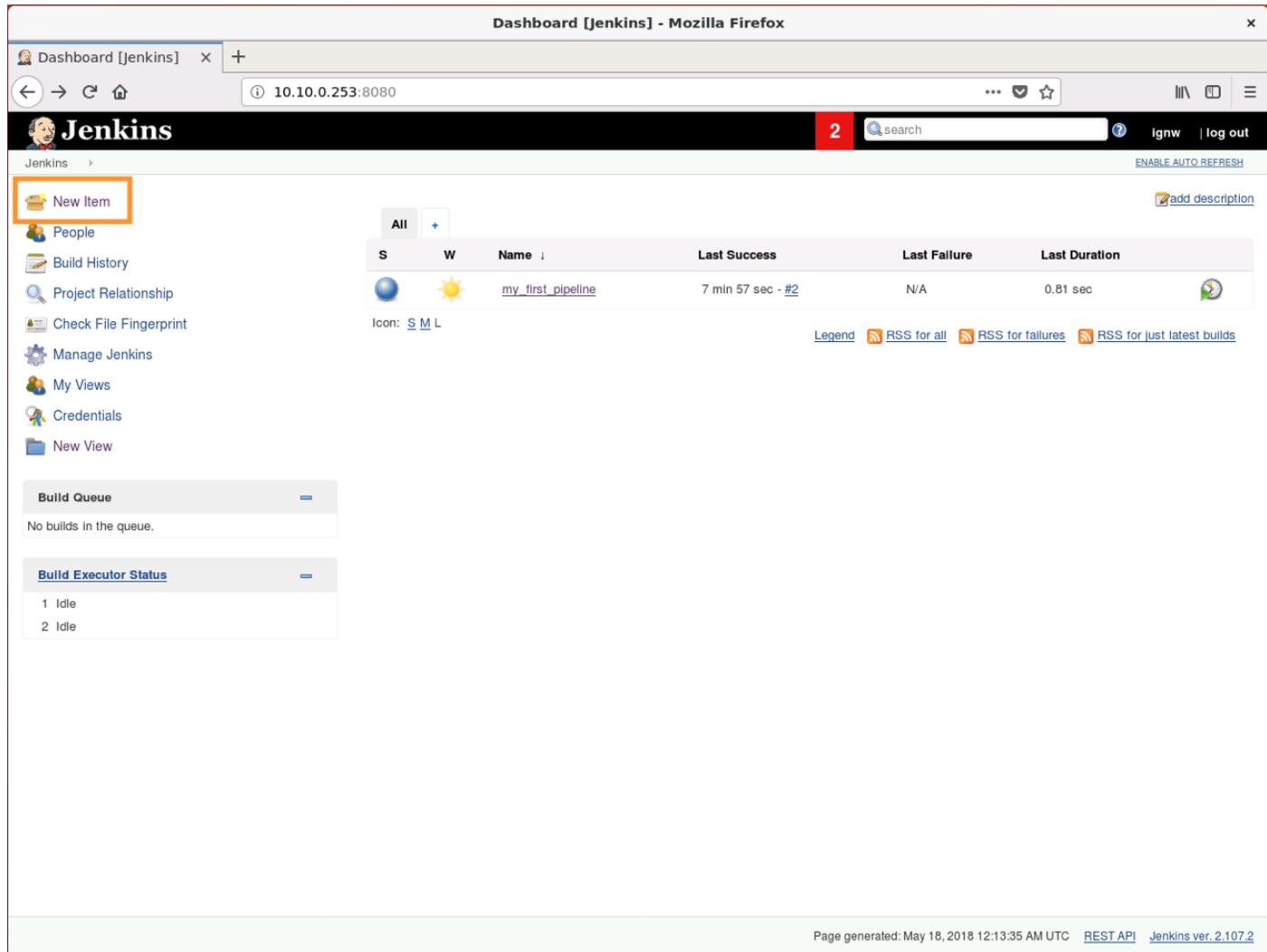
2 search add description

All +

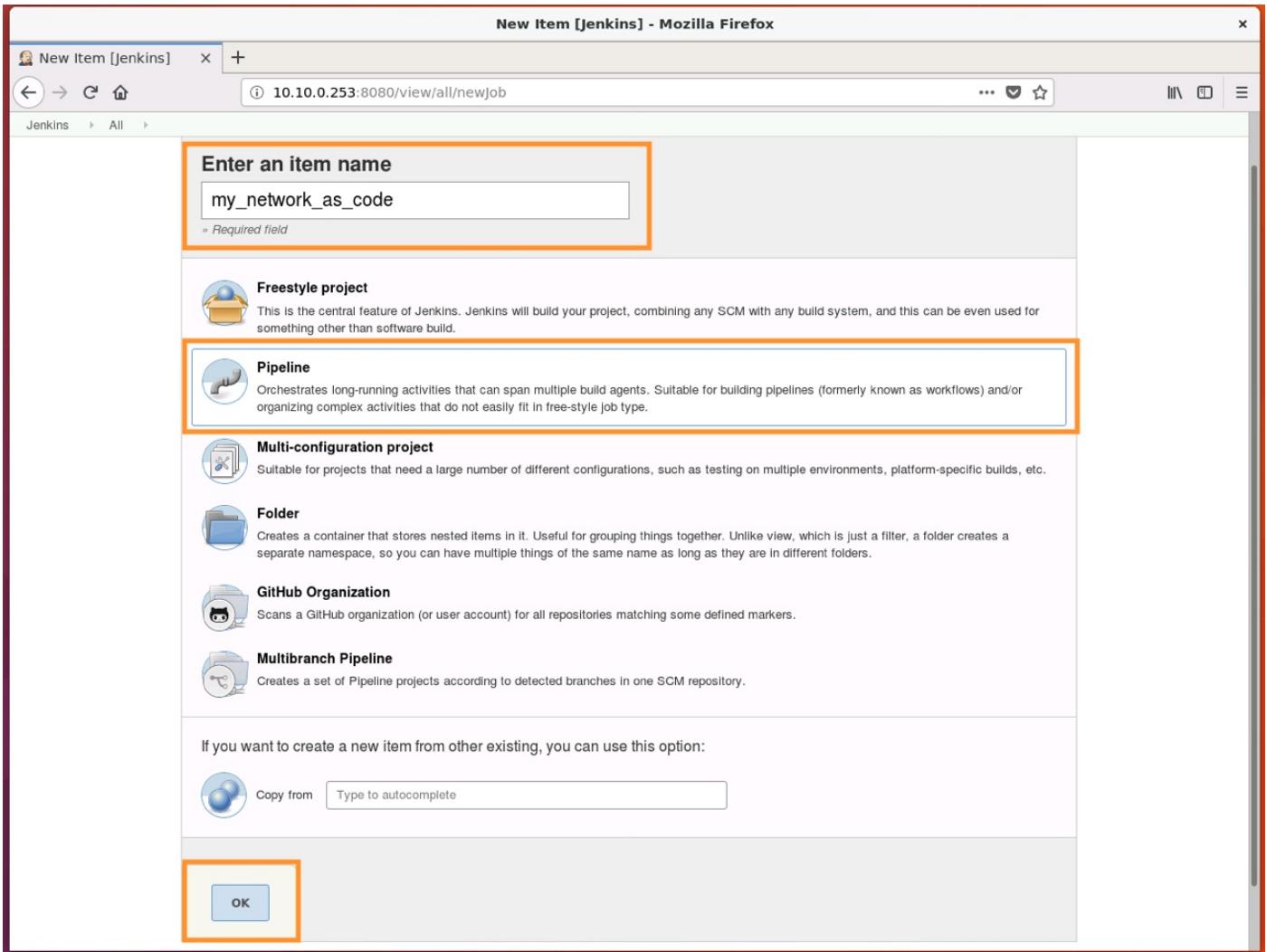
| S | W | Name ↓            | Last Success      | Last Failure | Last Duration |
|---|---|-------------------|-------------------|--------------|---------------|
| ● | ○ | my_first_pipeline | 7 min 57 sec - #2 | N/A          | 0.81 sec      |

Icon: S M L Legend RSS for all RSS for failures RSS for just latest builds

Page generated: May 18, 2018 12:13:35 AM UTC REST API Jenkins ver. 2.107.2



Once again, we'll select the "Pipeline" option. Enter the name for your new Pipeline, and then click "OK".



Enter the URL for your Github Project (again, not strictly required, but a good idea), and then scroll down to the Pipeline section.

Ensure "Pipeline script from SCM" is selected from the drop down, also select "Git" from the SCM drop down. Enter your repository URL and select the credentials you created previously from the drop down. Finally, ensure the "Script Path" is "Jenkinsfile" so it finds our newly added Jenkinsfile.

my\_network\_as\_code Config [Jenkins] - Mozilla Firefox

my\_network\_as\_code +  
10.10.0.253:8080/job/my\_network\_as\_code/configure

Jenkins my\_network\_as\_code > Pipeline

General Build Triggers Advanced Project Options Pipeline

**Pipeline**

Definition Pipeline script from SCM  
SCM Git

Repositories Repository URL `github.com/carlrieger/my_network_as_code`  
Credentials `carlrieger***** (carl)` Add Advanced...  
Add Repository

Branches to build Branch Specifier (blank for 'any') `*/master` Add Branch

Repository browser (Auto)

Additional Behaviours Add

Script Path Jenkinsfile  
Lightweight checkout

[Pipeline Syntax](#)

Save Apply

The screenshot shows the Jenkins Pipeline configuration interface. The 'Pipeline' tab is active. In the 'Definition' section, 'Pipeline script from SCM' is selected, with 'SCM' set to 'Git'. The 'Repositories' section contains one repository with the URL 'github.com/carlrieger/my\_network\_as\_code' and credentials 'carlrieger\*\*\*\*\* (carl)'. The 'Branches to build' section specifies a branch specifier of '\*/master'. The 'Script Path' is set to 'Jenkinsfile'. The 'Lightweight checkout' option is checked. At the bottom, there are 'Save' and 'Apply' buttons.

Once done, click "Save". You'll be dropped off at the "home page" for your new workflow.

my\_network\_as\_code [Jenkins] - Mozilla Firefox

10.10.0.253:8080/job/my\_network\_as\_code/

# Jenkins

Jenkins my\_network\_as\_code

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Build Now](#)

[Delete Pipeline](#)

[Configure](#)

[Full Stage View](#)

[GitHub](#)

[Pipeline Syntax](#)

## Pipeline my\_network\_as\_code

[Recent Changes](#)

### Stage View

No data available. This Pipeline has not yet run.

### Build History

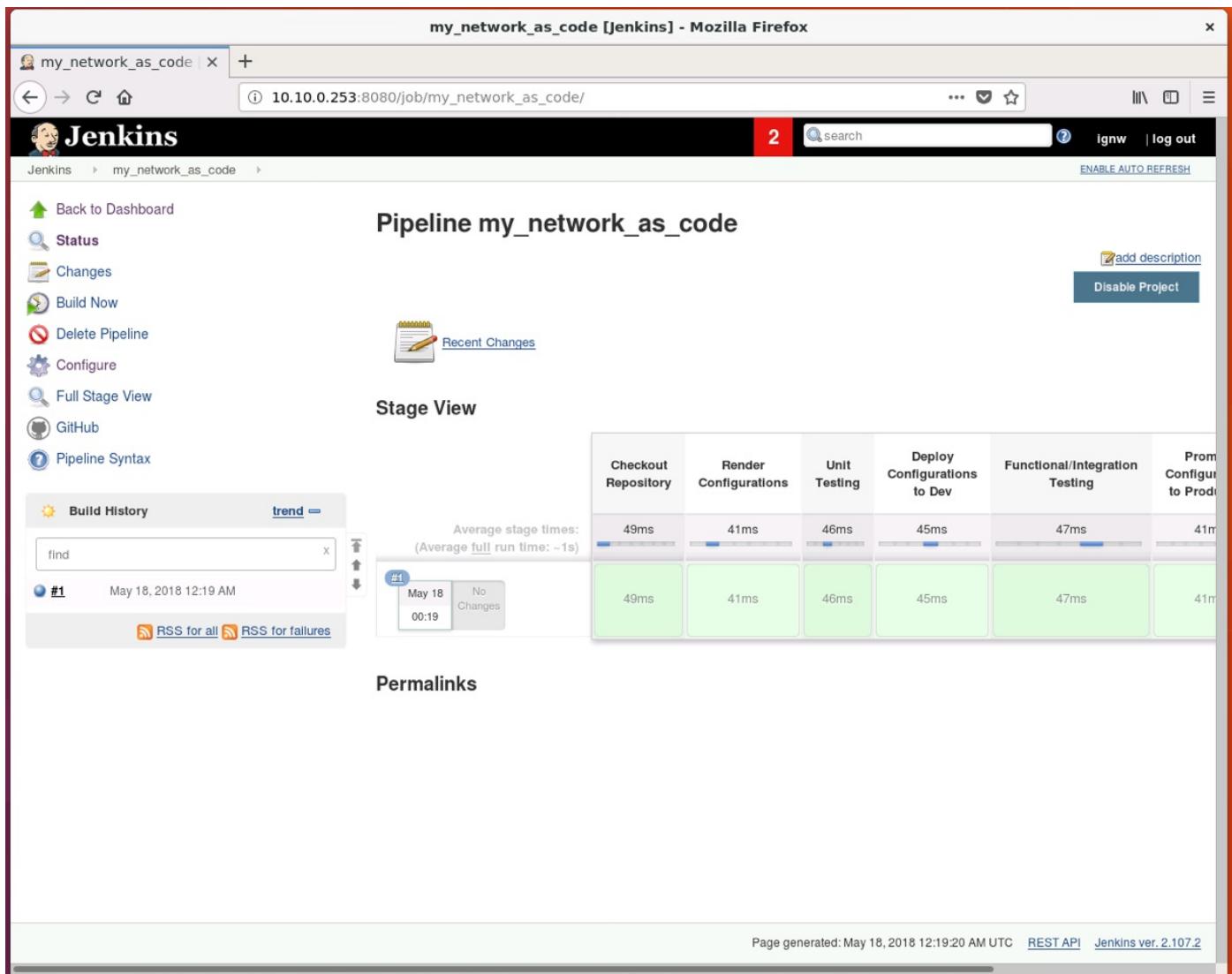
trend →

find

RSS for all RSS for failures

Page generated: May 18, 2018 12:19:20 AM UTC REST API Jenkins ver. 2.107.2

Click "Build Now" to make sure everything is looking good so far. If all goes according to plan your Pipeline should show all of the stages we created, even though they aren't doing anything just yet!



## Jenkins Build Environment

Now we need to think a little bit about how we want to have Jenkins interact with our code. We know that it will need a copy of our code -- since we built it all on the RDP server and not on the Jenkins box itself. Since we've stored all our code in GitHub, we can simply clone the repository and have all of the same stuff that we've been working with ready to go on the Jenkins server.

Unsurprisingly with all the GitHub integration Jenkins has, cloning our repository is a *very simple task*, and can be accomplished with only one line of code! One more item of housekeeping that we want to take care of -- Jenkins will create a folder for each of our Pipelines, and store the code/artifacts in there while it is doing what we ask it to do. A good idea is to have Jenkins start with a clean slate for each build it executes, we can also achieve that with a one liner.

Add the following to your Jenkinsfile:

```
node {  
    stage ('Checkout Repository') {  
        deleteDir()  
        checkout scm  
    }  
}
```

As you may expect, the "deleteDir()" function is doing exactly as it sounds. The "checkout scm" is also doing what you would expect (remember SCM = Source Control Manager).

This is a good time to note that we could have Jenkins spin up Docker containers in which we could generate our configurations and use that container as a platform to deploy our configs out, however we want to keep it as simple as possible for now.

Push your updated code to Github, then execute your build in Jenkins once again.

```
git add Jenkinsfile  
git commit -m 'updated stage 1'  
git push -u origin master
```

my\_network\_as\_code [Jenkins] - Mozilla Firefox

my\_network\_as\_code | +  
10.10.0.253:8080/job/my\_network\_as\_code/

Jenkins Pipeline my\_network\_as\_code

Back to Dashboard Status Changes Build Now Delete Pipeline Configure Full Stage View GitHub Pipeline Syntax

Recent Changes

Add description Disable Project

Build History trend → find #2 May 18, 2018 1:00 AM #1 May 18, 2018 12:19 AM RSS for all RSS for failures

Average stage times:  
(Average full run time: ~1s)

|                       | Checkout Repository | Render Configurations | Unit Testing | Deploy Configurations to Dev | Functional/Integration Testing | Promote Configurations to Prod |
|-----------------------|---------------------|-----------------------|--------------|------------------------------|--------------------------------|--------------------------------|
| #2<br>May 18<br>01:00 | 927ms               | 40ms                  | 43ms         | 42ms                         | 45ms                           | 42ms                           |
| #1<br>May 18<br>00:19 | 49ms                | 41ms                  | 46ms         | 45ms                         | 47ms                           | 41ms                           |

Permalinks

Page generated: May 18, 2018 12:19:20 AM UTC REST API Jenkins ver. 2.107.2

The screenshot shows the Jenkins Pipeline interface for the 'my\_network\_as\_code' project. On the left, there's a sidebar with links like Back to Dashboard, Status, Changes, Build Now, Delete Pipeline, Configure, Full Stage View, GitHub, and Pipeline Syntax. Below that is the Build History section, which lists two builds: #2 (May 18, 2018 1:00 AM) and #1 (May 18, 2018 12:19 AM). The RSS feed links are also present. The main content area is titled 'Pipeline my\_network\_as\_code' and features a 'Stage View' grid. The grid has columns for Checkout Repository, Render Configurations, Unit Testing, Deploy Configurations to Dev, Functional/Integration Testing, and Promote Configurations to Prod. Each row represents a build, and each column represents a stage. The 'Checkout Repository' stage for build #2 is highlighted with a red box and shows a run time of 927ms. The 'Checkout Repository' stage for build #1 shows a run time of 49ms. Other stages in the grid show run times ranging from 40ms to 44ms.

It sure looks like our "Checkout Repository" stage took a lot longer this time (927ms vs 49ms) -- that is probably a really good sign that Jenkins did in fact do *something!*

Click on your build in the Build History section so we can get some more details:

my\_network\_as\_code [Jenkins] - Mozilla Firefox

my\_network\_as\_code | +  
10.10.0.253:8080/job/my\_network\_as\_code/

Jenkins my\_network\_as\_code log out  
Back to Dashboard Status Changes Build Now Delete Pipeline Configure Full Stage View GitHub Pipeline Syntax

Pipeline my\_network\_as\_code

Recent Changes

Stage View

Average stage times:  
(Average full run time: ~1s)

| Checkout Repository | Render Configurations | Unit Testing | Deploy Configurations to Dev | Functional/Integration Testing | Promote Configurations to Prod |
|---------------------|-----------------------|--------------|------------------------------|--------------------------------|--------------------------------|
| 488ms               | 40ms                  | 43ms         | 42ms                         | 45ms                           | 42ms                           |
| 927ms               | 40ms                  | 40ms         | 39ms                         | 43ms                           | 44ms                           |
| 49ms                | 41ms                  | 46ms         | 45ms                         | 47ms                           | 41ms                           |

Build History

#2 May 18, 2018 1:00 AM  
#1 May 18, 2018 12:19 AM

RSS for all RSS for failures

Permalinks

- Last build (#2), 1 min 35 sec ago
- Last stable build (#2), 1 min 35 sec ago
- Last successful build (#2), 1 min 35 sec ago
- Last completed build (#2), 1 min 35 sec ago

Page generated: May 18, 2018 1:02:11 AM UTC REST API Jenkins ver. 2.107.2

Click on "Console Output"...

my\_network\_as\_code #2 [Jenkins] - Mozilla Firefox

my\_network\_as\_code + 10.10.0.253:8080/job/my\_network\_as\_code/2/ ... search ignw | log out

Jenkins my\_network\_as\_code #2 ENABLE AUTO REFRESH

Back to Project Status Changes Console Output Edit Build Information Delete Build Git Build Data No Tags Replay Pipeline Steps Previous Build

Build #2 (May 18, 2018 1:00:35 AM)

Started by user ignw  
Revision: 96f710c60254dfe3d93e5a7fb1338c6266d3fa8b  
refs/remotes/origin/master

add description

Started 2 min 18 sec ago Took 2 sec

Page generated: May 18, 2018 1:02:54 AM UTC REST API Jenkins ver. 2.107.2

We can see that Jenkins does indeed clone our repository!

my\_network\_as\_code #2 Console [Jenkins] - Mozilla Firefox

my\_network\_as\_code 10.10.0.253:8080/job/my\_network\_as\_code/2/console

Jenkins 2 search ignw log out

Console Output

Started by user ignw  
Obtained Jenkinsfile from git [https://github.com/carligner/my\\_network\\_as\\_code](https://github.com/carligner/my_network_as_code)  
Running in Durability level: MAX SURVIVABILITY  
[Pipeline] node  
Running on Jenkins in /var/lib/jenkins/jobs/my\_network\_as\_code/workspace  
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (Checkout Repository)  
[Pipeline] deleteDir  
[Pipeline] checkout  
Cloning the remote Git repository  
Cloning repository [https://github.com/carligner/my\\_network\\_as\\_code](https://github.com/carligner/my_network_as_code)  
> git init /var/lib/jenkins/jobs/my\_network\_as\_code/workspace # timeout=10  
Fetching upstream changes from [https://github.com/carligner/my\\_network\\_as\\_code](https://github.com/carligner/my_network_as_code)  
> git --version # timeout=10  
using GIT\_ASKPASS to set credentials carl  
> git fetch --tags --progress [https://github.com/carligner/my\\_network\\_as\\_code](https://github.com/carligner/my_network_as_code) +refs/heads/\*:refs/remotes/origin/\*  
> git config remote.origin.url [https://github.com/carligner/my\\_network\\_as\\_code](https://github.com/carligner/my_network_as_code) # timeout=10  
> git config --add remote.origin.fetch +refs/heads/\*:refs/remotes/origin/\* # timeout=10  
> git config remote.origin.url [https://github.com/carligner/my\\_network\\_as\\_code](https://github.com/carligner/my_network_as_code) # timeout=10  
Fetching upstream changes from [https://github.com/carligner/my\\_network\\_as\\_code](https://github.com/carligner/my_network_as_code)  
using GIT\_ASKPASS to set credentials carl  
> git fetch --tags --progress [https://github.com/carligner/my\\_network\\_as\\_code](https://github.com/carligner/my_network_as_code) +refs/heads/\*:refs/remotes/origin/\*  
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10  
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10  
Checking out Revision 96ff0c60254dfe3d93e5a7fb1338c6266d3fa8b (refs/remotes/origin/master)  
> git config core.sparsecheckout # timeout=10  
> git checkout -f 96ff0c60254dfe3d93e5a7fb1338c6266d3fa8b  
Commit message: "updated stage 1"  
First time build. Skipping changelog.  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] stage  
[Pipeline] { (Render Configurations)  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] stage  
[Pipeline] }

The next question is where is this all happening? SSH to your Jenkins server, and navigate to "/var/lib/jenkins/jobs"

```
ignw@ignw-jenkins: /var/lib/jenkins/jobs
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ssh 10.10.0.253
ignw@10.10.0.253's password:
Welcome to Ubuntu 17.10 (GNU/Linux 4.13.0-41-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:      https://landscape.canonical.com
 * Support:         https://ubuntu.com/advantage

New release '18.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

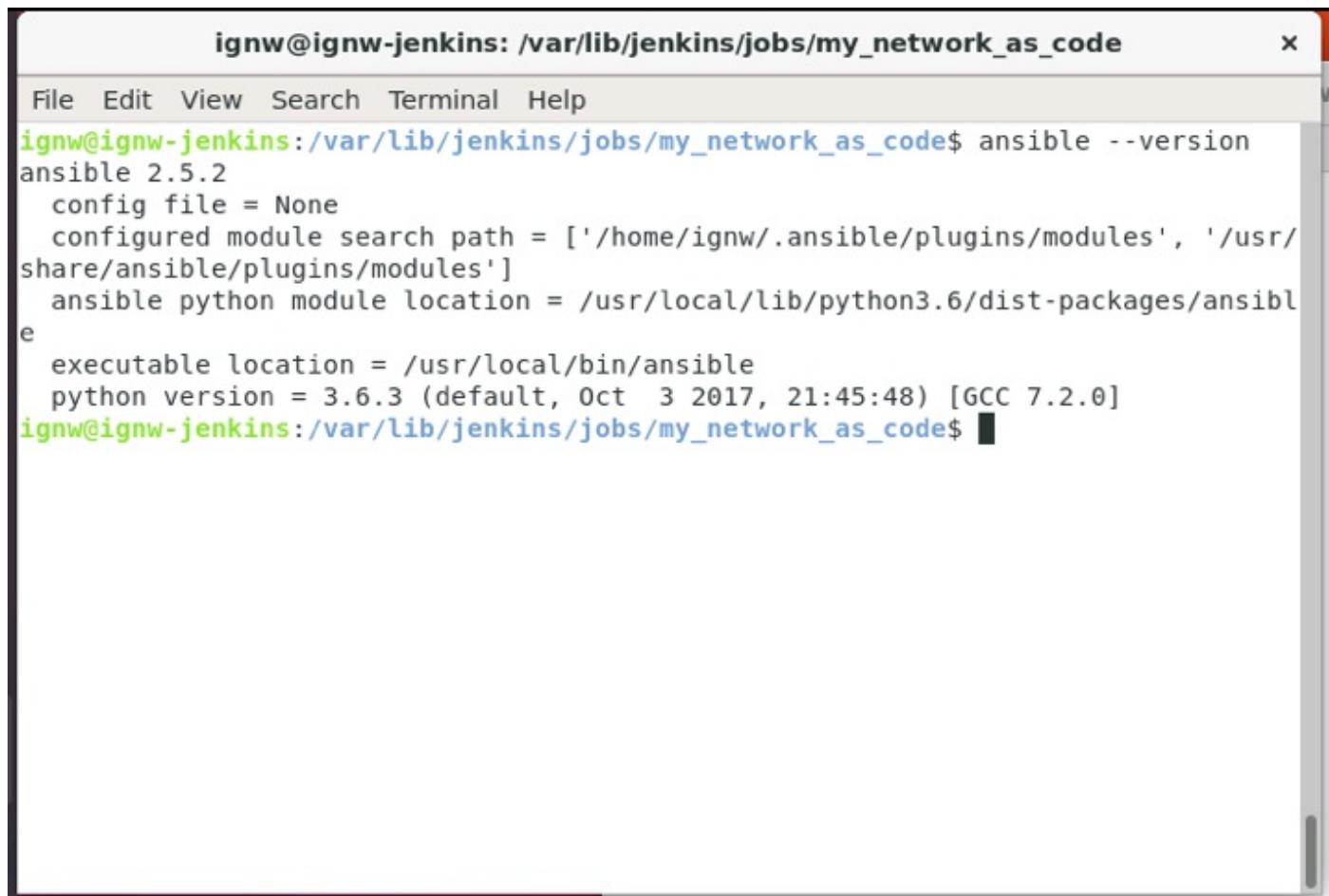
Last login: Fri May 18 01:05:05 2018 from 10.10.0.254
ignw@ignw-jenkins: ~$ cd /var/lib/jenkins/jobs/
ignw@ignw-jenkins:/var/lib/jenkins/jobs$ ls
my_first_pipeline  my_network_as_code
ignw@ignw-jenkins:/var/lib/jenkins/jobs$
```

Sure enough, both of our Pipelines are there! Within the Pipeline there is a workspace directory where our repository was cloned to:

```
ignw@ignw-jenkins: /var/lib/jenkins/jobs/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jenkins:/var/lib/jenkins/jobs$ cd my_network_as_code/
ignw@ignw-jenkins:/var/lib/jenkins/jobs/my_network_as_code$ ls
builds      lastStable      nextBuildNumber  workspace@tmp
config.xml   lastSuccessful  workspace
ignw@ignw-jenkins:/var/lib/jenkins/jobs/my_network_as_code$ ls workspace
ansible.cfg           group_vars  legacy_configs
configs               host_vars   README.md
deploy_configurations.yaml inventory templates
generate_configurations.yaml Jenkinsfile
ignw@ignw-jenkins:/var/lib/jenkins/jobs/my_network_as_code$ █
```

Great, now we know where things are happening, and better yet, things *are* happening! Our next step is to actually generate our configuration files in the same way we did on the RDP host.

Again, in production, you would likely be doing this on a container, and that container would be spun up and have your requirements installed onto it -- in this case, we need to make sure that our requirements are simply available on the Jenkins server. The only thing we need to generate our configurations is Ansible, this has been installed on the server for you already.



A screenshot of a terminal window titled "ignw@ignw-jenkins: /var/lib/jenkins/jobs/my\_network\_as\_code". The window shows the output of the "ansible --version" command. The output includes details about the config file (None), module search path ('/home/ignw/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules'), python module location ('/usr/local/lib/python3.6/dist-packages/ansible'), executable location ('/usr/local/bin/ansible'), and python version (3.6.3). The terminal window has a standard OS X-style interface with a menu bar at the top.

```
ignw@ignw-jenkins: /var/lib/jenkins/jobs/my_network_as_code$ ansible --version
ansible 2.5.2
  config file = None
  configured module search path = ['/home/ignw/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
    ansible python module location = /usr/local/lib/python3.6/dist-packages/ansible
  executable location = /usr/local/bin/ansible
  python version = 3.6.3 (default, Oct  3 2017, 21:45:48) [GCC 7.2.0]
ignw@ignw-jenkins: /var/lib/jenkins/jobs/my_network_as_code$
```

## Running the Playbooks

On our RDP host we simply ran "ansible-playbook generate\_configurations.yaml" -- it would stand to reason that we can do the same thing here. In an earlier section we wrote a Jenkinsfile to simply "echo" silly things to the console, this was just using shell. We can similarly use shell to execute our Playbook.

Replace the line "// Generate our configurations with our sweet Playbooks" with the following:

```
sh 'ansible-playbook generate_configurations.yaml'
```

Note that Jenkins will already be setting the working directory to the directory of the workspace that we investigated previously, so we don't need to worry about any of that!

Push this up to the GitHub repository so we can re-run our Pipeline:

```
git add -u
git commit -m 'updated jenkinsfile'
git push -u origin master
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ls
ansible.cfg deploy_configurations.yaml group_vars inventory legacy_configs templates
configs generate_configurations.yaml host_vars Jenkinsfile README.md
ignw@ignw-jumphost:~/my_network_as_code$ vi Jenkinsfile
ignw@ignw-jumphost:~/my_network_as_code$ git add -u
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m 'updated jenkinsfile'
[master f546502] updated jenkinsfile
Committer: ignw <ignw@ignw-jumphost.ignw.io>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 1 insertion(+), 1 deletion(-)
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 362 bytes | 362.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
```

Note that the "git add -u" will update already tracked files, so this is just a bit easier of a way to do the same thing we've been doing previously.

Once more, run your build from Jenkins.

my\_network\_as\_code [Jenkins] - Mozilla Firefox

my\_network\_as\_code | + 10.10.0.253:8080/job/my\_network\_as\_code/ 2 search Ignw | log out ENABLE AUTO REFRESH

Jenkins my\_network\_as\_code Pipeline my\_network\_as\_code Add description Disable Project

Back to Dashboard Status Changes Build Now Delete Pipeline Configure Full Stage View GitHub Pipeline Syntax

Recent Changes

Stage View

| Checkout Repository | Render Configurations | Unit Testing | Deploy Configurations to Dev | Functional/Integration Testing | Promote Configurations to Prod |
|---------------------|-----------------------|--------------|------------------------------|--------------------------------|--------------------------------|
| 734ms               | 910ms                 | 41ms         | 40ms                         | 42ms                           | 42ms                           |
| 966ms               | 3s                    | 37ms         | 40ms                         | 36ms                           | 36ms                           |
| 996ms               | 46ms                  | 42ms         | 38ms                         | 42ms                           | 48ms                           |
| 927ms               | 40ms                  | 40ms         | 39ms                         | 43ms                           | 44ms                           |
| 49ms                | 41ms                  | 46ms         | 45ms                         | 47ms                           | 41ms                           |

Build History trend →

- #4 May 18, 2018 1:48 PM
- #3 May 18, 2018 1:11 AM
- #2 May 18, 2018 1:00 AM
- #1 May 18, 2018 12:19 AM

RSS for all RSS for failures

Permalinks

- Last build (#2), 10 min ago

| Stage                          | Time  |
|--------------------------------|-------|
| Checkout Repository            | 734ms |
| Render Configurations          | 910ms |
| Unit Testing                   | 41ms  |
| Deploy Configurations to Dev   | 40ms  |
| Functional/Integration Testing | 42ms  |
| Promote Configurations to Prod | 42ms  |

| Build | Date   | Time  | Commits | Changes |
|-------|--------|-------|---------|---------|
| #4    | May 18 | 13:48 | 1       | commit  |
| #3    | May 18 | 01:11 | No      | Changes |
| #2    | May 18 | 01:00 | No      | Changes |
| #1    | May 18 | 00:19 | No      | Changes |

Looks like our second stage (Render Configurations) is now taking longer than 40ms, so it is probably doing something now! Pop over to the build details, and check out the console output:

my\_network\_as\_code #4 Console [Jenkins] - Mozilla Firefox

my\_network\_as\_code x +  
 Jenkins > my\_network\_as\_code > #4  
 10.10.0.253:8080/job/my\_network\_as\_code/4/console

```

    - git config --global https://github.com/carlniger/my_network_as_code url "https://$GITHUB_TOKEN@github.com/carlniger/my_network_as_code"
    Fetching upstream changes from https://github.com/carlniger/my_network_as_code
    using GIT_ASKPASS to set credentials carl
    > git fetch --tags --progress https://github.com/carlniger/my_network_as_code +refs/heads/*:refs/remotes/origin/*
    > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
    > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
    Checking out Revision f546502984ff39c9dacb5143e768d390efccb3a8 (refs/remotes/origin/master)
    > git config core.sparsecheckout # timeout=10
    > git checkout -f f546502984ff39c9dacb5143e768d390efccb3a8
    Commit message: "updated jenkinsfile"
    > git rev-list --no-walk 96f7fc60254dfe3d93e5a7fb1338c6266d3fa8b # timeout=10
  [Pipeline]
  [Pipeline] // stage
  [Pipeline] stage
  [Pipeline] { (Render Configurations)
  [Pipeline] sh
  [workspace] Running shell script
+ ansible-playbook generate_configurations.yaml

PLAY [Generate IOS-XE Router Configurations] ****
TASK [template] ****
ok: [acme-sea-rt1]

PLAY [Generate NX-OS Router Configurations] ****
TASK [template] ****
ok: [acme-sea-nxos1]

PLAY [Generate ASA Configurations] ****
TASK [template] ****
ok: [acme-sea-asal]

PLAY RECAP ****
acme-sea-asal      : ok=1    changed=0    unreachable=0    failed=0
acme-sea-nxos1     : ok=1    changed=0    unreachable=0    failed=0
acme-sea-rt1        : ok=1    changed=0    unreachable=0    failed=0

  [Pipeline]
  [Pipeline] // stage
  [Pipeline] stage
  [Pipeline] { (Unit Testing)
  [Pipeline]
  [Pipeline] // stage
  [Pipeline] stage
  [Pipeline] { (Deploy Configurations to Dev)

```

Very good, Ansible did in fact execute our Playbook, and the results were "OK" for each of our devices!

Up to this point we've been having a need to store our configurations locally so that we could inspect them and make sure we got our templating dialed in correctly, but now our builds are happening on Jenkins as soon as we hit "build now", so we really have no need to have our configs actually stored in the repository. Let's remove them from our repository on the RDP host, and commit those changes:

```

rm configs/*
git rm configs/*
git commit -m 'removing configurations from repository'
git push -u origin master

```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ 
ignw@ignw-jumphost:~/my_network_as_code$ 
ignw@ignw-jumphost:~/my_network_as_code$ rm configs/*
ignw@ignw-jumphost:~/my_network_as_code$ git rm configs/*
rm 'configs/acme-sea-asal'
rm 'configs/acme-sea-nxos1'
rm 'configs/acme-sea-rtr1'
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m 'removing configurations from repository'
[master e605dc8] removing configurations from repository
Committer: ignw <ignw@ignw-jumphost.ignw.io>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

3 files changed, 145 deletions(-)
delete mode 100644 configs/acme-sea-asal
delete mode 100644 configs/acme-sea-nxos1
delete mode 100644 configs/acme-sea-rtr1
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master
Counting objects: 2, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 242 bytes | 242.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:carlniger/my_network_as_code
 f546502..e605dc8  master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_network_as_code$
```

Do you think your build will run successfully now? Give it a shot:

my\_network\_as\_code [Jenkins] - Mozilla Firefox

carlniger/my\_network | + 10.10.0.253:8080/job/my\_network\_as\_code/ ... search log in | log out

# Jenkins

Back to Dashboard Status Changes Build Now Delete Pipeline Configure Full Stage View GitHub Pipeline Syntax

## Pipeline my\_network\_as\_code

[Recent Changes](#)

[add description](#) [Disable Project](#)

### Stage View

Average stage times: (Average full run time: ~2s)

| Checkout Repository | Render Configurations | Unit Testing | Deploy Configurations to Dev | Functional/Integration Testing | Promote Configurations to Prod |
|---------------------|-----------------------|--------------|------------------------------|--------------------------------|--------------------------------|
| 718ms               | 1s                    | 41ms         | 40ms                         | 42ms                           | 42ms                           |
| 652ms               | <b>2s</b> failed      |              |                              |                                |                                |
| 966ms               | 3s                    | 37ms         | 40ms                         | 36ms                           | 36ms                           |
| 996ms               | 46ms                  | 42ms         | 38ms                         | 42ms                           | 48ms                           |
| 927ms               | 40ms                  | 40ms         | 39ms                         | 43ms                           | 44ms                           |
| 49ms                | 41ms                  | 46ms         | 45ms                         | 47ms                           | 41ms                           |

**#5 May 18, 2018 1:57 PM** 1 commit

**#4 May 18, 2018 1:48 PM** 1 commit

**#3 May 18, 2018 1:11 AM** No Changes

**#2 May 18, 2018 1:00 AM** No Changes

**#1 May 18, 2018 12:19 AM** No Changes

RSS for all RSS for failures

Well... two steps forward, one step back! Why do you think it failed? Taking a look at the console output will shed some light on that:

my\_network\_as\_code #5 Console [Jenkins] - Mozilla Firefox

my\_network\_as\_code | carlniger/my\_network\_ | +

10.10.0.253:8080/job/my\_network\_as\_code/5/console

```

Jenkins > my_network_as_code > #5
  Pipeline Steps
    Previous Build

receiving upstream changes from https://github.com/carlniger/my_network_as_code
> git --version # timeout=10
using GIT_ASKPASS to set credentials carl
> git fetch --tags --progress https://github.com/carlniger/my_network_as_code +refs/heads/*:refs/remotes/origin/*
> git config remote.origin.url https://github.com/carlniger/my_network_as_code # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/carlniger/my_network_as_code # timeout=10
Fetching upstream changes from https://github.com/carlniger/my_network_as_code
using GIT_ASKPASS to set credentials carl
> git fetch --tags --progress https://github.com/carlniger/my_network_as_code +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision e605dc87296a0229e51f9f64c417b25357e9c273 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f e605dc87296a0229e51f9f64c417b25357e9c273
Commit message: "removing configurations from repository"
> git rev-list --no-walk f546502984ff39c9dacb5143e768d390efccb3a8 # timeout=10
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Render Configurations)
[Pipeline] sh
[workspace] Running shell script
+ ansible-playbook generate_configurations.yaml

PLAY [Generate IOS-XE Router Configurations] *****

TASK [template] *****
fatal: [acme-sea-rt1]: FAILED! => {"changed": false, "checksum": "8416e5d4c918e4ae396efd785696a4f7953ac2fe", "msg": "Destination directory configs does not exist"}

PLAY RECAP *****
acme-sea-rt1 : ok=0    changed=0    unreachable=0    failed=1

[Pipeline]
[Pipeline] // stage
[Pipeline]
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 2
Finished: FAILURE

```

Page generated: May 18, 2018 2:01:58 PM UTC REST API Jenkins ver. 2.107.2

Well that makes a lot of sense doesn't it! Git won't track our empty folder. The best way we can address this is by adding a task in our Playbook to ensure that the folder exists. Of course there is already a handy Ansible module (core module) for this: the [file module](#).

Add a Play to your Playbook to ensure that the "configs" directory is present:

```

- name: Ensure Configs Directory Present
  connection: local
  hosts: localhost
  gather_facts: no
  tasks:
    - file:
        path: configs/
        state: directory

```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
- name: Ensure Configs Directory Preset
  connection: local
  hosts: localhost
  gather_facts: no
  tasks:
    - file:
        path: configs/
        state: directory
- name: Generate IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  tasks:
    - template:
        src=templates/cisco_ios_xe_router.j2
        dest= configs/{{ inventory_hostname }}
- name: Generate NX-OS Router Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: no
  tasks:
    - template:
        src=templates/cisco_nxos.j2
        dest= configs/{{ inventory_hostname }}
- name: Generate ASA Configurations
  connection: local
  hosts: cisco-asa
  gather_facts: no
  tasks:
    - template:
        src=templates/cisco_asa.j2
        dest= configs/{{ inventory_hostname }}

"
"
"
"generate_configurations.yaml" 34 lines, 814 characters written
  add=0  del=0  mod=0  untracked=0  total=1
```

Once more, push your changes to your repository:

```
git add -u
git commit -m 'removing configurations from repository'
git push -u origin master
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ vi generate_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ git add -u
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m 'ensure configs directory'
[master e5f7755] ensure configs directory
Committer: ignw <ignw@ignw-jumphost.ignw.io>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 8 insertions(+)
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 384 bytes | 384.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:carlniger/my_network_as_code
  e605dc8..e5f7755  master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_network_as_code$
```

Build your Pipeline one more time:

my\_network\_as\_code [Jenkins] - Mozilla Firefox

my\_network\_as\_code | carligner/my\_network\_ | +  
 10.10.0.253:8080/job/my\_network\_as\_code/

**Jenkins**

Back to Dashboard | Status | Changes | Build Now | Delete Pipeline | Configure | Full Stage View | GitHub | Pipeline Syntax

**Pipeline my\_network\_as\_code**

**Recent Changes**

**Stage View**

|                                                      | Checkout Repository | Render Configurations | Unit Testing | Deploy Configurations to Dev | Functional/Integration Testing | Promote Configurations to Prod |
|------------------------------------------------------|---------------------|-----------------------|--------------|------------------------------|--------------------------------|--------------------------------|
| Average stage times:<br>(Average full run time: ~3s) | 721ms               | 1s                    | 40ms         | 40ms                         | 41ms                           | 41ms                           |
| #6<br>May 18, 2018 3:23 PM<br>15:23<br>1 commit      | 740ms               | 3s                    | 39ms         | 39ms                         | 38ms                           | 38ms                           |
| #5<br>May 18, 2018 1:57 PM<br>13:57<br>1 commit      | 652ms               | 2s<br>failed          |              |                              |                                |                                |
| #4<br>May 18, 2018 1:48 PM<br>13:48<br>1 commit      | 966ms               | 3s                    | 37ms         | 40ms                         | 36ms                           | 36ms                           |
| #3<br>May 18, 2018 1:11 AM<br>01:11<br>No Changes    | 996ms               | 46ms                  | 42ms         | 38ms                         | 42ms                           | 48ms                           |
| #2<br>May 18, 2018 1:00 AM<br>01:00<br>No Changes    | 927ms               | 40ms                  | 40ms         | 39ms                         | 43ms                           | 44ms                           |

**Build History**

- #6 May 18, 2018 3:23 PM
- #5 May 18, 2018 1:57 PM
- #4 May 18, 2018 1:48 PM
- #3 May 18, 2018 1:11 AM
- #2 May 18, 2018 1:00 AM
- #1 May 18, 2018 12:19 AM

RSS for all RSS for failures

Much better, and the console output should look as you would expect, now with all four of our Plays completing successfully.

my\_network\_as\_code #6 Console [Jenkins] - Mozilla Firefox

my\_network\_as\_code x carlniger/my\_network x +  
10.10.0.253:8080/job/my\_network\_as\_code/6/console

```
Checking out Revision e5f77556daabbe319766b547d600dec375fb9d8c (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f e5f77556daabbe319766b547d600dec375fb9d8c
Commit message: "ensure configs directory"
> git rev-list --no-walk e605dc87296a0229e51f9f64c417b25357e9c273 # timeout=10
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Render Configurations)
[Pipeline] sh
[workspace] Running shell script
+ ansible-playbook generate_configurations.yaml

PLAY [Ensure Configs Directory Preset] *****

TASK [file] *****
changed: [localhost]

PLAY [Generate IOS-XE Router Configurations] *****

TASK [template] *****
changed: [acme-sea-rtr1]

PLAY [Generate NX-OS Router Configurations] *****

TASK [template] *****
changed: [acme-sea-nxos1]

PLAY [Generate ASA Configurations] *****

TASK [template] *****
changed: [acme-sea-asal]

PLAY RECAP *****
acme-sea-asal : ok=1    changed=1    unreachable=0    failed=0
acme-sea-nxos1 : ok=1    changed=1    unreachable=0    failed=0
acme-sea-rtr1 : ok=1    changed=1    unreachable=0    failed=0
localhost      : ok=1    changed=1    unreachable=0    failed=0

[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Unit Testing)
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy Configurations to Dev)
```

We can also confirm that our configs are present in our Jenkins workspace:

```
ignw@ignw-jenkins: /var/lib/jenkins/jobs/my_network_as_code/workspace x
File Edit View Search Terminal Help
ignw@ignw-jenkins:/var/lib/jenkins/jobs/my_network_as_code$ cd workspace
ignw@ignw-jenkins:/var/lib/jenkins/jobs/my_network_as_code/workspace$ ls
ansible.cfg          group_vars  legacy_configs
configs              host_vars   README.md
deploy_configurations.yaml inventory templates
generate_configurations.yaml Jenkinsfile
ignw@ignw-jenkins:/var/lib/jenkins/jobs/my_network_as_code/workspace$ ls configs/
acme-sea-asal  acme-sea-nxos1  acme-sea-rtr1
ignw@ignw-jenkins:/var/lib/jenkins/jobs/my_network_as_code/workspace$
```

## Requirements, requirements, requirements

Alright, we're now ready to move on. Our next step should be to add some sort of unit testing to our Pipeline. Generally, in a software development world this would be where unit tests are ran to validate that the code is acting as it should be. In networking this is a bit trickier as we don't necessarily have a "unit testing" framework that we can use. For now, we'll leave this stage empty and move on to actually pushing our configurations out as we did previously. We will be revisiting this in a bit though!

Think back to what was needed for the configurations to be pushed with our other Playbook -- not a ton, but we did have to get the NAPALM module setup. We'll need to have that sorted out for our Jenkins server as well. We should also consider using a "venv" (virtual environment) for our Python libraries -- this will ensure that our libraries used for testing are not the system libraries, thus allowing us to install specific versions/libraries and not have any conflict/confusion. Again, in the future this could be converted into a container further isolating this from the Jenkins system itself.

To create a virtual environment the additional module needs to be installed on the system, just like Ansible, this has been taken care of for you, in general it is very simple to do though: "apt-get install python3-venv".

Within the venv we'll need to install our requirements, but before that we need to actually define what those are. In this case, the only real requirements (outside of venv and Ansible) are NAPAM (the "standalone" Python library) and NAPALM-Ansible (the Ansible module). A common convention in Python projects is to have a "requirements.txt" file that lists the -- you guessed it -- requirements for a project. Normally we would list "napalm", and "napalm-ansible" in this document, however due to an open pull request to mitigate a Python 2/3 compatibility issue we will need to manually clone a fork of the NAPALM Ansible library. For now, create

your requirements.txt file and add "napalm" to it, and add it to your repository:

```
touch requirements.txt
echo "napalm" >> requirements.txt
git add requirements.txt
```

Now we need to turn our attention to updating our Jenkinsfile to actually build the venv and install our requirements. As with our previous steps, we can simply use shell commands to do this. Modify your Jenkinsfile to match the commands below:

```
stage ('Deploy Configurations to Dev') {
    sh 'python3 -m venv jenkins_build'
    sh 'jenkins_build/bin/python -m pip install -r requirements.txt'
}
```

*Note:* Each line of "sh" in a Jenkinsfile is essentially its own shell script, as such we cannot activate a venv as we normally might because every line is a new script. To get around this we can simply use fully qualified commands -- in this case, basically calling specifically the Python instance created in our venv.

Next we will need to clone the forked NAPALM Ansible repository, and install it into our venv.

```
stage ('Deploy Configurations to Dev') {
    sh 'python3 -m venv jenkins_build'
    sh 'jenkins_build/bin/python -m pip install -r requirements.txt'
    sh 'git clone https://github.com/carlniger/napalm-ansible'
    sh 'cp -r napalm-ansible/napalm_ansible/ jenkins_build/lib/python3.6/site-packages/'
    sh 'jenkins_build/bin/python napalm-ansible/setup.py install'
}
```

The above bits will clone the repository, copy the Python modules to the venv's packages path, and install the setup module.

We have one last bit to satisfy -- we need to update the ansible.cfg file with the proper paths to the NAPALM Ansible modules as they are not part of Ansible core. To do this in a shell-able fashion, we can use sed to replace the paths from the ansible.cfg from our jumphost with the appropriate paths for our Jenkins server.

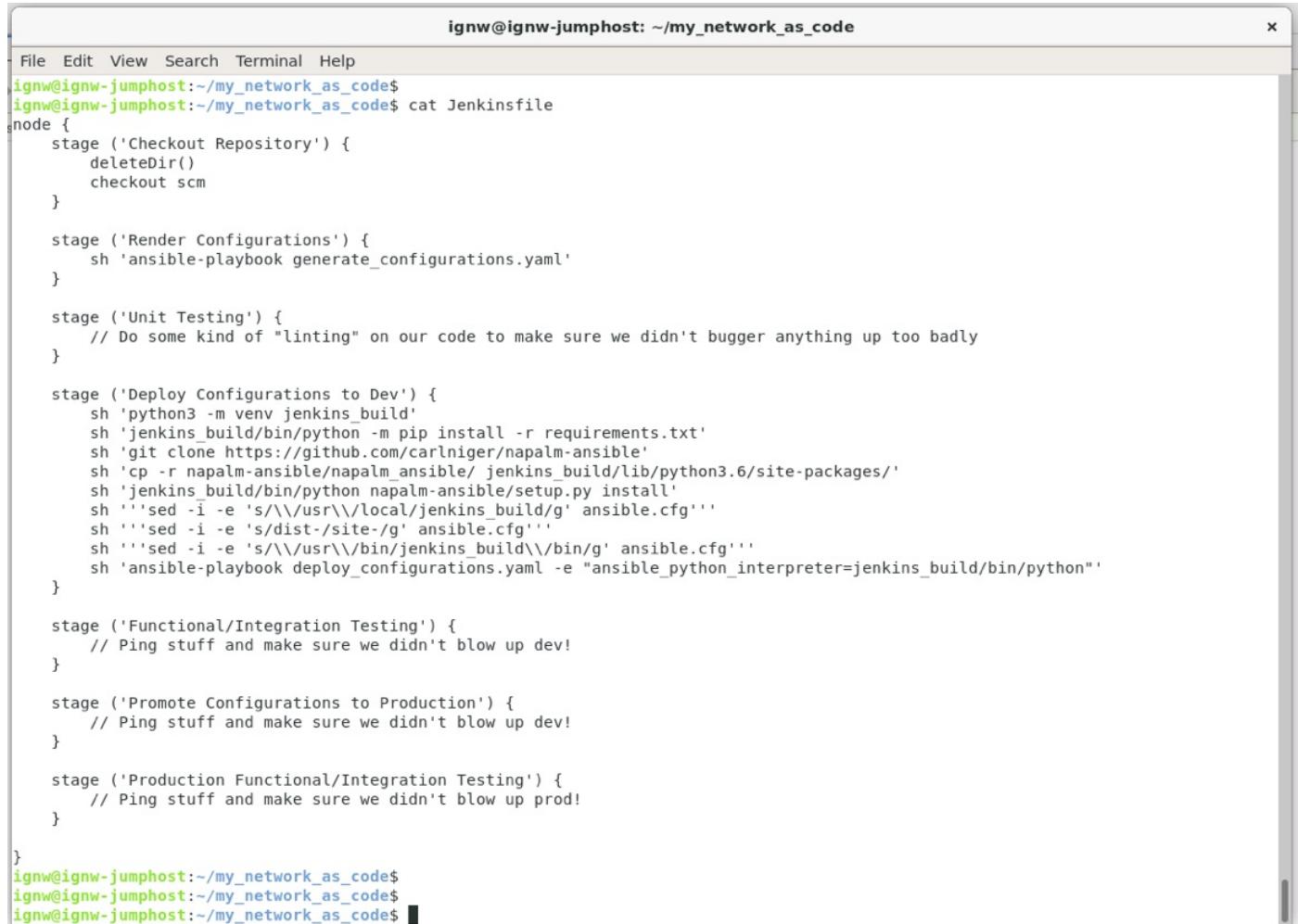
Finally, we can run our Playbook. There is one minor caveat here -- recall how we set the "ansible\_python\_interpreter" in the "all" host vars? We will need to override that on our Jenkins server to point to use the Python binaries in our venv. We can do this simply by running our Playbook and calling the "-e" switch, or the "EXTRA\_VARS" option, and passing in our interpreter as shown below.

```

stage ('Deploy Configurations to Dev') {
    sh 'python3 -m venv jenkins_build'
    sh 'jenkins_build/bin/python -m pip install -r requirements.txt'
    sh 'git clone https://github.com/carligner/napalm-ansible'
    sh 'cp -r napalm-ansible/napalm_ansible/ jenkins_build/lib/python3.6/site-packages/'
    sh 'jenkins_build/bin/python napalm-ansible/setup.py install'
    sh '''sed -i -e 's/\\usr\\\\local/jenkins_build/g' ansible.cfg'''
    sh '''sed -i -e 's/dist-/site-/g' ansible.cfg'''
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

```

Your "finished" product should look similar to this:



The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~/my\_network\_as\_code". The window displays a Jenkinsfile with the following content:

```

File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ cat Jenkinsfile
node {
    stage ('Checkout Repository') {
        deleteDir()
        checkout scm
    }

    stage ('Render Configurations') {
        sh 'ansible-playbook generate_configurations.yaml'
    }

    stage ('Unit Testing') {
        // Do some kind of "linting" on our code to make sure we didn't bugger anything up too badly
    }

    stage ('Deploy Configurations to Dev') {
        sh 'python3 -m venv jenkins_build'
        sh 'jenkins_build/bin/python -m pip install -r requirements.txt'
        sh 'git clone https://github.com/carligner/napalm-ansible'
        sh 'cp -r napalm-ansible/napalm_ansible/ jenkins_build/lib/python3.6/site-packages/'
        sh 'jenkins_build/bin/python napalm-ansible/setup.py install'
        sh '''sed -i -e 's/\\usr\\\\local/jenkins_build/g' ansible.cfg'''
        sh '''sed -i -e 's/dist-/site-/g' ansible.cfg'''
        sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }

    stage ('Functional/Integration Testing') {
        // Ping stuff and make sure we didn't blow up dev!
    }

    stage ('Promote Configurations to Production') {
        // Ping stuff and make sure we didn't blow up dev!
    }

    stage ('Production Functional/Integration Testing') {
        // Ping stuff and make sure we didn't blow up prod!
    }
}

ignw@ignw-jumphost:~/my_network_as_code$ ignw@ignw-jumphost:~/my_network_as_code$ ignw@ignw-jumphost:~/my_network_as_code$ 

```

Update the Jenkinsfile in git, and commit the code:

```
git add -u  
git commit -m 'added requirements, updated jenkinsfile'  
git push -u origin master
```

```
ignw@ignw-jumphost: ~/my_network_as_code  
File Edit View Search Terminal Help  
    // Ping stuff and make sure we didn't blow up dev!  
}  
  
stage ('Promote Configurations to Production') {  
    // Ping stuff and make sure we didn't blow up dev!  
}  
  
stage ('Production Functional/Integration Testing') {  
    // Ping stuff and make sure we didn't blow up prod!  
}  
  
}  
ignw@ignw-jumphost:~/my_network_as_code$ git add -u  
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m 'added requirements, updated jenkinsfile'  
[master ac73be2] added requirements, updated jenkinsfile  
Committer: ignw <ignw@ignw-jumphost.ignw.io>  
Your name and email address were configured automatically based  
on your username and hostname. Please check that they are accurate.  
You can suppress this message by setting them explicitly. Run the  
following command and follow the instructions in your editor to edit  
your configuration file:  
  
git config --global --edit  
  
After doing this, you may fix the identity used for this commit with:  
  
git commit --amend --reset-author  
  
2 files changed, 9 insertions(+), 1 deletion(-)  
create mode 100644 requirements.txt  
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master  
Counting objects: 4, done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (3/3), done.  
Writing objects: 100% (4/4), 632 bytes | 632.00 KiB/s, done.  
Total 4 (delta 2), reused 0 (delta 0)  
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.  
To github.com:carlniger/my_network_as_code  
 e5f7755..ac73be2 master -> master  
Branch master set up to track remote branch master from origin.  
ignw@ignw-jumphost:~/my_network_as_code$
```

Run your build, hopefully its looking good!

my\_network\_as\_code #9 Console [Jenkins] - Mozilla Firefox

my\_network\_as\_code carlniger/my\_network 10.10.0.253:8080/job/my\_network\_as\_code/9/console Jenkins my\_network\_as\_code #9

```
        }
    ok: [acme-sea-nxos1] => (item=None) => {
        "msg": " no shutdown"
    }
    ok: [acme-sea-nxos1] => (item=None) => {
        "msg": " switchport mode trunk"
    }
    ok: [acme-sea-nxos1] => (item=None) => {
        "msg": " switchport trunk native vlan 1000"
    }
    ok: [acme-sea-nxos1] => (item=None) => {
        "msg": " no shutdown"
    }
    ok: [acme-sea-nxos1] => (item=None) => {
        "msg": "boot nxos bootflash:/nxos.7.0.3.I7.3.bin"
    }

PLAY [Deploy Cisco ASA Configurations] *****

TASK [asa_config] *****
changed: [acme-sea-asal]

PLAY RECAP *****
acme-sea-asal : ok=1    changed=1    unreachable=0    failed=0
acme-sea-nxos1 : ok=2    changed=1    unreachable=0    failed=0
acme-sea-rtr1  : ok=2    changed=1    unreachable=0    failed=0

[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Functional/Integration Testing)
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Promote Configurations to Production)
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Production Functional/Integration Testing)
[Pipeline]
[Pipeline] // stage
[Pipeline]
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Page generated: May 19, 2018 5:12:30 PM UTC REST API Jenkins ver.2.107.2

With the core of our work flow wrapped up in a Jenkins Pipeline, we can now focus on the other stages that we left empty -- this is where we'll start seeing a ton of value from this type of Infrastructure as Code deployment. Next we'll be looking at adding in some basic testing both unit testing and functional testing.

# Testing in the Pipeline

## Overview and Objectives

In this lab you will add in some rudimentary testing to your Pipeline. The goal here is to try to catch as many errors as possible as early as possible. We will take advantage of some basic built in syntax-checking with Ansible, as well as using the Ansible [asalnxosios]-ping modules for testing connectivity.

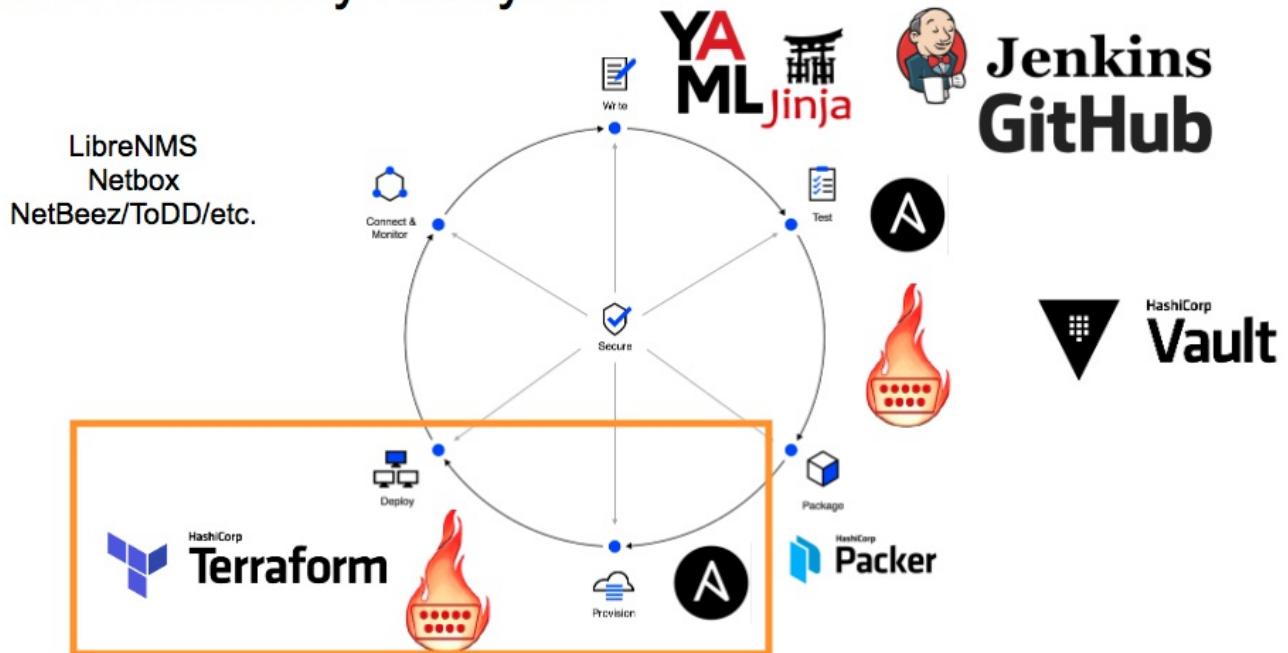
Objectives:

- Create a testing Playbook
- Update the Pipeline to include testing stage(s)

## Network Delivery Lifecycle Overview

We want to ensure that testing is done as pervasively as possible to ensure that our builds are successful and the end result of our deployments are operating as desired. In a perfect world we will be testing in all phases, in reality we are predominantly testing during our deployment phase though.

## Network Delivery Lifecycle



## What are we Testing?

As outlined there are types of testing that occur in software projects. Unit and functional testing are the two primary categories.

Unit testing ensures that code behaves as expected -- unit tests are designed to break a program into tiny pieces and test each "chunk". These tests can then be ran during the execution of a pipeline (as code is committed) to validate or assert that each small unit of code works properly, thus giving greater confidence that the program as a whole works as intended. This is particularly useful in compiled languages as this serves as

a way to validate code before the compiling process, which can sometimes take quite a bit of time.

Functional tests are pretty much what they sound like -- testing whether or not the program as a whole functions as desired. You may have also heard terms like "smoke testing", "regression testing", or "usability testing" -- these all fall under the umbrella of functional testing.

All of this is well and good for software development, but how can we apply this to our Pipeline and our network as code? As a human, how would you validate your configurations?

There are some obvious ways that we can functionally validate our configurations -- ping and traceroute of course spring to mind. We can wrap up some tests that we as humans would do in something like an Ansible Playbook and integrate that as part of our testing suite.

Testing firewalls could involve testing telnet to a specific port to validate whether or not that traffic is allowed. Validating quantity of routes in a routing table is relatively straight forward, but could be a very solid indicator of issues (i.e. missing routes or having a ton of extra routes all the sudden). There are even open open source fully distributed testing engines which could be integrated to validate the network is behaving as desired from all points within the network.

At first glance, unit testing is a bit harder to relate in the network or infrastructure as code world, but there are certainly things that we can do on this front. Perhaps the lowest hanging fruit is to do a simple syntax check on our Ansible Playbooks -- this is a built in utility in Ansible that can help to ensure that any new Plays or Tasks that we've created comply with proper Yaml/Ansible syntax.

Thinking a bit bigger, we could also use open source parsing libraries such as TextFSM to parse and validate the syntax of our templated configurations. We could validate that no illegal IP addresses exist within the configurations, validate that VLANs are within an acceptable range, and even compare IPs/VLANs to some sort of CMDB to validate that there are no overlapping/duplicate entries.

For now, we'll start small and get some simple "wins" knocked out. The easiest thing we can do is to use that syntax check built into Ansible to validate our Playbooks before we execute them.

## Ansible syntax-check

Edit your Jenkins file to add a new stage to check the syntax of the "generate\_configurations.yaml" Playbook before it is ran:

```
stage ('Validate Generate Configurations Playbook') {
    sh 'ansible-playbook generate_configurations.yaml --syntax-check'
}
```

While we're at it we may as well add the syntax check to the second Playbook -- it makes sense to do this in the "Unit Testing" stage that we already have:

```
stage ('Unit Testing') {
    sh 'ansible-playbook depoy_configurations.yaml --syntax-check'
}
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
node {
    stage ('Checkout Repository') {
        deleteDir()
        checkout scm
    }
    stage ('Validate Generate Configurations Playbook') {
        sh 'ansible-playbook generate_configurations.yaml --syntax-check'
    }
    stage ('Render Configurations') {
        sh 'ansible-playbook generate_configurations.yaml'
    }
    stage ('Unit Testing') {
        sh 'ansible-playbook deploy_configurations.yaml --syntax-check'
    }
    stage ('Deploy Configurations to Dev') {
        sh 'python3 -m venv jenkins_build'
        sh 'jenkins_build/bin/python -m pip install -r requirements.txt'
        sh 'git clone https://github.com/carlniger/napalm-ansible'
        sh 'cp -r napalm-ansible/napalm_ansible/ jenkins_build/lib/python3.6/site-packages/'
        sh 'jenkins_build/bin/python napalm-ansible/setup.py install'
        sh "'sed -i -e 's/^\[local\]/jenkins_build/g' ansible.cfg'"
        sh "'sed -i -e 's/^\[dist-site\]/g' ansible.cfg'"
        sh "'sed -i -e 's/^\[bin\]/jenkins_build/bin/g' ansible.cfg'"
        sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }
    stage ('Functional/Integration Testing') {
        // Ping stuff and make sure we didn't blow up dev!
    }
    stage ('Promote Configurations to Production') {
        // Ping stuff and make sure we didn't blow up dev!
    }
    stage ('Production Functional/Integration Testing') {
        // Ping stuff and make sure we didn't blow up prod!
    }
}
"Jenkinsfile" 43 lines, 1501 characters written
```

Push your configurations up to Github and run your build.

The screenshot shows a Jenkins console log for a job named "my\_network\_as\_code #10". The log output is as follows:

```

my_network_as_code #10 Console [Jenkins] - Mozilla Firefox
my_network_as_code x carlniger/my_network x +
10.10.0.253:8080/job/my_network_as_code/10/console
Jenkins > my_network_as_code > #10

TASK [template] *****
changed: [acme-sea-rtr1]

PLAY [Generate NX-OS Router Configurations] *****

TASK [template] *****
changed: [acme-sea-nxos1]

PLAY [Generate ASA Configurations] *****

TASK [template] *****
changed: [acme-sea-asal1]

PLAY RECAP *****
acme-sea-asal1 : ok=1    changed=1    unreachable=0    failed=0
acme-sea-nxos1 : ok=1    changed=1    unreachable=0    failed=0
acme-sea-rtr1  : ok=1    changed=1    unreachable=0    failed=0
localhost       : ok=1    changed=1    unreachable=0    failed=0

[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Unit Testing)
[Pipeline] sh
[workspace] Running shell script
+ ansible-playbook deploy_configurations.yaml --syntax-check
ERROR! no action detected in task. This often indicates a misspelled module name, or incorrect module path.

The error appears to have been in '/var/lib/jenkins/jobs/my_network_as_code/workspace/deploy_configurations.yaml': line 14,
column 7, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

tasks:
  - napalm_install_config:
    ^ here
[Pipeline]
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 4
Finished: FAILURE

```

The error message is highlighted with an orange box: "The error appears to have been in '/var/lib/jenkins/jobs/my\_network\_as\_code/workspace/deploy\_configurations.yaml': line 14, column 7, but may be elsewhere in the file depending on the exact syntax problem."

Page generated: May 19, 2018 7:03:00 PM UTC REST API Jenkins ver. 2.107.2

This is a perfect example of why we want to include testing into our workflow. We can see in the above screen shot that our Pipeline failed, and that it appears that Ansible is a bit confused about our NAPALM tasks, why do you think that is? Recall that we "pointed" Ansible at our venv in the previous lab. We did this in part to ensure that every time the Pipeline is ran, we have a consistent experience -- using the same installation of Python, with the same modules, etc..

When we ran our syntax-check on the "deploy\_configurations.yaml" Playbook we failed to specify the proper Python binaries/modules which caused our Playbook to fail. While this is obviously an engineered error, we can see that our Pipeline essentially "failed out" when it encountered an error, preventing any configurations to be pushed (even if it is just to the "dev" environment).

Update the Jenkinsfile to add the "-e" switch pointing to the venv Python binaries on all of the "ansible-playbook" tasks.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
node {
    stage ('Checkout Repository') {
        deleteDir()
        checkout scm
    }

    stage ('Validate Generate Configurations Playbook') {
        sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    }

    stage ('Render Configurations') {
        sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }

    stage ('Unit Testing') {
        sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    }

    stage ('Deploy Configurations to Dev') {
        sh 'python3 -m venv jenkins_build'
        sh 'jenkins_build/bin/python -m pip install -r requirements.txt'
        sh 'git clone https://github.com/carlniger/napalm-ansible'
        sh 'cp -r napalm-ansible/napalm_ansible/ jenkins_build/lib/python3.6/site-packages/'
        sh 'jenkins_build/bin/python napalm-ansible/setup.py install'
        sh '''sed -i -e 's/\\usr\\\\local/jenkins_build/g' ansible.cfg'''
        sh '''sed -i -e 's/\\dist-/site-/g' ansible.cfg'''
        sh '''sed -i -e 's/\\usr\\\\bin/jenkins_build\\\\bin/g' ansible.cfg'''
        sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }

    stage ('Functional/Integration Testing') {
        // Ping stuff and make sure we didn't blow up dev!
    }

    stage ('Promote Configurations to Production') {
        // Ping stuff and make sure we didn't blow up dev!
    }

    stage ('Production Functional/Integration Testing') {
        // Ping stuff and make sure we didn't blow up prod!
    }
}

"Jenkinsfile" 43 lines, 1672 characters written
```

## Unit Testing in Action

This does raise one issue for us -- we need to move the setup of our venv to an earlier stage in our Jenkinsfile so that we can utilize it for the config generation and checking. Move all of the environment setup tasks to a new stage "Setup Environment":

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
node {
    stage ('Checkout Repository') {
        deleteDir()
        checkout scm
    }

    stage ('Setup Environment') {
        sh 'python3 -m venv jenkins_build'
        sh 'jenkins_build/bin/python -m pip install -r requirements.txt'
        sh 'git clone https://github.com/carlniger/napalm-ansible'
        sh 'cp -r napalm-ansible/napalm_ansible/ jenkins_build/lib/python3.6/site-packages/'
        sh 'jenkins_build/bin/python napalm-ansible/setup.py install'
        sh '''sed -i -e 's/\\usr\\\\local/jenkins_build/g' ansible.cfg'''
        sh '''sed -i -e 's/dist-/site-/g' ansible.cfg'''
        sh '''sed -i -e 's/\\usr\\\\bin/jenkins_build\\\\bin/g' ansible.cfg'''
    }

    stage ('Validate Generate Configurations Playbook') {
        sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    }

    stage ('Render Configurations') {
        sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }

    stage ('Unit Testing') {
        sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    }

    stage ('Deploy Configurations to Dev') {
        sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }

    stage ('Functional/Integration Testing') {
        // Ping stuff and make sure we didn't blow up dev!
    }

    stage ('Promote Configurations to Production') {
        // Ping stuff and make sure we didn't blow up dev!
    }

    stage ('Production Functional/Integration Testing') {
        // Ping stuff and make sure we didn't blow up prod!
    }
}
"Jenkinsfile" 46 lines, 1713 characters
```

Push your updates to GitHub and re-run your build:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ vi Jenkinsfile
ignw@ignw-jumphost:~/my_network_as_code$ git add -u
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m 'update jenkinsfile'
[master 36a3dbe] update jenkinsfile
Committer: ignw <ignw@ignw-jumphost.ignw.io>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 11 insertions(+), 8 deletions(-)
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 340 bytes | 340.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:carlniger/my_network_as_code
  253ef95..36a3dbe master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_network_as_code$
```

The screenshot shows the Jenkins interface for the job 'my\_network\_as\_code'. At the top, there's a 'Stage Logs (Unit Testing)' window displaying command-line output:

```
Shell Script -- ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check -- (self time 1s)

[workspace] Running shell script
+ ansible-playbook deploy_configurations.yaml -e ansible_python_interpreter=jenkins_build/bin/python --syntax-check

playbook: deploy_configurations.yaml
```

Below the logs, there's a 'Stage View' section with a timeline and a grid of stages:

| Checkout Repository | Setup Environment | Validate Configurations Playbook | Render Configurations | Unit Testing | Deploy Configurations to Dev | Functional/Integration Testing | Promote Configurations to Production | Production Functional/Integration Testing |
|---------------------|-------------------|----------------------------------|-----------------------|--------------|------------------------------|--------------------------------|--------------------------------------|-------------------------------------------|
| 814ms               | 35s               | 1s                               | 3s                    | 1s           | 1min 49s                     | 36ms                           | 37ms                                 | 37ms                                      |
| 814ms               | 35s               | 1s                               | 3s                    | 1s           | 1min 49s                     | 36ms                           | 37ms                                 | 37ms                                      |

On the left, there's a sidebar with a 'trend' chart and a 'Permalinks' section listing recent builds.

Great, let's break one of our Playbooks to validate that our unit testing is doing what we think it is. Create some sort of invalid Yaml syntax then push your code to GitHub and re-run your build.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
---
- name: Ensure Configs Directory Preset
  connection: local
  hosts: localhost
  gather_facts: no
  tasks:
    - file
      path: configs/
      state: directory
- name: Generate IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  tasks:
    - template:
        src=templates/cisco_ios_xe_router.j2
        dest=configs/{{ inventory_hostname }}
- name: Generate NX-OS Router Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: no
  tasks:
    - template:
        src=templates/cisco_nxos.j2
        dest=configs/{{ inventory_hostname }}
- name: Generate ASA Configurations
  connection: local
  hosts: cisco-asa
  gather_facts: no
  tasks:
    - template:
        src=templates/cisco_asa.j2
        dest=configs/{{ inventory_hostname }}

"generate_configurations.yaml" 34 lines, 813 characters
```

In the above screen shot the ":" after "file" has been removed.

As we had hoped, our Pipeline build failed and stopped before pushing any changes:

The screenshot shows a Jenkins console log for job 'my\_network\_as\_code' build #13. The log output is as follows:

```
Best match: certifi 2018.4.16
Adding certifi 2018.4.16 to easy-install.pth file

Using /var/lib/jenkins/jobs/my_network_as_code/workspace/jenkins_build/lib/python3.6/site-packages
Searching for asn1crypto==0.24.0
Best match: asn1crypto 0.24.0
Adding asn1crypto 0.24.0 to easy-install.pth file

Using /var/lib/jenkins/jobs/my_network_as_code/workspace/jenkins_build/lib/python3.6/site-packages
Finished processing dependencies for napalm-ansible==0.9.1
[Pipeline] sh
[workspace] Running shell script
+ sed -i -e s@/usr/local/jenkins_build@g ansible.cfg
[Pipeline] sh
[workspace] Running shell script
+ sed -i -e s@dist-/site-/g ansible.cfg
[Pipeline] sh
[workspace] Running shell script
+ sed -i -e s@/usr/bin/jenkins_build/bin@g ansible.cfg
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Validate Generate Configurations Playbook)
[Pipeline] sh
[workspace] Running shell script
+ ansible-playbook generate_configurations.yaml -e ansible_python_interpreter=jenkins_build/bin/python --syntax-check
ERROR! Syntax Error while loading YAML.
  mapping values are not allowed in this context

The error appears to have been in '/var/lib/jenkins/jobs/my_network_as_code/workspace/generate_configurations.yaml': line 8,
column 13, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:
  - file
    path: configs/
      ^ here
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 4
Finished: FAILURE
```

A yellow box highlights the error message and the offending line of code.

Page generated: May 19, 2018 7:44:26 PM UTC REST API Jenkins ver. 2.107.2

Fix whatever syntax change you broke in your Playbook before continuing.

## Functional Testing

Now we'll work on crafting a Playbook to add in some very basic functional testing. Create a new Playbook called "validate\_configurations.yaml" in your repository.

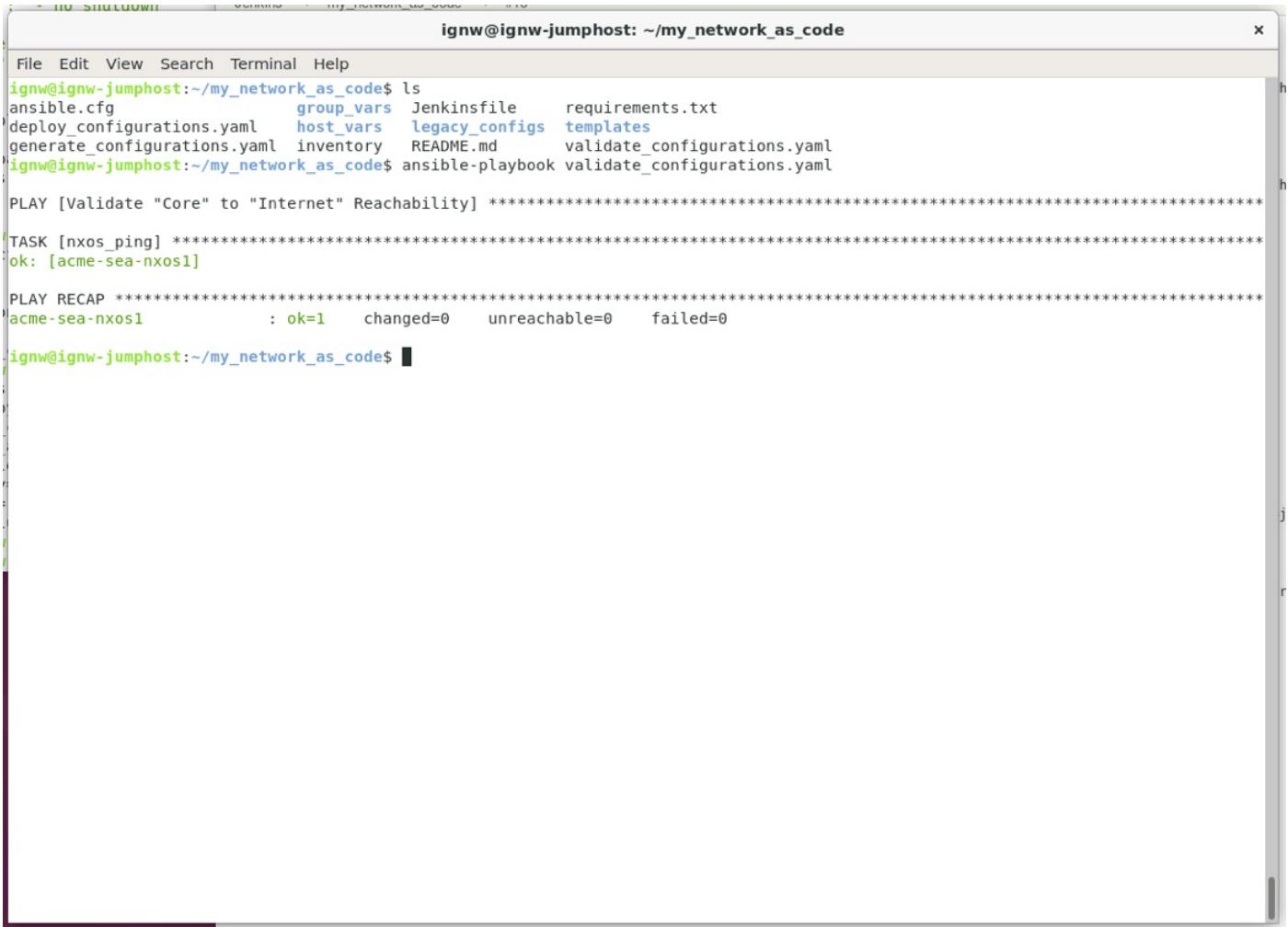
We will first use the "nxos\_ping" module to validate that our core device has reachability to the "Internet" (loopback with IP address 8.8.8.8 on the edge router). This module is very similar to the modules previously used, take a quick look at the [documentation](#) to familiarize yourself.

We can see that we will need a provider -- containing the relevant login information, and then a ping task with destination(s) and a source. Add the following to your new Playbook:

```
---
- name: Validate "Core" to "Internet" Reachability
  connection: local
  hosts: acme-sea-nxos1
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
  tasks:
    - nxos_ping:
        provider: "{{ creds }}"
        dest: 8.8.8.8
        vrf: default
```

Run the Playbook to make sure it is working correctly.

```
ansible-playbook validate_configurations.yaml
```

A screenshot of a terminal window titled "ignw@ignw-jumphost: ~/my\_network\_as\_code". The window shows the output of an Ansible command-line interface. It starts with the user navigating to the directory, listing files, and running an ansible-playbook. The playbooks run successfully, with one task for "nxos\_ping" on "acme-sea-nxos1" which succeeds ("ok: [acme-sea-nxos1] : ok=1 changed=0 unreachable=0 failed=0").

```
ignw@ignw-jumphost:~/my_network_as_code$ ls
ansible.cfg      group_vars  Jenkinsfile    requirements.txt
deploy_configurations.yaml  host_vars  legacy_configs  templates
generate_configurations.yaml  inventory  README.md      validate_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook validate_configurations.yaml

PLAY [Validate "Core" to "Internet" Reachability] ****
TASK [nxos_ping] ****
ok: [acme-sea-nxos1] : ok=1    changed=0    unreachable=0    failed=0

PLAY RECAP ****
acme-sea-nxos1        : ok=1    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$
```

Next we can test pings from the ASA to both the "Internet" and internal networks. Unfortunately there is an "nxos\_ping" and an "ios\_ping", but no "asa\_ping" module, lucky for us, we can use the "asa\_command" module to very easily gain the same functionality.

Add another Play (or just a Task if you prefer) using the "asa\_command" module on the ASA device -- this time ping the "Internet" and also the internal IP (of the "core") device. Copy the connection, hosts, gather\_facts, and vars sections from your previous task as we'll need most of that information for the ASA as well. Update the values to match the configurations below:

```

- name: Validate "Core" & "Internet" Reachability from Firewall
  connection: local
  hosts: acme-sea-asa1
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ username }}"
      authorize: yes
      auth_pass: "{{ enable_password }}"
  tasks:
    - asa_command:

```

We now need to tell our Task what we want to ping, since this is "asa\_command" not "xyz\_ping" our syntax will be slightly different -- the "asa\_command" module simply takes a list of commands. Let's provide just one command to start "ping 8.8.8.8".

Next we need to "register" the output from the results of this command so we can properly parse it to determine success or failure of our test.

Finally, Ansible has a "failed\_when" option that we can use to basically tell Ansible when our task fails -- this is useful since the "asa\_command" module is simply going to execute the commands we provide, and only "fail" if it for some reason cannot run the commands.

```

- asa_command:
  provider: "{{ creds }}"
  commands:
    - ping 8.8.8.8
  register: ping_output
  failed_when:
    - "'(5/5)' not in ping_output.stdout_lines[0][-1]"
    - "'(4/5)' not in ping_output.stdout_lines[0][-1]"
- debug:
  msg: "{{ ping_output.stdout_lines[0][-1] }}"

```

In the above snippet we are expecting to see "(5/5)" in the output -- indicating a 100% successful ping, if we do not see this returned in the "stdout\_lines" then the task will fail. Ideally we would leave this at just the "5/5" -- meaning we always want 100% success on our pings, but since this is a lab and sometimes things are a bit sluggish you should add the second line to match "4/5" as well so we don't have any failures even when things are working (even if they're working slowly!).

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
-- name: Validate "Core" to "Internet" Reachability
connection: local
hosts: acme-sea-nxos1
gather_facts: no
vars:
  creds:
    host: "{{ ansible_host }}"
    username: "{{ username }}"
    password: "{{ password }}"
tasks:
- nxos_ping:
    provider: "{{ creds }}"
    dest: 8.8.8.8
    vrf: default
- name: Validate "Core" & "Internet" Reachability from Firewall
connection: local
hosts: acme-sea-asal
gather_facts: no
vars:
  creds:
    host: "{{ ansible_host }}"
    username: "{{ username }}"
    password: "{{ username }}"
    authorize: yes
    auth_pass: "{{ enable_password }}"
tasks:
- asa_command:
    provider: "{{ creds }}"
    commands:
      - ping 8.8.8.8
register: ping_output
failed_when: "'(5/5)' not in ping_output.stdout_lines[0][-1]"
~
~
```

Executing the Playbook should yield successful results.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ vi validate_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook validate_configurations.yaml

PLAY [Validate "Core" to "Internet" Reachability] ****
TASK [nxos_ping] ****
ok: [acme-sea-nxos1]

PLAY [Validate "Core" & "Internet" Reachability from Firewall] ****
TASK [asa_command] ****
ok: [acme-sea-asal]

PLAY RECAP ****
acme-sea-asal      : ok=1    changed=0    unreachable=0    failed=0
acme-sea-nxos1     : ok=1    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$
```

Modify the IP address to something that is unreachable (ex: 8.8.8.81) to validate that our Task fails properly if the ping does indeed fail.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ vi validate_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook validate_configurations.yaml

PLAY [Validate "Core" to "Internet" Reachability] ****
TASK [nxos_ping] ****
ok: [acme-sea-nxos1]

PLAY [Validate "Core" & "Internet" Reachability from Firewall] ****
TASK [asa_command] ****
ok: [acme-sea-asal]

PLAY RECAP ****
acme-sea-asal      : ok=1    changed=0    unreachable=0    failed=0
acme-sea-nxos1     : ok=1    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$ vi validate_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook validate_configurations.yaml

PLAY [Validate "Core" to "Internet" Reachability] ****
TASK [nxos_ping] ****
ok: [acme-sea-nxos1]

PLAY [Validate "Core" & "Internet" Reachability from Firewall] ****
TASK [asa_command] ****
fatal: [acme-sea-asal]: FAILED! => {"changed": false, "failed_when_result": true, "stdout": ["None"], "stdout_lines": ["None"]}

PLAY RECAP ****
acme-sea-asal      : ok=0    changed=0    unreachable=0    failed=1
acme-sea-nxos1     : ok=1    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$
```

Great, set the IP back and let's work on adding our internal IP to our test.

For now create a new list in the vars section of the ASA Play. In the future we would likely want to move this out to a real vars file, but for now this will do nicely. In this list have an entry for 8.8.8.8 and 10.255.255.2 (the acme-sea-nxos1 internal IP address (VLAN 1000)).

Iterate over the list in the Task pinging each device. Your Play should look something like this:

```

- name: Validate "Core" & "Internet" Reachability from Firewall
  connection: local
  hosts: acme-sea-asa1
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ username }}"
      authorize: yes
      auth_pass: "{{ enable_password }}"
    test_ips:
      - 8.8.8.8
      - 10.255.255.2
  tasks:
    - asa_command:
        provider: "{{ creds }}"
        commands:
          - "ping {{ item }}"
      register: ping_output
      failed_when:
        - "'(5/5)' not in ping_output.stdout_lines[0][-1]"
        - "'(4/5)' not in ping_output.stdout_lines[0][-1]"
      with_items: "{{ test_ips }}"

```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
---
- name: Validate "Core" to "Internet" Reachability
  connection: local
  hosts: acme-sea-nxos1
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
  tasks:
    - nxos_ping:
        provider: "{{ creds }}"
        dest: 8.8.8.8
        vrf: default
- name: Validate "Core" & "Internet" Reachability from Firewall
  connection: local
  hosts: acme-sea-asal
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ username }}"
      authorize: yes
      auth_pass: "{{ enable_password }}"
    test_ips:
      - 8.8.8.8
      - 10.255.255.2
  tasks:
    - asa_command:
        provider: "{{ creds }}"
        commands:
          - "ping {{ item }}"
        register: ping_output
        failed_when: "'(5/5)' not in ping_output.stdout_lines[0][-1]"
        with_items: "{{ test_ips }}"
"validate_configurations.yaml" 37 lines, 940 characters
```

## Testing for Failures (on purpose!)

Add another Play using "ios\_ping" to test connectivity from the edge router -- in this case we want to validate that we can ping the outside IP address of the ASA (icmp is permitted), and that we *cannot* ping the inside IP address on the NX-OS device. The "ios\_ping" module has a very standard Ansible option for "state"; to validate that a ping is unsuccessful (meaning we are hoping it will fail) we can set the state to "absent".

```

- name: Validate "Edge" Reachability from Edge Router
  connection: local
  hosts: acme-sea-rtr1
  gather_facts: no
  tags: router
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ username }}"
    test_success_ips:
      - 203.0.113.2
    test_fail_ips:
      - 10.255.255.2
  tasks:
    - name: Test Success Pings
      ios_ping:
        provider: "{{ creds }}"
        dest: "{{ item }}"
        with_items: "{{ test_success_ips }}"
    - name: Test Fail Pings
      ios_ping:
        provider: "{{ creds }}"
        dest: "{{ item }}"
        state: absent
        count: 4
      with_items: "{{ test_fail_ips }}"

```

One other minor caveat -- ensure that the count is set to four or less -- the default timeout is 10 seconds, at the default ping count of 5, with the default interval of 2 seconds we will reach the timeout and our Task will fail. This (the timeout value) could of course be modified, but in our case bumping the count down to four works just fine.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
password: "{{ username }}"
authorize: yes
auth_pass: "{{ enable_password }}"
test_ips:
- 8.8.8.8
- 10.255.255.2
tasks:
- asa_command:
  provider: "{{ creds }}"
  commands:
    - "ping {{ item }}"
register: ping_output
failed_when: "'(5/5)' not in ping_output.stdout_lines[0][-1]"
with_items: "{{ test_ips }}"
- name: Validate "Edge" Reachability from Edge Router
connection: local
hosts: acme-sea-rtr1
gather_facts: no
vars:
  creds:
    host: "{{ ansible_host }}"
    username: "{{ username }}"
    password: "{{ username }}"
  test_success_ips:
    - 203.0.113.2
  test_fail_ips:
    - 10.255.255.2
tasks:
- name: Test Success Pings
  ios_ping:
    provider: "{{ creds }}"
    dest: "{{ item }}"
  with_items: "{{ test_success_ips }}"
- name: Test Fail Pings
  ios_ping:
    provider: "{{ creds }}"
    dest: "{{ item }}"
    state: absent
    count: 4
  with_items: "{{ test_fail_ips }}"
"validate_configurations.yaml" 63 lines, 1599 characters written
```

Run your Playbook, all of your Plays should be OK at this point:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook validate_configurations.yaml

PLAY [Validate "Core" to "Internet" Reachability] ****
TASK [nxos_ping] ****
ok: [acme-sea-nxos1]

PLAY [Validate "Core" & "Internet" Reachability from Firewall] ****
TASK [asa_command] ****
ok: [acme-sea-asal] => (item=8.8.8)
ok: [acme-sea-asal] => (item=10.255.255.2)

PLAY [Validate "Edge" Reachability from Edge Router] ****
TASK [Test Success Pings] ****
ok: [acme-sea-rtr1] => (item=203.0.113.2)

TASK [Test Fail Pings] ****
ok: [acme-sea-rtr1] => (item=10.255.255.2)

PLAY RECAP ****
acme-sea-asal      : ok=1    changed=0    unreachable=0    failed=0
acme-sea-nxos1     : ok=1    changed=0    unreachable=0    failed=0
acme-sea-rtr1      : ok=2    changed=0    unreachable=0    failed=0
ignw@ignw-jumphost:~/my_network_as_code$
```

Update the "Functional/Integration Testing" stage of your Jenkinsfile to run the newly crafted "validate\_configurations.yaml" Playbook.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
node {
    stage ('Checkout Repository') {
        deleteDir()
        checkout scm
    }

    stage ('Setup Environment') {
        sh 'python3 -m venv jenkins_build'
        sh 'jenkins_build/bin/python -m pip install -r requirements.txt'
        sh 'git clone https://github.com/carlniger/napalm-ansible'
        sh 'cp -r napalm-ansible/napalm_ansiible/ jenkins_build/lib/python3.6/site-packages/'
        sh 'jenkins_build/bin/python napalm-ansible/setup.py install'
        sh "'sed -i -e 's/\\usr\\\\local/jenkins_build/g' ansible.cfg'''"
        sh "'sed -i -e 's/dist-/site-/g' ansible.cfg'''"
        sh "'sed -i -e 's/\\bin/jenkins_build\\\\bin/g' ansible.cfg'''"
    }

    stage ('Validate Generate Configurations Playbook') {
        sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    }

    stage ('Render Configurations') {
        sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }

    stage ('Unit Testing') {
        sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    }

    stage ('Deploy Configurations to Dev') {
        sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }

    stage ('Functional/Integration Testing') {
        sh 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }
}

stage ('Promote Configurations to Production') {
    // Ping stuff and make sure we didn't blow up dev!
}

"Jenkinsfile" 46 lines, 1763 characters written
```

We should also add a "syntax-check" step for this Playbook. Let's do that in the "Unit Testing" stage:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
node {
    stage ('Checkout Repository') {
        deleteDir()
        checkout scm
    }

    stage ('Setup Environment') {
        sh 'python3 -m venv jenkins_build'
        sh 'jenkins_build/bin/python -m pip install -r requirements.txt'
        sh 'git clone https://github.com/carlniger/napalm-ansible'
        sh 'cp -r napalm-ansible/napalm_ansible/ jenkins_build/lib/python3.6/site-packages/'
        sh 'jenkins_build/bin/python napalm-ansible/setup.py install'
        sh '''sed -i -e 's/\\usr\\\\local/jenkins_build/g' ansible.cfg'''
        sh '''sed -i -e 's/dist-/site-/g' ansible.cfg'''
        sh '''sed -i -e 's/\\usr\\\\bin/jenkins_build\\\\bin/g' ansible.cfg'''
    }

    stage ('Validate Generate Configurations Playbook') {
        sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    }

    stage ('Render Configurations') {
        sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }

    stage ('Unit Testing') {
        sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
        sh 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    }

    stage ('Deploy Configurations to Dev') {
        sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }

    stage ('Functional/Integration Testing') {
        sh 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    }

    stage ('Promote Configurations to Production') {
        // Ping stuff and make sure we didn't blow up dev!
    }
}
"Jenkinsfile" 47 lines, 1894 characters written
May 10, 2018 1:00 AM
```

Push your changes to GitHub.

```
git add validate_configurations.yaml
git add -u
git commit -m 'validation and jenkinsfile'
git push -u origin master
```

carlniger/my\_network\_as\_code - Mozilla Firefox

my\_network\_as\_code | carlniger/my\_network\_

GitHub, Inc. (US) https://github.com/carlniger/my\_network\_as\_code

This repository Search Pull requests Issues Apps Explore

carlniger / my\_network\_as\_code Private

Code Issues Pull requests Projects Wiki Insights Settings

No description, website, or topics provided.

Add topics

20 commits 1 branch 0 releases 0 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

lgnw validation and jenkinsfile Latest commit 422e455 a minute ago

group\_vars templatized and deployable configurations! 4 days ago

host\_vars templatized and deployable configurations! 4 days ago

legacy\_configs templatized and deployable configurations! 4 days ago

templates templatized and deployable configurations! 4 days ago

Jenkinsfile validation and jenkinsfile a minute ago

README.md whoops, forgot a README 4 days ago

ansible.cfg templatized and deployable configurations! 4 days ago

deploy\_configurations.yaml templatized and deployable configurations! 4 days ago

generate\_configurations.yaml validation and jenkinsfile a minute ago

inventory templatized and deployable configurations! 4 days ago

requirements.txt added requirements\_.undated.jenkinsfile 2 days ago

validate\_configurations.yaml validation and jenkinsfile a minute ago

README.md

automate all the things

[https://github.com/carlniger/my\\_network\\_as\\_code/issues](https://github.com/carlniger/my_network_as_code/issues)

The screenshot shows a GitHub repository page for 'carlniger/my\_network\_as\_code'. The 'Code' tab is selected, displaying a list of files and their commit history. Two specific files are highlighted with orange boxes: 'Jenkinsfile' and 'validate\_configurations.yaml'. Both of these files have a commit message indicating they are validation Jenkinsfiles. The 'Jenkinsfile' was committed a minute ago, and 'validate\_configurations.yaml' was also committed a minute ago. Other files listed include 'README.md', 'ansible.cfg', 'deploy\_configurations.yaml', 'generate\_configurations.yaml', 'inventory', and 'requirements.txt'. The commits for these files are dated between 4 days ago and 2 days ago.

Run your build in Jenkins again.

my\_network\_as\_code #15 Console [Jenkins] - Mozilla Firefox

carlniger/my\_network x | +  
10.10.0.253:8080/job/my\_network\_as\_code/15/console

Jenkins > my\_network\_as\_code > #15

```
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Functional/Integration Testing)
[Pipeline] sh
[Pipeline] Running shell script
+ ansible-playbook validate_configurations.yaml -e ansible_python_interpreter=jenkins_build/bin/python
PLAY [Validate "Core" to "Internet" Reachability] *****

TASK [nxos_ping] *****
ok: [acme-sea-nxos1]

PLAY [Validate "Core" & "Internet" Reachability from Firewall] *****

TASK [asa_command] *****
ok: [acme-sea-asal] => (item=8.8.8.8)
ok: [acme-sea-asal] => (item=10.255.255.2)

PLAY [Validate "Edge" Reachability from Edge Router] *****

TASK [Test Success Pings] *****
ok: [acme-sea-rtr1] => (item=203.0.113.2)

TASK [Test Fail Pings] *****
ok: [acme-sea-rtr1] => (item=10.255.255.2)

PLAY RECAP *****
acme-sea-asal      : ok=1    changed=0    unreachable=0    failed=0
acme-sea-nxos1     : ok=1    changed=0    unreachable=0    failed=0
acme-sea-rtr1      : ok=2    changed=0    unreachable=0    failed=0

[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Promote Configurations to Production)
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Production Functional/Integration Testing)
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Page generated: May 21, 2018 4:05:44 PM UTC REST API Jenkins ver. 2.107.2

To test out our Pipeline in a more meaningful way than simply re-merging the same configs over and over again, let's add a new VLAN and VLAN interface to our Nexus device. At this point this is a simple matter of adding some variables to our host vars file for the specific device.

Update the host\_vars/acme-sea-nxos1 file with the VLAN 1001 information as shown below:

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
features:
  - interface-vlan
  - scp-server
  - nxapi

l2_vlans: {
  '1000': {
    'name': 'routing_to_acme_sea_asal'
  },
  '1001': {
    'name': 'my_new_vlan'
  }
}

l2_ethernet_interfaces: {
  'Ethernet1/2': {
    'mode': 'trunk',
    'native_vlan': 1000,
    'description': 'to acme-sea-asal'
  }
}

svi_interfaces: {
  '1000': {
    'ip': '10.255.255.2',
    'mask': '255.255.255.240',
    'redirects': true,
    'description': 'routing to acme-sea-asal'
  },
  '1001': {
    'ip': '10.255.254.1',
    'mask': '255.255.255.0',
    'redirects': true,
    'description': 'my new vlan'
  }
}

routes: {
  'default': {
    '0.0.0.0 0.0.0.0': {
      'destination': '10.255.255.1'
    }
  }
}
"host_vars/acme-sea-nxos1.yaml" 50 lines, 821 characters

```

We could commit our code, and re-run our build once more at this point and it (should) properly render our updated configuration and push it out nicely. By itself that is pretty neat, but part of the value of this Pipeline/IaC paradigm is that we can also *test* and *validate* what we're doing.

We know now that if we entered some bogus text that would make our Playbooks invalid our build would fail. We also know that our validation Playbook is using basic ping tests to validate reachability. As we add, remove, or modify IP addressing information (for example) we should get into the habit of updating our testing to reflect that. Ensure that the new interface is tested for reachability by updating the list of IPs the ASA will test ping during the validation phase:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
---
- name: Validate "Core" to "Internet" Reachability
  connection: local
  hosts: acme-sea-nxos1
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
  tasks:
    - nxos_ping:
        provider: "{{ creds }}"
        dest: 8.8.8.8
        vrf: default
- name: Validate "Core" & "Internet" Reachability from Firewall
  connection: local
  hosts: acme-sea-asal
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ username }}"
      authorize: yes
      auth_pass: "{{ enable_password }}"
    test_ips:
      - 8.8.8.8
      - 10.255.255.2
      - 10.255.254.1
  tasks:
    - asa_command:
        provider: "{{ creds }}"
        commands:
          - "ping {{ item }}"
        register: ping_output
        failed_when: "'(5/5)' not in ping_output.stdout_lines[0][-1]"
        with_items: "{{ test_ips }}"
- name: Validate "Edge" Reachability from Edge Router
  connection: local
  hosts: acme-sea-rtr1
"validate_configurations.yaml" 64 lines, 1620 characters
```

Finally, push your code to GitHub and run your build, hopefully your results will be successful as shown below!

my\_network\_as\_code #16 Console [Jenkins] - Mozilla Firefox

carlniger/my\_network my\_network #16

10.10.0.253:8080/job/my\_network\_as\_code/16/console

```
"msg": " switchport trunk native vlan 1000"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": " no shutdown"
}
ok: [acme-sea-nxos1] => (item=None) => {
    "msg": "boot nxos bootflash:/nxos.7.0.3.I7.3.bin"
}

PLAY [Deploy Cisco ASA Configurations] *****

TASK [asa_config] *****
changed: [acme-sea-asal]

PLAY RECAP *****
acme-sea-asal      : ok=1    changed=1    unreachable=0    failed=0
acme-sea-nxos1     : ok=2    changed=1    unreachable=0    failed=0
acme-sea-rtr1      : ok=2    changed=1    unreachable=0    failed=0

[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Functional/Integration Testing)
[Pipeline] sh
[workspace] Running shell script
+ ansible-playbook validate_configurations.yaml -e ansible_python_interpreter=jenkins_build/bin/python
PLAY [Validate "Core" to "Internet" Reachability] *****

TASK [nxos_ping] *****
ok: [acme-sea-nxos1]

PLAY [Validate "Core" & "Internet" Reachability from Firewall] *****

TASK [asa_command] *****
ok: [acme-sea-asal] => (item=8.8.8.8)
ok: [acme-sea-asal] => (item=10.255.255.2)
ok: [acme-sea-asal] => (item=10.255.254.1)

PLAY [Validate "Edge" Reachability from Edge Router] *****

TASK [Test Success Pings] *****
ok: [acme-sea-rtr1] => (item=203.0.113.2)

TASK [Test Fail Pings] *****

```

Page generated: May 21, 2018 4:15:46 PM UTC REST API Jenkins ver. 2.107.2

# Rollback Functionality

## Overview and Objectives

In this lab you will setup the automated rollback of configurations in the event that a testing Play(s) fails.

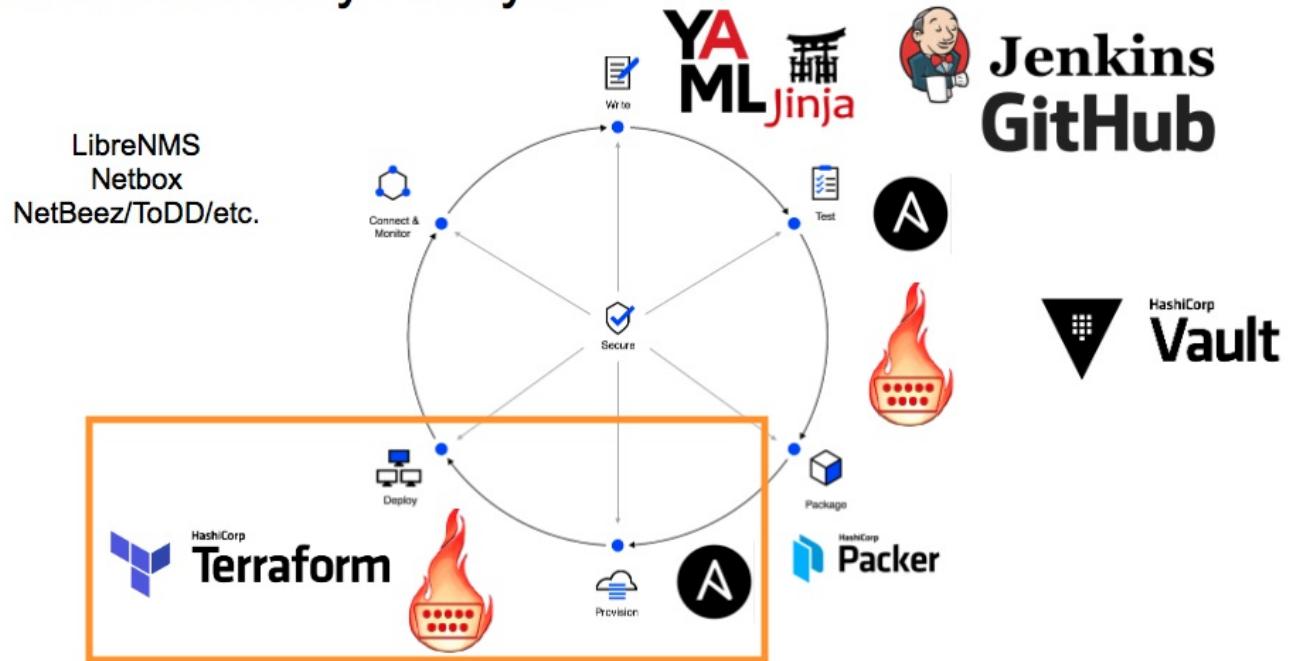
Objectives:

- Create a new Playbook to handle Rollback
- Sort out some minor issues associated with configuration replacements (instead of merges!)
- Update the Pipeline to automatically handle rollbacks on failure

## Network Delivery Lifecycle Overview

The concept of a "Rollback" phase doesn't necessarily enter into the HashiCorp Application Lifecycle. Why? In general modern applications are built, and if they break, they are simply destroyed and re-provisioned. This is the idea of "immutable infrastructure" -- that is configurations/deployments are replaced rather than changed. This concept is a bit tricky in traditional network (or other) infrastructure, and as such we need the ability to conduct rollbacks. This will of course take place during the "deploy" phase if any of our tests fail.

## Network Delivery Lifecycle



## Rollback Time

So far we've created a Pipeline that builds a predictable environment to work in, generates configurations from templates and variables, validates our Playbook syntax, merges the configurations into the development environment, and executes integration/functional testing at a basic level. If any of these steps fail, then our build as a whole fails. What happens if only the last step fails? Our configurations have already been pushed to development... lucky for us it is "just" the development environment, but if the testing fails wouldn't it be great if we could simply rollback to have a clean slate to work from?

As you may have guessed, this is a matter of writing a few new Plays and extending our Pipeline to add a tiny bit of additional logic.

For this task we can continue to use the Ansible NAPALM modules -- this is handy as it gives us the option to continue leveraging Ansible, or convert to pure Python in the future if we so desired. As with the previous tasks, there is no NAPALM support for the ASA so we will use the "asa\_config" Ansible core module to capture the backup.

## Capture Backup/Baseline Configurations

Create a new file in your repository called "backup\_configurations.yaml", you may want to copy/paste from the "deploy\_configurations.yaml" file as there will be a lot of similarity (since it mostly all NAPALM after all).

The Play to backup the IOS-XE device configurations is below. This should look fairly familiar at this point. Enter this into your new file:

```
---
- name: Backup Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  vars:
    creds:
      hostname: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
      dev_os: "{{ device_type }}"
  tags: ios-xe
  tasks:
    - napalm_get_facts:
        provider: "{{ creds }}"
        filter: "config"
        register: config
    - copy:
        content: "{{ config['ansible_facts']['napalm_config']['running'] }}"
        dest: "backup/{{ inventory_hostname }}_config"
```

There are a few minor differences between this Play and the NAPALM Plays we've crafted previously. First and most obvious is that we are using a different module -- this time we're using the "napalm\_get\_facts" module, and passing this module the "config" argument in the filter section. As you would expect this captures the configurations (candidate (which is a NAPALM/JuneOS/IOS-XR kind of idea), running, and startup).

We then "register" the output of this module to the variable "config" so that we can write it to a file in the following copy task.

Create a directory "backup" for use in testing (the copy module will not create the directory for you), then run the Playbook to test it out:

```
mkdir backup
ansible-playbook backup_configurations.yaml
```

The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~/my\_network\_as\_code". The terminal displays the command-line session used to run the Ansible playbook:

```
: "- no shutdown"
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ vi backup_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ mkdir backup
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook backup_configurations.yaml
PLAY [Backup Cisco IOS-XE Router Configurations] ****
TASK [Gathering Facts] ****
ok: [acme-sea-rtr1]
TASK [napalm_get_facts] ****
ok: [acme-sea-rtr1]
TASK [copy] ****
changed: [acme-sea-rtr1]
PLAY RECAP ****
acme-sea-rtr1 : ok=3    changed=1    unreachable=0    failed=0
ignw@ignw-jumphost:~/my_network_as_code$ ls backup
acme-sea-rtr1_config
ignw@ignw-jumphost:~/my_network_as_code$ cat backup/acme-sea-rtr1_config
Building configuration...
Current configuration : 4147 bytes
!
! Last configuration change at 09:16:45 PDT Mon May 21 2018 by *****
! NVRAM config last updated at 09:16:49 PDT Mon May 21 2018 by *****
!
version 16.8
service timestamps debug datetime msec
service timestamps log datetime localtime show-timezone
service password-encryption
platform qfp utilization monitor load 80
no platform punt-keepalive disable-kernel-core
platform console virtual
!
hostname acme-sea-rtr1
!
boot-start-marker
boot-end-marker
!
```

As previously mentioned we'll need to use the Ansible "asa\_config" module to snag the configuration from our ASA. This is even simpler than the NAPALM method as there is a built in "backup" command that we can take advantage of.

```
- name: Backup Cisco ASA Configurations
  connection: local
  hosts: cisco-asa
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ username }}"
      authorize: yes
      auth_pass: "{{ enable_password }}"
  tags: asa
  tasks:
    - asa_config:
        provider: "{{ creds }}"
        backup: yes
```

The "backup" command will look for a directory called "backup" in the path the Playbook is executed from; in our case we already created that, but even if we hadn't this module would create that directory for us.

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
line vty
boot nxos bootflash:/nxos.7.0.3.I7.3.bin

ignw@ignw-jumphost:~/my_network_as_code$ vi backup_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook backup_configurations.yaml

PLAY [Backup Cisco IOS-XE Router Configurations] ****
TASK [Gathering Facts] ****
ok: [acme-sea-rtr1]

TASK [napalm_get_facts] ****
ok: [acme-sea-rtr1]

TASK [copy] ****
ok: [acme-sea-rtr1]

PLAY [Backup Cisco NXOS Configurations] ****
TASK [napalm_get_facts] ****
ok: [acme-sea-nxos1]

TASK [copy] ****
changed: [acme-sea-nxos1]

PLAY [Backup Cisco ASA Configurations] ****
TASK [asa_config] ****
ok: [acme-sea-asal]

PLAY RECAP ****
acme-sea-asal      : ok=1    changed=0    unreachable=0    failed=0
acme-sea-nxos1     : ok=2    changed=1    unreachable=0    failed=0
acme-sea-rtr1      : ok=3    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$ cat backup/acme-sea-asal_config.2018-05-22@15\:44\:56
: Saved

:
: Serial Number: 9ABBQGHEEH
: Hardware: ASA5520, 1024 MB RAM, CPU Clarkdale 2300 MHz

```

## What's in a Name?

One irksome thing about the "backup" command is that it does not allow for setting the name of the config file it outputs. While it may seem nit-picky to gripe about that, consistent naming is pretty important! Having the naming consistent will ensure that it is easy to rollback all configurations to the same point, will make configurations easier to search, and will make automation tasks surrounding the configurations simpler.

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ls backup/
acme-sea-asal_config.2018-05-22@15:44:56  acme-sea-nxos1_config  acme-sea-rtr1_config
ignw@ignw-jumphost:~/my_network_as_code$
```

Since we can't modify the "asa\_config" output, let's update the NAPALM task to output configurations with naming that matches the "asa\_config" task.

The first thing we'll notice is the "asa\_config" task output appends "\_config" then the timestamp with an annoying "@" symbol between the date and the time. This isn't a super standard timestamp that we can add, but we can get all of that information easily enough from "ansible\_facts".

Modify the Playbook to set "gather\_facts" to "yes" for the NAPALM tasks.

```
- name: Backup Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: yes
```

Now that facts are gathered for that Play, we can access the fact "ansible\_date\_time" which is a dictionary containing datetime information. Change the destination file name for the NAPALM task to match below:

```
dest: "backup/{{ inventory_hostname }}_config.{{ ansible_date_time.date
}}@{{ ansible_date_time.time }}"
```

One more minor "optimization" we can make would be to move the "asa\_config" to the top of the Playbook -- this ensures we don't need to worry about creating the "backup" directory (since that task will do it for us). Your final "backup" Playbook should look similar to the following:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ cat backup_configurations.yaml
---
- name: Backup Cisco ASA Configurations
  connection: local
  hosts: cisco-asa
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ username }}"
      authorize: yes
      auth_pass: "{{ enable_password }}"
  tags: asa
  tasks:
    - asa_config:
        provider: "{{ creds }}"
        backup: yes
- name: Backup Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: yes
  vars:
    creds:
      hostname: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
      dev_os: "{{ device_type }}"
  tags: ios-xe
  tasks:
    - napalm_get_facts:
        provider: "{{ creds }}"
        filter: "config"
        register: config
    - copy:
        content: "{{ config['ansible_facts']['napalm_config']['running'] }}"
        dest: "backup/{{ inventory_hostname }}_config.{{ ansible_date_time.date }}@{{ ansible_date_time.time }}"
- name: Backup Cisco NXOS Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: yes
  vars:
    creds:
      hostname: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
      dev_os: "{{ device_type }}"
  tags: nxos
  tasks:
    - napalm_get_facts:
        provider: "{{ creds }}"
        filter: "config"
```

Ensure that the "backup" directory is removed, then run the Playbook again to validate it properly creates the directory and our files are named in the same format:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
TASK [asa_config] *****
ok: [acme-sea-asal]

PLAY [Backup Cisco IOS-XE Router Configurations] *****

TASK [Gathering Facts] *****
ok: [acme-sea-rtr1]

TASK [napalm_get_facts] *****
ok: [acme-sea-rtr1]

TASK [copy] *****
changed: [acme-sea-rtr1]

PLAY [Backup Cisco NXOS Configurations] *****

TASK [Gathering Facts] *****
ok: [acme-sea-nxos1]

TASK [napalm_get_facts] *****
ok: [acme-sea-nxos1]

TASK [copy] *****
changed: [acme-sea-nxos1]

PLAY RECAP *****
acme-sea-asal      : ok=1    changed=0    unreachable=0    failed=0
acme-sea-nxos1     : ok=3    changed=1    unreachable=0    failed=0
acme-sea-rtr1      : ok=3    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$ ls backup
acme-sea-asal_config.2018-05-22@16:01:04  acme-sea-rtr1_config.2018-05-22@16:01:05
acme-sea-nxos1_config.2018-05-22@16:01:16
ignw@ignw-jumphost:~/my_network_as_code$
```

## NX-OS Checkpoints

You may be wondering why we did not do the same type of Play for the NXOS device(s) that we did for the IOS-XE device(s). This is due to the way NXOS does config replacement that creates a little bit of a challenge for us. In order to rollback a configuration on NXOS without reloading the host or doing a merge with the appropriate "no" commands (to get rid of extra config) we need to use a special "checkpoint" file. Check out the [NAPALM documentation for a bit more detail](#).

Fortunately for us, there is an "nxos\_rollback" module in Ansible. Unfortunately for us it will only create the rollback on box. There isn't a super simple Ansible NAPALM or Ansible Core way to get the Checkpoint from the host, so for now we'll just backup the config as we did with the IOS-XE devices so we have it for posterity.

Create a new Play to take an NXOS Checkpoint and export the configuration as done with the IOS-XE devices.

```

- name: Backup Cisco NXOS Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: yes
  vars:
    napalm_creds:
      hostname: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
      dev_os: "{{ device_type }}"
    nxos_rollback_creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
      transport: "cli"
  tags: nxos
  tasks:
    - nxos_rollback:
        provider: "{{ nxos_rollback_creds }}"
        checkpoint_file: "my_net_as_code_checkpoint"
    - napalm_get_facts:
        provider: "{{ napalm_creds }}"
        filter: "config"
        register: config
    - copy:
        content: "{{ config['ansible_facts']['napalm_config']['running'] }}"
        dest: "backup/{{ inventory_hostname }}_config.{{ ansible_date_time.date }}@{{ ansible_date_time.time }}"

```

*Note:* NAPALM can indeed handle capturing the checkpoints for us, however that functionality is not currently exposed in Ansible NAPALM.

Run the Playbook again to make sure the NXOS device config is captured, and the checkpoint is created successfully:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
!
wsma agent notify
!
!
endignw@ignw-jumphost:~/my_network_as_code$ vi backup_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook backup_configurations.yaml

PLAY [Backup Cisco IOS-XE Router Configurations] ****
TASK [Gathering Facts] ****
ok: [acme-sea-rtr1]

TASK [napalm_get_facts] ****
ok: [acme-sea-rtr1]

TASK [copy] ****
ok: [acme-sea-rtr1]

PLAY [Backup Cisco NXOS Configurations] ****
TASK [napalm_get_facts] ****
ok: [acme-sea-nxos1]

TASK [copy] ****
changed: [acme-sea-nxos1]

PLAY RECAP ****
acme-sea-nxos1      : ok=2    changed=1      unreachable=0      failed=0
acme-sea-rtr1       : ok=3    changed=0      unreachable=0      failed=0

ignw@ignw-jumphost:~/my_network_as_code$ cat backup/acme-sea-nxos1_config
!Command: show running-config
!Time: Tue May 22 15:37:21 2018

version 7.0(3)I7(3)
hostname acme-sea-nxos1
vdc acme-sea-nxos1 id 1
  limit-resource vlan minimum 16 maximum 4094
  limit-resource vrf minimum 2 maximum 4096
  limit-resource port-channel minimum 0 maximum 511
```

Now would be a good time to add the "backup\_configurations.yaml" file to git and push it to GitHub.

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
acme-sea-rtr1      : ok=3    changed=1    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$ ls backup
acme-sea-asal_config.2018-05-22@16:01:04  acme-sea-rtr1_config.2018-05-22@16:01:05
acme-sea-nxos1_config.2018-05-22@16:01:16
ignw@ignw-jumphost:~/my_network_as_code$ git add backup_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m 'created backup playbook'
[master 3887c2c] created backup playbook
  Committer: ignw <ignw@ignw-jumphost.ignw.io>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

  git config --global --edit

After doing this, you may fix the identity used for this commit with:

  git commit --amend --reset-author

1 file changed, 56 insertions(+)
create mode 100644 backup_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 671 bytes | 671.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:carlniger/my_network_as_code
  1fa008d..3887c2c  master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_network_as_code$
```

## Pushing Rollback Configs Out

With that out of the way, we need to establish a mechanism with which to rollback to our backup configs, once again, NAPALM provides this capability for us, but we will have to work for it a little bit!

The first thing we'll need to understand is that NAPALM requires the "archive" option enabled on IOS/IOS-XE devices. This is a pretty neat feature built into IOS/IOS-XE a very long time ago in the "12.2" days of IOS! Let's update our template to include this configuration set for our "templates/cisco\_ios\_xe\_router.j2" file. We may as well also drop this onto the router directly so we can continue with our testing. Add the following to the end of the template (before the "end" line) and also to the CSR.

```
archive
path bootflash:archive
write-memory
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
{% for interface, data in ethernet_interfaces.items() %}
interface {{ interface }}
{% include 'generic_l3_interface_configs.j2' ignore missing %}
ip address {{ data.ip }} {{ data.mask }}
negotiation auto
no shutdown
{% endfor %}
!
{% for interface, data in loopback_interfaces.items() %}
interface loopback{{ interface }}
ip address {{ data.ip }} {{ data.mask }}
no shutdown
{% endfor %}
!
no ip http server
ip scp server enable
!
ip ssh version 2
!
line vty 0 4
login local
!
ntp master 5
{% for server, data in ntp.items() %}
ntp server {{ server }}{% if data['prefer'] == true %} prefer {% endif %}
{% endfor %}
!
archive
path bootflash:archive
write-memory
!
end
"templates/cisco_ios_xe_router.j2" 54 lines, 1247 characters written
[Running] done
```

## Ansible and Secrets

Our next challenge is that NAPALM and "asa\_config" replace the username Ansible uses to connect to the device with "\*\*\*\*\*". It only does this for the Ansible users, so we will need to replace this in order to have a valid configuration to push back in event of a rollback.

The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~/my\_network\_as\_code". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area displays the following command-line session:

```
ignw@ignw-jumphost:~/my_network_as_code$  
ignw@ignw-jumphost:~/my_network_as_code$  
ignw@ignw-jumphost:~/my_network_as_code$ cat backup/acme-sea-rtr1_config.2018-05-22@18\:18\:51 | grep user  
username ***** privilege 15 password 7 1210021905  
ignw@ignw-jumphost:~/my_network_as_code$
```

**Note:** At this point it is worth noting that NAPALM supports rollback of changes within a script very easily, however as we aren't doing this all at once (running tests between stages, etc.), and we're dealing with other devices NAPALM does not support we'll do this in a bit less optimal fashion.

To replace the username stars with "ignw" we can simply add some tasks to our "backup\_configurations.yaml" Playbook to basically do a find and replace for us. To do this we can use the "replace" (shocking!) module. Add the following task to each of the Plays in your "backup\_configurations.yaml" Playbook.

```
- replace:  
    path: "{{ lookup('pipe', 'ls -1 backup/{{ inventory_hostname }}_config.*') }}"  
    regexp: '^username \*\*\*\*\*\*\*\*\*\*\*\*'   
    replace: "username {{ username }}"
```

The above snippet makes use of the "lookup" functionality of Ansible. In this case we are looking for "acme-sea-rtr1\_config.SOMETHING" -- as we continue integrating this into our Pipeline we should only ever have one copy of the configuration in the "backup" folder, so this should work nicely. Maybe we'll add an "archive" folder later so we can keep X number of configurations too..

With that prepped, delete the "backup" directory and re-execute your Playbook.

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ rm -rf backup
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook backup_configurations.yaml

PLAY [Backup Cisco ASA Configurations] ****
TASK [asa_config] ****
ok: [acme-sea-asal]

TASK [replace] ****
changed: [acme-sea-asal]

PLAY [Backup Cisco IOS-XE Router Configurations] ****
TASK [Gathering Facts] ****
ok: [acme-sea-rtr1]

TASK [napalm_get_facts] ****
ok: [acme-sea-rtr1]

TASK [copy] ****
changed: [acme-sea-rtr1]

TASK [replace] ****
changed: [acme-sea-rtr1]

PLAY [Backup Cisco NXOS Configurations] ****
TASK [Gathering Facts] ****
ok: [acme-sea-nxos1]

TASK [napalm_get_facts] ****
ok: [acme-sea-nxos1]

TASK [copy] ****
changed: [acme-sea-nxos1]

TASK [replace] ****
changed: [acme-sea-nxos1]

PLAY RECAP ****
acme-sea-asal      : ok=2    changed=1    unreachable=0    failed=0
acme-sea-nxos1     : ok=4    changed=2    unreachable=0    failed=0
acme-sea-rtr1      : ok=4    changed=2    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$ 

```

The username should no longer be "\*\*\*\*\*" in the backup files:

```

PLAY RECAP ****
acme-sea-asal      : ok=2    changed=1    unreachable=0    failed=0
acme-sea-nxos1     : ok=4    changed=2    unreachable=0    failed=0
acme-sea-rtr1      : ok=4    changed=2    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$ cat backup/acme-sea-rtr1_config.2018-05-22@18\:33\:43 | grep user
username ignw privilege 15 password 7 1210021905
ignw@ignw-jumphost:~/my_network_as_code$ 

```

We're now ready to attempt to push out one of our backed up configurations. Create yet another Playbook -- call this one "replace\_configurations.yaml". Note that word *-- replace*; we want to fully replace (as opposed to merge as we've been doing previously) the configurations so that we know we get back to the previous state that we captured.

This new Playbook will be fairly similar to our "deploy" Playbook, but instead of setting "replace\_config" to False, we'll switch that to True:

```
---
```

```
- name: Replace Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  vars:
    creds:
      hostname: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
      dev_os: "{{ device_type }}"
  tags: ios-xe
  tasks:
    - napalm_install_config:
        provider: "{{ creds }}"
        config_file: "{{ lookup('pipe', 'ls -1 backup/{{ inventory_hostname }}_config.*') }}"
        commit_changes: True
        replace_config: True
```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ls
ansible.cfg          deploy_configurations.yaml    inventory      replace_configurations.yaml
backup                generate_configurations.yaml Jenkinsfile   requirements.txt
backup_configurations.yaml group_vars           legacy_configs templates
core_module_way.yaml host_vars            README.md     validate_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ vi replace_configurations.yaml
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook replace_configurations.yaml

PLAY [Replace Cisco IOS-XE Router Configurations] ****
TASK [napalm_install_config] ****
fatal: [acme-sea-rtr1]: FAILED! => {"changed": false, "msg": "cannot install config: Candidate config could not be applied\nFailed to apply command Building configuration...\nAborting Rollback.\n\nRollback failed.Reverting back to the original configuration: bootflash:archive-May-22-12-05-32-PDT-4 ...\\n\\nTotal number of passes: 1\\nRollback Done\\n\\nThe original configuration has been successfully restored.\\n"}
PLAY RECAP ****
acme-sea-rtr1 : ok=0    changed=0    unreachable=0    failed=1
ignw@ignw-jumphost:~/my_network_as_code$
```

Looks like we're not *quite* done massaging the configuration file -- take a look at the error message, why do you think the router is balking at this?

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
Building configuration...
Current configuration : 4193 bytes
!
! Last configuration change at 11:41:10 PDT Tue May 22 2018 by *****
! NVRAM config last updated at 11:41:14 PDT Tue May 22 2018 by *****
!
version 16.8
service timestamps debug datetime msec
service timestamps log datetime localtime show-timezone
service password-encryption
platform qfp utilization monitor load 80
no platform punt-keepalive disable-kernel-core
platform console virtual
!
hostname acme-sea-rtr1
!
boot-start-marker
boot-end-marker
!
!
vrf definition management
 rd 1:1
!
address-family ipv4
 route-target export 1:1
 route-target import 1:1
 exit-address-family
!
logging buffered 8192
!
no aaa new-model
clock timezone PST -8 0
clock summer-time PDT recurring
!
!
!
!
!
ip name-server 8.8.8.8 8.8.4.4
ip domain name sea.acme.io
!
```

Pasting in "Building configuration..." into a router prompt would probably not work very well... same goes for "Current configuration...", so looks like we need to slice these few lines out of the configuration. Let's just get this sorted out in a down and dirty fashion -- add the following line to the IOS-XE section of the "backup\_configurations.yaml" Playbook:

```
- shell: sed -i 1,3d "{{ lookup('pipe', 'ls -1 backup/{{ inventory_hostname }}_config.*') }}"
```

Delete the backup directory and re-run the "backup\_configurations.yaml" Playbook:

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
TASK [copy] *****
changed: [acme-sea-rtr1]
TASK [replace] *****
changed: [acme-sea-rtr1]
TASK [shell] *****
[WARNING]: Consider using the replace, lineinfile or template module rather than running sed. If you need to use command because replace, lineinfile or template is insufficient you can add warn=False to this command task or set command_warnings=False in ansible.cfg to get rid of this message.
changed: [acme-sea-rtr1]
PLAY [Backup Cisco NXOS Configurations] *****
TASK [Gathering Facts] *****
ok: [acme-sea-nxos1]
TASK [napalm_get_facts] *****
ok: [acme-sea-nxos1]
TASK [copy] *****
changed: [acme-sea-nxos1]
TASK [replace] *****
changed: [acme-sea-nxos1]
PLAY RECAP *****
acme-sea-asal      : ok=2    changed=1    unreachable=0    failed=0
acme-sea-nxos1     : ok=4    changed=2    unreachable=0    failed=0
acme-sea-rtr1      : ok=5    changed=3    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$ cat backup/acme-sea-rtr1_config.2018-05-22@19\:20\:13
!
! Last configuration change at 12:05:42 PDT Tue May 22 2018 by *****
! NVRAM config last updated at 11:41:14 PDT Tue May 22 2018 by *****
!
version 16.8
service timestamps debug datetime msec
service timestamps log datetime localtime show-timezone
service password-encryption
platform qfp utilization monitor load 80
no platform punt-keepalive disable-kernel-core
platform console virtual
!
```

With that sorted, run the "replace\_configurations.yaml" Playbook again to see if we get better results.

```

endignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook replace_configurations.yaml
PLAY [Replace Cisco IOS-XE Router Configurations] *****
TASK [napalm_install_config] *****
ok: [acme-sea-rtr1]
PLAY RECAP *****
acme-sea-rtr1      : ok=1    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$
```

## NX-OS Checkpoint Rollback

Much better! Time to add in the NX-OS rollback:

```
- name: Replace Cisco NXOS Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: no
  vars:
    nxos_rollback_creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
      transport: "cli"
  tags: nxos
  tasks:
    - nxos_rollback:
        provider: "{{ nxos_rollback_creds }}"
        rollback_to: my_net_as_code_checkpoint
```

This should look awfully similar to creation of the checkpoint, so there shouldn't be much of a surprise here!

Executing your Playbook at this point should result in successfully rolling back configs. Now would be a great time to go add a dummy loopback to the devices, then test your Playbook to ensure it reverts back to the previous config!

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook replace_configurations.yaml
PLAY [Replace Cisco IOS-XE Router Configurations] ****
TASK [napalm_install_config] ****
ok: [acme-sea-rtr1]

PLAY [Replace Cisco NXOS Configurations] ****
TASK [nxos_rollback] ****
changed: [acme-sea-nxos1]

PLAY RECAP ****
acme-sea-nxos1      : ok=1    changed=1    unreachable=0    failed=0
acme-sea-rtr1       : ok=1    changed=0    unreachable=0    failed=0

ignw@ignw-jumphost:~/my_network_as_code$
```

## ASA Checkpoint? No... Config Replace? No..

Finally, we need to replace the configuration on the ASA. This is a bit trickier as the ASA doesn't have a checkpoint or archive feature. It *does* have a "backup" and "restore" feature, however in testing this acts more like a merge (similar to copy start run), which is not quite what we are after. So... we'll have to deal with pushing the backed up config to the ASA and reloading it. Not ideal for certain. It should be possible to do this much more gracefully in an HA pair (install backed up config on standby, reload it, flip it to active, repeat for other side), but this is a lab after all!

The plot thickens a bit in that there isn't a great way to use Ansible (or NAPALM because there is no ASA driver) to get files *to* the ASA. To work around this we'll lightly modify a very great Python script that Kirk Byers (the creator of Netmiko, and contributor to many of the Ansible modules we've been using!) created as a Netmiko example. You can find this script in its original form [here](#).

We don't need to modify this much at all, but we will make a few tweaks. First, we'll modify the script accept a few arguments: IP, username, password, enable password of our ASA, as well as the file that we are sending to it. Copy the script from GitHub, and create a new file in your repository called "asa\_scp.py". Modify the top portion of the script to match:

```

#!/usr/bin/python3
"""Script to upgrade a Cisco ASA."""
import sys
from netmiko import ConnectHandler, FileTransfer

ASA_IP = sys.argv[1]
ASA_USER = sys.argv[2]
ASA_PASSWORD = sys.argv[3]
ASA_ENABLE = sys.argv[4]
BACKUP_FILE = sys.argv[5]

def asa_scp_handler(ssh_conn, cmd='ssh copy enable', mode='enable'):
    """Enable/disable SCP on Cisco ASA."""
    if mode == 'disable':
        cmd = 'no ' + cmd
    return ssh_conn.send_config_set([cmd])

def main():
    net_device = {
        'device_type': 'cisco_asa',
        'ip': ASA_IP,
        'username': ASA_USER,
        'password': ASA_USER,
        'secret': ASA_ENABLE,
        'port': 22,
    }

```

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
#!/usr/bin/python3
"""Script to upgrade a Cisco ASA."""
import sys
from netmiko import ConnectHandler, FileTransfer

ASA_IP = sys.argv[1]
ASA_USER = sys.argv[2]
ASA_PASSWORD = sys.argv[3]
ASA_ENABLE = sys.argv[4]
BACKUP_FILE = sys.argv[5]

def asa_scp_handler(ssh_conn, cmd='ssh scopy enable', mode='enable'):
    """Enable/disable SCP on Cisco ASA."""
    if mode == 'disable':
        cmd = 'no ' + cmd
    return ssh_conn.send_config_set([cmd])

def main():
    net_device = {
        'device_type': 'cisco_asa',
        'ip': ASA_IP,
        'username': ASA_USER,
        'password': ASA_PASSWORD,
        'secret': ASA_ENABLE,
        'port': 22,
    }

    ssh_conn = ConnectHandler(**net_device)

    dest_file_system = 'disk0:'
    source_file = BACKUP_FILE
    dest_file = 'backup.cfg'

    with FileTransfer(ssh_conn, source_file=source_file, dest_file=dest_file,
                      file_system=dest_file_system) as scp_transfer:
        if scp_transfer.check_file_exists():
            ssh_conn.send_command(f'delete /noconfirm {dest_file_system}:{dest_file}')
"asa_scp.py" 58 lines, 1514 characters

```

This will allow us to have Ansible pass appropriate arguments, and get our "net\_device" (dictionary with access info for our ASA) properly setup. We've also removed a few unnecessary things to keep it as simple as possible (imports, user prompts, and the like).

Next, we can set the file information for our use case:

```

ssh_conn = ConnectHandler(**net_device)

dest_file_system = 'disk0:'
source_file = BACKUP_FILE
dest_file = 'backup.cfg'

```

We can then modify the main chunk of logic to suite our needs:

```

with FileTransfer(ssh_conn, source_file=source_file, dest_file=dest_file,
                  file_system=dest_file_system) as scp_transfer:

    if scp_transfer.check_file_exists():
        ssh_conn.send_command(f'delete /noconfirm {dest_file_system}:{dest_file}')

    if not scp_transfer.verify_space_available():
        raise ValueError("Insufficient space available on remote device")

    asa_scp_handler(ssh_conn, mode='enable')

    scp_transfer.transfer_file()

    asa_scp_handler(ssh_conn, mode='disable')

    if not scp_transfer.verify_file():
        raise ValueError("MD5 failure between source and destination files"
)
)

if __name__ == "__main__":
    main()

```

In the above section we've removed all of the printing, added a very simple block to delete the file we're trying to transfer if it exists, and modified the exception handling just a tick to meet our needs.

Your end result should look like the below screen shot:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ASA_IP = sys.argv[1]
ASA_USER = sys.argv[2]
ASA_PASSWORD = sys.argv[3]
ASA_ENABLE = sys.argv[4]
BACKUP_FILE = sys.argv[5]

def asa_scp_handler(ssh_conn, cmd='ssh copy enable', mode='enable'):
    """Enable/disable SCP on Cisco ASA."""
    if mode == 'disable':
        cmd = 'no ' + cmd
    return ssh_conn.send_config_set([cmd])

def main():
    net_device = {
        'device_type': 'cisco_asa',
        'ip': ASA_IP,
        'username': ASA_USER,
        'password': ASA_PASSWORD,
        'secret': ASA_ENABLE,
        'port': 22,
    }
    ssh_conn = ConnectHandler(**net_device)

    dest_file_system = 'disk0:'
    source_file = BACKUP_FILE
    dest_file = 'backup.cfg'

    with FileTransfer(ssh_conn, source_file=source_file, dest_file=dest_file,
                      file_system=dest_file_system) as scp_transfer:
        if scp_transfer.check_file_exists():
            ssh_conn.send_command(f'delete /noconfirm {dest_file_system}:{dest_file}')
        if not scp_transfer.verify_space_available():
            raise ValueError("Insufficient space available on remote device")
        asa_scp_handler(ssh_conn, mode='enable')
        scp_transfer.transfer_file()
        asa_scp_handler(ssh_conn, mode='disable')

    if not scp_transfer.verify_file():
        raise ValueError("MD5 failure between source and destination files")

if __name__ == "__main__":
    main()
```

Ensure that your new script is executable:

```
chmod +x asa_scp.py
```

We can now add a Play to execute our script, passing in the appropriate variables much the same way we've done before:

```

- name: Replace Cisco ASA Configurations
  connection: local
  hosts: cisco-asa
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
      authorize: yes
      auth_pass: "{{ enable_password }}"
  tags: asa
  tasks:
    - shell: asa_scp.py "{{ ansible_host }} {{ username }} {{ password }} {{ enable_password }} {{ lookup('pipe', 'ls -1 backup/{{ inventory_hostname }}_config.*') }}"

```

It may not be the most beautiful thing, but it does work!

```

ignw@ignw-jumphost:~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook replace_configurations.yaml --tags asa
PLAY [Replace Cisco IOS-XE Router Configurations] ****
PLAY [Replace Cisco NXOS Configurations] ****
PLAY [Replace Cisco ASA Configurations] ****
TASK [script] ****
changed: [acme-sea-asa1]
PLAY RECAP ****
acme-sea-asa1 : ok=1    changed=1    unreachable=0    failed=0
ignw@ignw-jumphost:~/my_network_as_code$
```

Next we need to set our startup config to the file we just loaded up. This is a simple one-liner with the Ansible "asa\_config" module. Add the following task to your "replace\_configurations.yaml" Playbook:

```

- asa_config:
    provider: "{{ creds }}"
    lines:
      - copy /noconfirm disk0:/backup.cfg startup-config

```

Note that we can use the "/noconfirm" options to make sure we don't have to deal with any kind of prompts.

Finally, we need to reload the ASA. This can be done with the "asa\_command" module:

```

- asa_command:
  provider: "{{ creds }}"
  commands:
    - reload noconfirm

```

Execute your Playbook:

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook replace_configurations.yaml --tags asa
PLAY [Replace Cisco IOS-XE Router Configurations] ****
PLAY [Replace Cisco NXOS Configurations] ****
PLAY [Replace Cisco ASA Configurations] ****
TASK [script] ****
changed: [acme-sea-asal]
TASK [asa_config] ****
changed: [acme-sea-asal]
TASK [asa_command] ****
ok: [acme-sea-asal]
PLAY RECAP ****
acme-sea-asal : ok=3    changed=2    unreachable=0    failed=0
ignw@ignw-jumphost:~/my_network_as_code$ ping 10.0.0.8
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
64 bytes from 10.0.0.8: icmp_seq=68 ttl=254 time=19193 ms
64 bytes from 10.0.0.8: icmp_seq=69 ttl=254 time=18169 ms
64 bytes from 10.0.0.8: icmp_seq=70 ttl=254 time=17145 ms
64 bytes from 10.0.0.8: icmp_seq=71 ttl=254 time=16121 ms
64 bytes from 10.0.0.8: icmp_seq=72 ttl=254 time=15097 ms
64 bytes from 10.0.0.8: icmp_seq=73 ttl=254 time=14073 ms
64 bytes from 10.0.0.8: icmp_seq=74 ttl=254 time=13049 ms
64 bytes from 10.0.0.8: icmp_seq=75 ttl=254 time=12025 ms
64 bytes from 10.0.0.8: icmp_seq=76 ttl=254 time=11001 ms
64 bytes from 10.0.0.8: icmp_seq=77 ttl=254 time=9977 ms
64 bytes from 10.0.0.8: icmp_seq=78 ttl=254 time=8953 ms
64 bytes from 10.0.0.8: icmp_seq=79 ttl=254 time=7929 ms
64 bytes from 10.0.0.8: icmp_seq=80 ttl=254 time=6905 ms
64 bytes from 10.0.0.8: icmp_seq=81 ttl=254 time=5881 ms
64 bytes from 10.0.0.8: icmp_seq=82 ttl=254 time=4857 ms
64 bytes from 10.0.0.8: icmp_seq=83 ttl=254 time=3833 ms
64 bytes from 10.0.0.8: icmp_seq=84 ttl=254 time=2809 ms
64 bytes from 10.0.0.8: icmp_seq=85 ttl=254 time=1785 ms
64 bytes from 10.0.0.8: icmp_seq=86 ttl=254 time=761 ms
64 bytes from 10.0.0.8: icmp_seq=87 ttl=254 time=1.26 ms
64 bytes from 10.0.0.8: icmp_seq=88 ttl=254 time=0.995 ms
64 bytes from 10.0.0.8: icmp_seq=89 ttl=254 time=0.745 ms
^C
--- 10.0.0.8 ping statistics ---
89 packets transmitted, 22 received, 75% packet loss, time 90043ms
rtt min/avg/max/mdev = 0.745/8616.938/19193.352/6236.092 ms, pipe 19
ignw@ignw-jumphost:~/my_network_as_code$ 

```

## Push to Git

This would be a great time to update GitHub with our edits. We'll need to add the "asa\_scp.py" file, as well as the "replace\_configurations.yaml" file. We can also throw in a "git add -u" just in case we changed some other stuff!

```
git add asa_scp.py  
git add replace_configurations.yaml  
git add -u  
git commit -m 'config rollbacks'  
git push -u origin master
```

```
ignw@ignw-jumphost: ~/my_network_as_code  
File Edit View Search Terminal Help  
ignw@ignw-jumphost:~/my_network_as_code$ git add asa_scp.py  
ignw@ignw-jumphost:~/my_network_as_code$ git add replace_configurations.yaml  
ignw@ignw-jumphost:~/my_network_as_code$ git add -u  
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m 'config rollbacks'  
[master b93b149] config rollbacks  
Committer: ignw <ignw@ignw-jumphost.ignw.io>  
Your name and email address were configured automatically based  
on your username and hostname. Please check that they are accurate.  
You can suppress this message by setting them explicitly. Run the  
following command and follow the instructions in your editor to edit  
your configuration file:  
git config --global --edit  
After doing this, you may fix the identity used for this commit with:  
git commit --amend --reset-author  
4 files changed, 141 insertions(+), 4 deletions(-)  
create mode 100755 asa_scp.py  
create mode 100644 replace_configurations.yaml  
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master  
Counting objects: 7, done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (7/7), done.  
Writing objects: 100% (7/7), 1.76 KiB | 1.76 MiB/s, done.  
Total 7 (delta 5), reused 0 (delta 0)  
remote: Resolving deltas: 100% (5/5), completed with 4 local objects.  
To github.com:carlniger/my_network_as_code  
 3887c2c..b93b149 master -> master  
Branch master set up to track remote branch master from origin.  
ignw@ignw-jumphost:~/my_network_as_code$ █
```

## When do we Rollback?

We now need to determine how we can rollback our configurations *if* our functional testing fails. Lucky for us Jenkins supports this type of behavior -- we can wrap one of our shell commands in a function and return the results of that function to a variable. At that point we can have an if clause that checks the value of our returned status -- if its "bad" we can rollback the configurations with our nifty new rollback Playbook.

First things first, add a "Backup Dev Configurations" stage to your Jenkinsfile before the "Deploy Configurations to Dev" stage:

```
stage ('Backup Dev Configurations') {
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}
```

Next, edit the "Functional/Integration Testing" stage of your Jenkinsfile to wrap the validate\_configurations Playbook in a function, returning the result:

```
stage ('Functional/Integration Testing') {
    def r = sh(script: 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"', returnStatus: true)
```

Great, now we need a conditional that can run our rollback Playbook and fail the build *if* our testing failed. Anything greater than zero indicates a failed script, so we can key off of that:

```
if(r > 0) {
    sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
    currentBuild.result = 'ABORTED'
    error('Integration testing FAILED')
}
```

The Jenkinsfile should look like the screen shot below at this point:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
}

stage ('Backup Dev Configurations') {
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Deploy Configurations to Dev') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Functional/Integration Testing') {
    def r = sh(script: 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"', returnStatus: true)
    if(r > 0) {
        sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
        currentBuild.result = 'ABORTED'
        error('Integration testing FAILED')
    }
}

stage ('Promote Configurations to Production') {
    // Ping stuff and make sure we didn't blow up dev!
}

stage ('Production Functional/Integration Testing') {
    // Ping stuff and make sure we didn't blow up prod!
}

}

~
"Jenkinsfile" 58 lines, 2638 characters written
```

We also need to run our "unit tests" on the newly created replace\_configurations Playbook. Add the following to the "Unit Testing" stage:

```
sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
```

Make sure you've got syntax checking set for all four of your Playbooks.

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
}

stage ('Validate Generate Configurations Playbook') {
    sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
}

stage ('Render Configurations') {
    sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Unit Testing') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    sh 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
}

stage ('Backup Dev Configurations') {
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Deploy Configurations to Dev') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

```

## Environment Updates

Our Playbooks should now all be tested (syntax-check at the least), our Pipeline is setup so that if our integration testing fails a rollback should occur and our build will be marked as failed. We're *almost* ready to test, but we need to address three more issues: 1) we need to update requirements.txt to include netmiko (for our asa\_scp.py file), 2) we need to update the shebang on the Python script to utilize the venv that will be created in our Jenkins workspace, and 3) we need to ensure that our Python script is an executable file for being ran on our Jenkins host.

The first is obviously very straightforward:

```
echo netmiko >> requirements.txt
```

The second we can handle with a bit of down and dirty sed. This sed stuff is getting out of hand though, so think about ways we can clean this up a bit later (there are many better ways to do this type of thing!). Add the following line to the "Setup Environment" stage of your Jenkinsfile:

```
sh '''sed -i -e 's/\\usr\\/bin\\/python3\\/var\\/lib\\/jenkins\\/jobs\\
\\/my_network_as_code\\/workspace\\/jenkins_build\\/bin\\/python/g' asa_scp.py'''
```

The third, we can add a simple task to our ASA Rollback Play in our "replace\_configurations.yaml" Playbook:

```
- name: Make Script Executable
  become: true
  file:
    path: "{{ playbook_dir }}/asa_scp.py"
    mode: "+x"
```

Ensure that the above Task is before any of the other ASA Rollback Tasks.

Once again it is time to push our updates to GitHub.

```
git add -u
git commit -m 'updated Jenkinsfile'
git push -u origin master
```

Hop back over to Jenkins run a build, and then observe the console output.



## Console Output

```
Started by user ignw
Obtained Jenkinsfile from git https://github.com/carlniger/my\_network\_as\_code
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/jobs/my_network_as_code/workspace
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Checkout Repository)
[Pipeline] deleteDir
[Pipeline] checkout
Cloning the remote Git repository
Cloning repository https://github.com/carlniger/my\_network\_as\_code
> git init /var/lib/jenkins/jobs/my_network_as_code/workspace # timeout=10
Fetching upstream changes from https://github.com/carlniger/my\_network\_as\_code
> git --version # timeout=10
using GIT_ASKPASS to set credentials carl
> git fetch --tags --progress https://github.com/carlniger/my\_network\_as\_code +refs/heads/*:refs/remotes/origin/*
> git config remote.origin.url https://github.com/carlniger/my\_network\_as\_code # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/carlniger/my\_network\_as\_code # timeout=10
Fetching upstream changes from https://github.com/carlniger/my\_network\_as\_code
using GIT_ASKPASS to set credentials carl
> git fetch --tags --progress https://github.com/carlniger/my\_network\_as\_code +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 6a2ec16d210c4436506bdf33e9199b156af91929 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 6a2ec16d210c4436506bdf33e9199b156af91929
Commit message: "."
> git rev-list --no-walk 593994cf614c25380b5b9bb0b8af895b2d7acf91 # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
```

At this point the build *should* successfully complete.

```

TASK [nxos_ping] ****
ok: [acme-sea-nxos1]

PLAY [Validate "Core" & "Internet" Reachability from Firewall] ****

TASK [asa_command] ****
ok: [acme-sea-asal] => (item=8.8.8)
ok: [acme-sea-asal] => (item=10.255.255.2)
ok: [acme-sea-asal] => (item=10.255.254.1)

PLAY [Validate "Edge" Reachability from Edge Router] ****

TASK [Test Success Pings] ****
ok: [acme-sea-rtr1] => (item=203.0.113.2)

TASK [Test Fail Pings] ****
ok: [acme-sea-rtr1] => (item=10.255.255.2)

PLAY RECAP ****
acme-sea-asal : ok=1    changed=0    unreachable=0   failed=0
acme-sea-nxos1: ok=1    changed=0    unreachable=0   failed=0
acme-sea-rtr1 : ok=2    changed=0    unreachable=0   failed=0

[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Promote Configurations to Production)
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Production Functional/Integration Testing)
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

## Testing Rollbacks

This is of course great, but we need someway to test our rollback functionality. We can do this easily enough by adding an IP address to one of our tests that will not respond, thus causing our Playbook to fail, causing that shell script to fail, ultimately causing Jenkins to fire off the rollback functionality.

Add the following line to the "validate\_configurations.yaml" file:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
---
- name: Validate "Core" to "Internet" Reachability
  connection: local
  hosts: acme-sea-nxos1
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
  tasks:
    - nxos_ping:
        provider: "{{ creds }}"
        dest: 8.8.8.8
        vrf: default
- name: Validate "Core" & "Internet" Reachability from Firewall
  connection: local
  hosts: acme-sea-asa1
  gather_facts: no
  vars:
    creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ username }}"
      authorize: yes
      auth_pass: "{{ enable_password }}"
    test_ips:
      - 8.8.8.8
      - 10.255.255.2
      - 10.255.254.1
      - 1.2.3.4
  tasks:
    - asa_command:
```

> git init /var

And once again, commit your code, and re-run your build.

my\_network\_as\_code #45 Console [Jenkins] - Mozilla Firefox

my\_network\_as\_code #45

10.10.0.253:8080/job/my\_network\_as\_code/45/console

```
[Pipeline] sh
[workspace] Running shell script
+ ansible-playbook validate_configurations.yaml -e ansible_python_interpreter=jenkins_build/bin/python
PLAY [Validate "Core" to "Internet" Reachability] *****

TASK [nxos_ping] *****
ok: [acme-sea-nxos1]

PLAY [Validate "Core" & "Internet" Reachability from Firewall] *****

TASK [asa_command] *****
ok: [acme-sea-asal] => (item=8.8.8)
ok: [acme-sea-asal] => (item=10.255.255.2)
ok: [acme-sea-asal] => (item=10.255.254.1)
failed: [acme-sea-asal] (item=1.2.3.4) => {"changed": false, "failed_when_result": true, "item": "1.2.3.4", "stdout": ["None"], "stdout_lines": [{"None"}]}

PLAY RECAP *****
acme-sea-asal      : ok=0    changed=0    unreachable=0    failed=1
acme-sea-nxos1     : ok=1    changed=0    unreachable=0    failed=0

[Pipeline] sh
[workspace] Running shell script
+ ansible-playbook replace_configurations.yaml -e ansible_python_interpreter=jenkins_build/bin/python
PLAY [Replace Cisco IOS-XE Router Configurations] *****

TASK [napalm_install_config] *****
changed: [acme-sea-rtr1]

PLAY [Replace Cisco NXOS Configurations] *****

TASK [nxos_rollback] *****
```

Page generated: May 25, 2018 5:33:23 PM UTC [REST API](#) Jenkins ver. 2.107.2

This looks promising -- we can see that our validation stage has clearly failed to ping the 1.2.3.4 address, and the very next thing that happens is that the "replace\_configurations" Playbook is executed!

Unfortunately, we seem to have ran into another issue:

my\_network\_as\_code #45 Console [Jenkins] - Mozilla Firefox

my\_network\_as\_code +  
10.10.0.253:8080/job/my\_network\_as\_code/45/console  
Jenkins > my\_network\_as\_code > #45

```
[workspace] Running shell script
+ ansible-playbook replace_configurations.yaml -e ansible_python_interpreter=jenkins_build/bin/python
PLAY [Replace Cisco IOS-XE Router Configurations] ****
TASK [napalm_install_config] ****
changed: [acme-sea-rtr1]

PLAY [Replace Cisco NXOS Configurations] ****
TASK [nxos_rollback] ****
changed: [acme-sea-nxos1]

PLAY [Replace Cisco ASA Configurations] ****
TASK [shell] ****
fatal: [acme-sea-asal]: FAILED! => {"changed": true, "cmd": "asa_scp.py \"10.0.0.8 ignw ignw ignw backup/acme-sea-asal_config.2018-05-25@17:34:42\"", "delta": "0:00:00.003546", "end": "2018-05-25 17:39:23.832683", "msg": "non-zero return code", "rc": 127, "start": "2018-05-25 17:39:23.829137", "stderr": "/bin/sh: 1: asa_scp.py: not found", "stderr_lines": ["/bin/sh: 1: asa_scp.py: not found"], "stdout": "", "stdout_lines": []}

PLAY RECAP ****
acme-sea-asal : ok=0    changed=0    unreachable=0    failed=1
acme-sea-nxos1 : ok=1    changed=1    unreachable=0    failed=0
acme-sea-rtr1 : ok=1    changed=1    unreachable=0    failed=0

[Pipeline]
[Pipeline] // stage
[Pipeline]
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 2
Finished: FAILURE
```

Page generated: May 25, 2018 5:33:23 PM UTC [REST API](#) Jenkins ver. 2.107.2

Unfortunately Jenkins and/or Ansible are having a hard time finding our Python script to move the backup file over to the ASA. However we address this we will want to be sure that our Playbook can execute locally (on the Jumphost) as well as on the Jenkins server or a future container, etc..

Ansible has a pretty simple built in variable aptly named "playbook\_dir" which is the full path to where the Playbook lives. Let's add this to the shell command executing our script -- this way no matter *where* our script is, as long as it is in the directory with our Playbook, we should be good to go:

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
nxos_rollback_creds:
  host: "{{ ansible_host }}"
  username: "{{ username }}"
  password: "{{ password }}"
  transport: "cli"
tags: nxos
tasks:
  - nxos_rollback:
      provider: "{{ nxos_rollback_creds }}"
      rollback_to: my_net_as_code_checkpoint
- name: Replace Cisco ASA Configurations
  connection: local
  hosts: cisco-asa
  gather_facts: no
vars:
  creds:
    host: "{{ ansible_host }}"
    username: "{{ username }}"
    password: "{{ password }}"
    authorize: yes
    auth_pass: "{{ enable_password }}"
tags: asa
tasks:
  - shell: "{{ playbook_dir }}/asa_scp.py {{ ansible_host }} {{ username }} {{ password }} {{ enable_password }} {{ lookup('pipe', 'ls -1 backup/{{ inventory_hostname }}_config.*') }}"
    asa_config:
      provider: "{{ creds }}"
      lines:
        - copy /noconfirm disk0:/backup.cfg startup-config
  - asa_command:
      provider: "{{ creds }}"
      commands:
        - reload noconfirm
"replace_configurations.yaml" 55 lines, 1622 characters written
```

At this point you know the drill -- push your configs to GitHub and re-run your build.

my\_network\_as\_code #46 Console [Jenkins] - Mozilla Firefox

my\_network\_as\_code x +  
10.10.0.253:8080/job/my\_network\_as\_code/46/console

Jenkins > my\_network\_as\_code > #46

```
TASK [napalm_install_config] ****
changed: [acme-sea-rtr1]

PLAY [Replace Cisco NXOS Configurations] ****

TASK [nxos_rollback] ****
changed: [acme-sea-nxos1]

PLAY [Replace Cisco ASA Configurations] ****

TASK [shell] ****
changed: [acme-sea-asal]

TASK [asa_config] ****
changed: [acme-sea-asal]

TASK [asa_command] ****
ok: [acme-sea-asal]

PLAY RECAP ****
acme-sea-asal : ok=3    changed=2    unreachable=0    failed=0
acme-sea-nxos1 : ok=1    changed=1    unreachable=0    failed=0
acme-sea-rtr1 : ok=1    changed=1    unreachable=0    failed=0

[Pipeline] error
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: Integration testing FAILED
Finished: ABORTED
```

Page generated: May 25, 2018 5:46:24 PM UTC [REST API](#) Jenkins ver. 2.107.2

Excellent! Our script ran, all of the rollbacks completed successfully and then our build failed, just like we wanted.

Obviously there is a ton of room for improvement on our Pipeline as a whole, but even in it's rag-tag state, this is pretty powerful stuff!

Delete the line for the 1.2.3.4 IP address and commit your code one last time. Next we'll work on promoting our builds to production if everything goes according to plan!

# Promoting to Production

## Overview and Objectives

In this lab you will "promote" configurations into the production environment if all checks up to that point pass. You will also handle archiving configurations before any changes are made -- just in case!

Objectives:

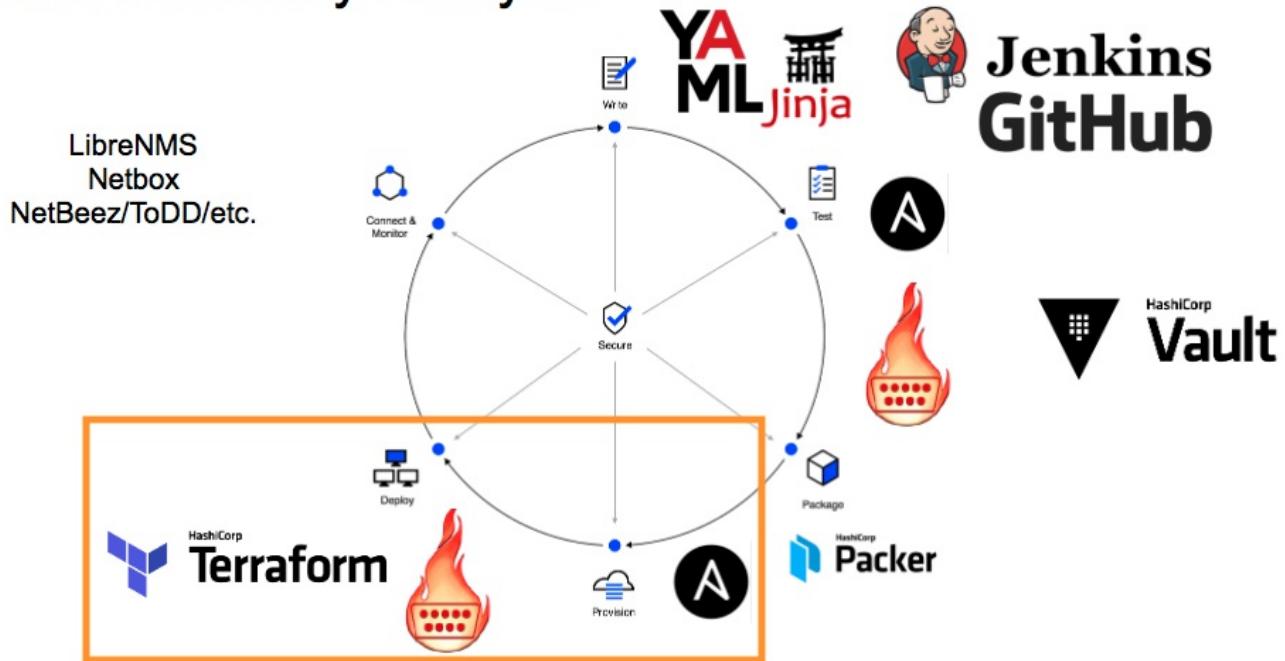
- Tweak Ansible a bit to support a second inventory file
- Add tasks to archive and export configurations for posterity
- Modify the Pipeline to deploy configurations to Production

## Network Delivery Lifecycle Overview

While the HashiCorp Application Lifecycle doesn't at first glance appear to map tightly to a "promotion" phase, it actually fits in perfectly! The HashiCorp Application Lifecycle would, just as we are doing, generally be "wrapped up" with a CI/CD Pipeline. Deployments that pass testing scrutiny would then be promoted to production in the same way that we will be doing.

Once again this maps most closely to the "deploy" phase:

## Network Delivery Lifecycle



## Promotion Time!

Now that our Pipeline supports rollback and some basic testing, we can look into how we can push the configurations into production. In general it is a good idea to keep dev and prod as identical as possible. In this lab everything but management IP addressing is identical, because of this we've simply left management access/IP information out of our "IAC".

In real world environments you'll need to decide how you want to handle management information -- it may be nice to be able to enforce management configurations programmatically, however with great power... you get the idea!

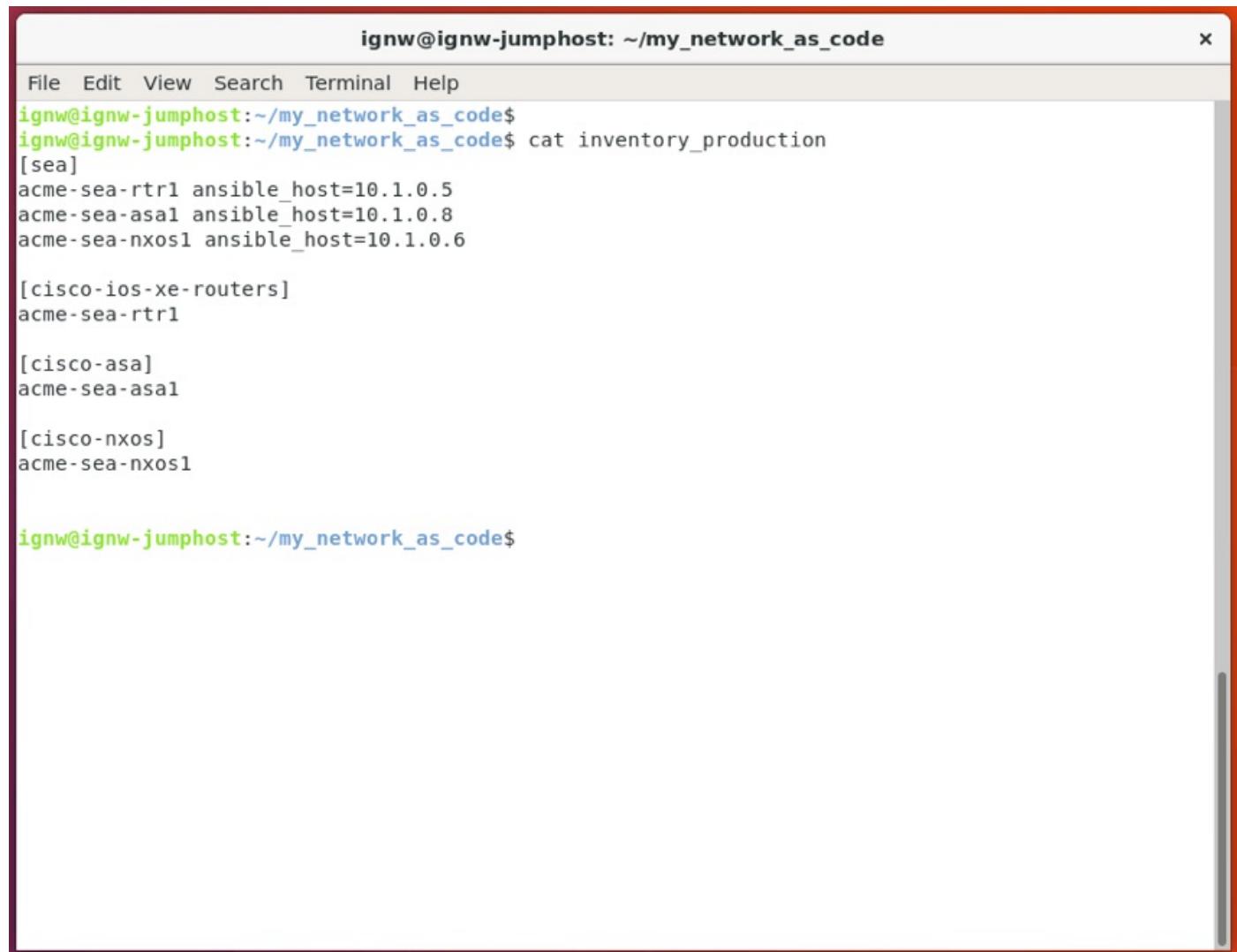
Since we are using Ansible, have identical configs, and are not worrying about management information, we can pretty much just clone our inventory file and tweak the IP addressing to match the production IPs. We'll also need to ensure that our Pipeline is passing the proper inventory file to the Playbooks at the appropriate stage.

## Inventory Part 2

Create a copy of the inventory file called "inventory\_production".

```
cp inventory inventory_production
```

Edit the inventory file to modify the second octet of all of the ansible\_host IP addresses to "1".



The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~/my\_network\_as\_code". The window contains the following command and its output:

```
ignw@ignw-jumphost:~/my_network_as_code$ cat inventory_production
[sea]
acme-sea-rtr1 ansible_host=10.1.0.5
acme-sea-asal ansible_host=10.1.0.8
acme-sea-nxos1 ansible_host=10.1.0.6

[cisco-ios-xe-routers]
acme-sea-rtr1

[cisco-asa]
acme-sea-asal

[cisco-nxos]
acme-sea-nxos1

ignw@ignw-jumphost:~/my_network_as_code$
```

# Updating the Jenkinsfile

We now need to turn our attention to our Jenkinsfile. Our Pipeline so far looks like this:

1. Clean up workspace, checkout code from SCM
2. Setup build environment (venv, requirements, modify paths where required)
3. Validate "generate\_configurations" Playbook
4. Run "generate\_configurations" Playbook (dev)
5. Validate the "deploy", "validate", "backup", and "replace" Playbooks
6. Run "backup\_configurations" Playbook (dev)
7. Run "deploy\_configurations" Playbook (dev)
8. Run "validate\_configurations" Playbook (dev)
  - IF validation fails, run "replace\_configurations" Playbook (dev)
9. ???
10. Profit

Since we are able to reuse all of the Playbooks we don't need to worry about any of our "syntax-check" tasks, but we do need to run through steps 6-8 for production. We already sort of stubbed out those stages in our Jenkinsfile, but now we will formalize them and get them ready to run.

Delete our previous stubbed out stages, and replace them with a copy of the backup, deploy and test stages, change these to be named for "prod".

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
}

stage ('Validate Generate Configurations Playbook') {
    sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
}

stage ('Render Configurations') {
    sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Unit Testing') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    sh 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
}

stage ('Backup Dev Configurations') {
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Deploy Configurations to Dev') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Functional/Integration Testing Dev') {
    def r = sh(script: 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"', returnStatus: true)
    if(r > 0) {
        sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
        currentBuild.result = 'ABORTED'
        error('Integration testing FAILED')
    }
}

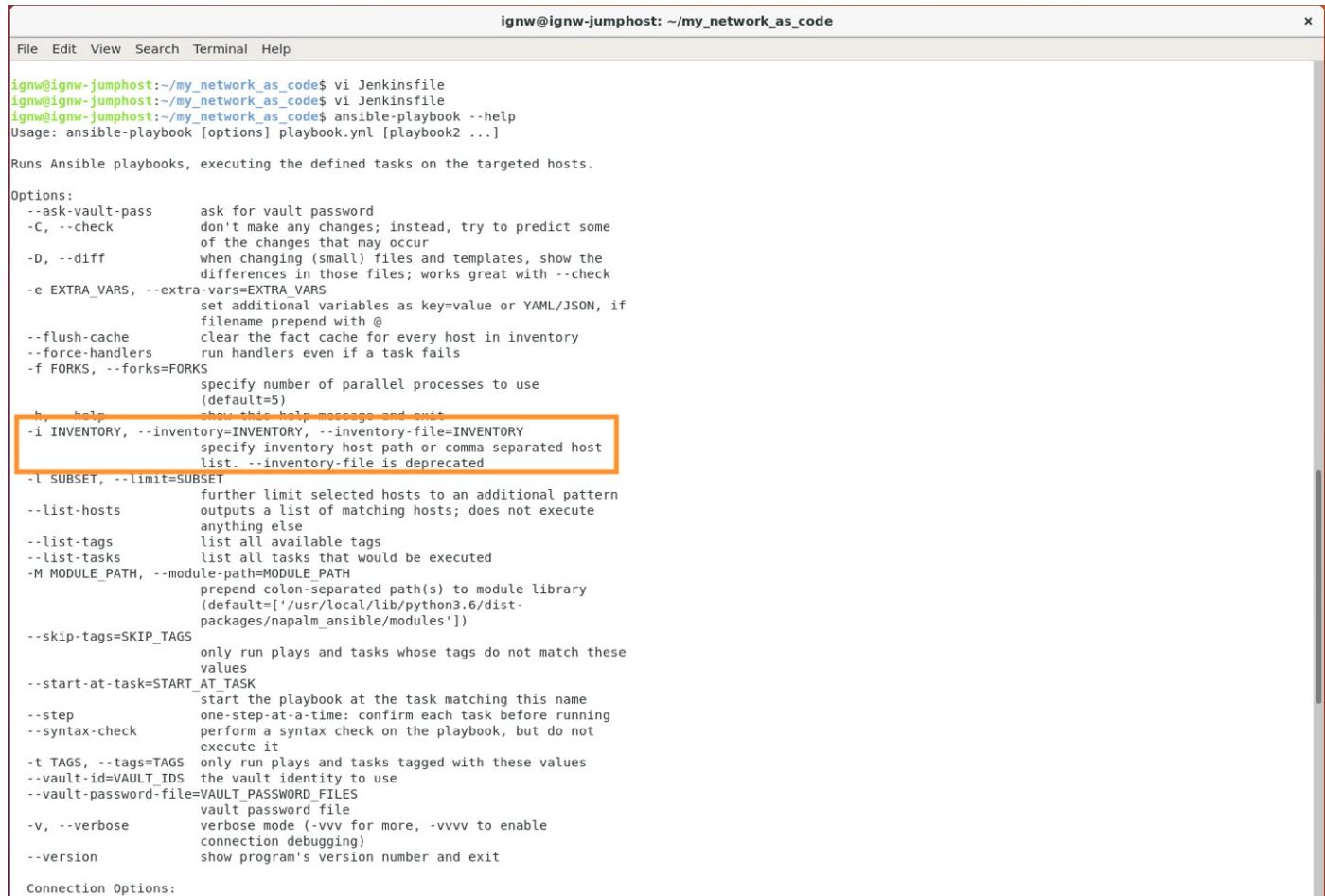
stage ('Backup Prod Configurations') {
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Deploy Configurations to Prod') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Functional/Integration Testing Prod') {
    def r = sh(script: 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"', returnStatus: true)
    if(r > 0) {
        sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
        currentBuild.result = 'ABORTED'
        error('Integration testing FAILED')
    }
}
```

Easy enough! Next we need to tell Ansible to use the inventory file we created for our production hosts. You can check out the available arguments we can pass when executing a Playbook by checking the help page:

"ansible-playbook --help".



```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ vi Jenkinsfile
ignw@ignw-jumphost:~/my_network_as_code$ vi Jenkinsfile
ignw@ignw-jumphost:~/my_network_as_code$ ansible-playbook --help
Usage: ansible-playbook [options] playbook.yml [playbook2 ...]

Runs Ansible playbooks, executing the defined tasks on the targeted hosts.

Options:
  --ask-vault-pass      ask for vault password
  -C, --check           don't make any changes; instead, try to predict some
                        of the changes that may occur
  -D, --diff            when changing (small) files and templates, show the
                        differences in those files; works great with --check
  -e EXTRA_VARS, --extra-vars=EXTRA_VARS
                        set additional variables as key=value or YAML/JSON, if
                        filename prepend with @
  --flush-cache         clear the fact cache for every host in inventory
  --force-handlers      run handlers even if a task fails
  -f FORKS, --forks=FORKS
                        specify number of parallel processes to use
                        (default=5)
  h, help               show this help message and exit
  -i INVENTORY, --inventory=INVENTORY, --inventory-file=INVENTORY
                        specify inventory host path or comma separated host
                        list. --inventory-file is deprecated
  -l SUBSET, --limit=SUBSET
                        further limit selected hosts to an additional pattern
                        outputs a list of matching hosts; does not execute
                        anything else
  --list-hosts          list all available hosts
  --list-tags           list all available tags
  --list-tasks          list all tasks that would be executed
  -M MODULE_PATH, --module-path=MODULE_PATH
                        prepend colon-separated path(s) to module library
                        (default=['/usr/local/lib/python3.6/dist-
                        packages/napalm_ansible/modules'])
  --skip-tags=SKIP_TAGS
                        only run plays and tasks whose tags do not match these
                        values
  --start-at-task=START_AT_TASK
                        start the playbook at the task matching this name
  --step                one-step-at-a-time: confirm each task before running
  --syntax-check        perform a syntax check on the playbook, but do not
                        execute it
  -t TAGS, --tags=TAGS
                        only run plays and tasks tagged with these values
  --vault-id=VAULT_IDS
                        the vault identity to use
  --vault-password-file=VAULT_PASSWORD_FILES
                        vault password file
  -v, --verbose         verbose mode (-vv for more, -vvvv to enable
                        connection debugging)
  --version             show program's version number and exit

Connection Options:
```

We can simply pass in our preferred inventory with the -i switch as shown above. This will override our "ansible.cfg" which points to the dev inventory file.

Update the Jenkinsfile to pass in the new inventory file for the Production stages:

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help

stage ('Validate Generate Configurations Playbook') {
    sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
}

stage ('Render Configurations') {
    sh 'ansible-playbook generate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Unit Testing') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    sh 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
    sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
}

stage ('Backup Dev Configurations') {
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Deploy Configurations to Dev') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Functional/Integration Testing Dev') {
    def r = sh(script: 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"', returnStatus: true)
    if(r > 0) {
        sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
        currentBuild.result = 'ABORTED'
        error('Integration testing FAILED')
    }
}

stage ('Backup Prod Configurations') {
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production'
}

stage ('Deploy Configurations to Prod') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production'
}

stage ('Functional/Integration Testing Prod') {
    def r = sh(script: 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production', returnStatus: true)
    if(r > 0) {
        sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production'
        currentBuild.result = 'ABORTED'
        error('Integration testing FAILED')
    }
}
ignw@ignw-jumphost:~/my_network_as_code$ 

```

Add the new inventory file to git, and push all your updates:

```

git add inventory_production
git add -u
git commit -m 'added prod stages'
git push -u origin master

```

```

ignw@ignw-jumphost:~/my_network_as_code$ git add inventory_production
ignw@ignw-jumphost:~/my_network_as_code$ git add -u
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m 'added prod stages'
>
[master 20f286d] added prod stages
Committer: ignw <ignw@ignw-jumphost.ignw.io>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:
git config --global --edit
After doing this, you may fix the identity used for this commit with:
git commit --amend --reset-author
2 files changed, 29 insertions(+), 5 deletions(-)
create mode 100644 inventory_production
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 558 bytes | 558.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:carlniger/my_network_as_code
  b9e2582..20f286d master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_network_as_code$
```

## Features Features Features

Before we test the final stages, check out the configs on the production CSR -- they should have pretty much no configurations on them since we haven't messed with them yet.

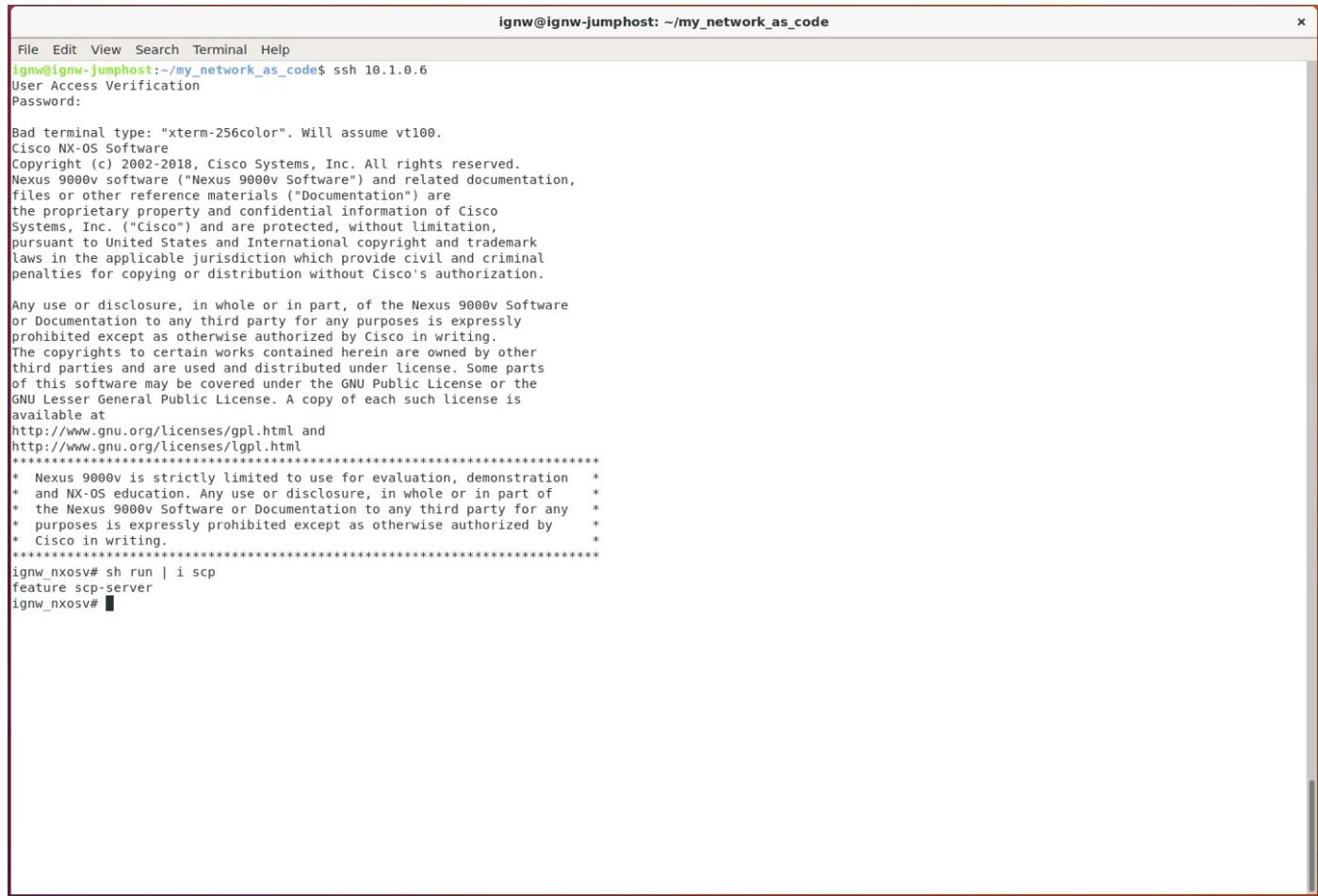
```

}
ignw@ignw-jumphost:~/my_network_as_code$ ssh 10.1.0.5
Password:

ignw-csr#show ip int brief
Interface          IP-Address      OK? Method Status          Protocol
GigabitEthernet1   10.1.0.5        YES NVRAM  up           up
GigabitEthernet2   unassigned     YES unset   administratively down down
ignw-csr#
```

We did have to enable a few features to get things in our journey of getting the dev side done -- will we need to go back and add these configs? We had to enable archive on the IOS-XE device, and also confirm that scp-server feature is enabled on the NX-OS device.

SCP should be enabled on the NX-OS device already, but go ahead and double check to be safe!



The screenshot shows a terminal window titled "ignw@ignw-jumphost: ~/my\_network\_as\_code". The window contains the following text:

```
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ ssh 10.1.0.6
User Access Verification
Password:
Bad terminal type: "xterm-256color". Will assume vt100.
Cisco NX-OS Software
Copyright (c) 2002-2018, Cisco Systems, Inc. All rights reserved.
Nexus 9000v software ("Nexus 9000v Software") and related documentation,
files or other reference materials ("Documentation") are
the proprietary property and confidential information of Cisco
Systems, Inc. ("Cisco") and are protected, without limitation,
pursuant to United States and International copyright and trademark
laws in the applicable jurisdiction which provide civil and criminal
penalties for copying or distribution without Cisco's authorization.

Any use or disclosure, in whole or in part, of the Nexus 9000v Software
or Documentation to any third party for any purposes is expressly
prohibited except as otherwise authorized by Cisco in writing.
The copyrights to certain works contained herein are owned by other
third parties and are used and distributed under license. Some parts
of this software may be covered under the GNU Public License or the
GNU Lesser General Public License. A copy of each such license is
available at
http://www.gnu.org/licenses/gpl.html and
http://www.gnu.org/licenses/lgpl.html
*****
* Nexus 9000v is strictly limited to use for evaluation, demonstration *
* and NX-OS education. Any use or disclosure, in whole or in part of *
* the Nexus 9000v Software or Documentation to any third party for any *
* purposes is expressly prohibited except as otherwise authorized by *
* Cisco in writing. *
*****
ignw_nxosv# sh run | i scp
feature scp-server
ignw_nxosv#
```

Recall also that we had to set the hostname so our Playbook merge operations didn't break. It would be nice if we didn't have to bother with that so we can change a hostname at a later date if we wanted to. We can add a simple task to use the Ansible core "nxos\_config" module to do this for us.

Add a new provider dictionary and task to your NXOS section of the deploy Playbook:

```

- name: Deploy Cisco NX-OS Configurations
  connection: local
  hosts: cisco-nxos
  gather_facts: no
  vars:
    creds:
      hostname: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
      dev_os: "{{ device_type }}"
    nxos_config_creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
      transport: cli
  tags: nxos
  tasks:
    - nxos_config:
        provider: "{{ nxos_config_creds }}"
        lines: "hostname {{ inventory_hostname }}"
    - napalm_install_config:
        provider: "{{ creds }}"
        config_file: "configs/{{ inventory_hostname }}"
        commit_changes: True
        replace_config: False
        register: napalm_diff
    - debug:
        msg: "{{ item }}"
      with_items: "{{ napalm_diff.msg.split('\n') }}"

```

We may as well do the same thing to setup archive on the IOS-XE devices. While we're at it, we can also add a Task to set the hostname of the device prior to the config merge -- this should help us avoid any trouble with that in the future. Modify the Play for the IOS-XE deployment to match the setup below:

```

- name: Deploy Cisco IOS-XE Router Configurations
  connection: local
  hosts: cisco-ios-xe-routers
  gather_facts: no
  vars:
    creds:
      hostname: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
      dev_os: "{{ device_type }}"
    ios_config_creds:
      host: "{{ ansible_host }}"
      username: "{{ username }}"
      password: "{{ password }}"
  tags: ios-xe
  tasks:
    - ios_config:
        provider: "{{ ios_config_creds }}"
        lines:
          - path bootflash:archive
          - write-memory
        parents: archive
    - ios_config:
        provider: "{{ ios_config_creds }}"
        lines:
          - "hostname {{ inventory_hostname }}"
    - napalm_install_config:
        provider: "{{ creds }}"
        config_file: "configs/{{ inventory_hostname }}"
        commit_changes: True
        replace_config: False
        register: napalm_diff
    - debug:
        msg: "{{ item }}"
        with_items: "{{ napalm_diff.msg.split('\n') }}"

```

Push those minor changes back to GitHub.

```

git add -u
git commit -m 'added nxos hostname changer + ios-xe archive'
git push -u origin master

```

```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~$ cd my_network_as_code/
ignw@ignw-jumphost:~/my_network_as_code$ git add -u
ignw@ignw-jumphost:~/my_network_as_code$ git commit -m 'added nxos hostname changer + ios-xe archive'
[master ec35408] added nxos hostname changer + ios-xe archive
Committer: ignw <ignw@ignw-jumphost.ignw.io>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 16 insertions(+), 9 deletions(-)
ignw@ignw-jumphost:~/my_network_as_code$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 349 bytes | 349.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:carlniger/my_network_as_code
 b30fc4b..ec35408  master -> master
Branch master set up to track remote branch master from origin.
ignw@ignw-jumphost:~/my_network_as_code$
```

Finally, the ASAv SCP functionality required for our workflow should be handled by the Python script we wrote. Obviously the SCP feature and archive configurations could be automated as well -- at scale that would probably be a very good idea, but for the small lab we're working with simply enabling these manually is OK.

With all of that out of the way, head over to Jenkins and trigger a build.

my\_network\_as\_code #50 Console [Jenkins] - Mozilla Firefox

my\_network\_as\_code #50 Jenkins my\_network\_as\_code #50

```
[Pipeline] stage
[Pipeline] { (Backup Prod Configurations)
[Pipeline] sh
[workspace] Running shell script
+ ansible-playbook backup_configurations.yaml -e ansible_python_interpreter=jenkins_build/bin/python -i inventory_production

PLAY [Backup Cisco ASA Configurations] ****
TASK [asa_config] ****
ok: [acme-sea-asal]

TASK [replace] ****
changed: [acme-sea-asal]

PLAY [Backup Cisco IOS-XE Router Configurations]
TASK [Gathering Facts] ****
ok: [acme-sea-rtr1]

TASK [napalm_get_facts] ****
ok: [acme-sea-rtr1]

TASK [copy] ****
changed: [acme-sea-rtr1]

TASK [replace] ****
fatal: [acme-sea-rtr1]: FAILED! => {"changed": false, "msg": "Path backup/acme-sea-rtr1_config.2018-05-29@15:36:48\nbackup/acme-sea-rtr1_config.2018-05-29@15:39:55 does not exist!", "rc": 257}

PLAY RECAP ****
acme-sea-asal      : ok=2    changed=1    unreachable=0    failed=0
acme-sea-rtr1     : ok=3    changed=1    unreachable=0    failed=1

[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 2
Finished: FAILURE
```

Page generated: May 29, 2018 3:40:30 PM UTC REST API Jenkins ver. 2.107.2

Uhoh, what do we have going on here? Our "replace" task for the IOS-XE device (prod) has failed looking for the backup file. Why? It appears that the backup *did* get taken, and we can confirm that by investigating the Jenkins server build directory:

```
ignw@ignw-jenkins: /var/lib/jenkins/jobs/my_network_as_code/workspace/backup x
File Edit View Search Terminal Help
ignw@ignw-jenkins: /var/lib/jenkins/jobs/my_network_as_code/workspace/backup$ ls
ignw@ignw-jenkins: /var/lib/jenkins/jobs/my_network_as_code/workspace/backup$ ls
acme-sea-asal_config.2018-05-29@15:39:54  acme-sea-rtr1_config.2018-05-29@15:36:48
acme-sea-nxos1_config.2018-05-29@15:37:03  acme-sea-rtr1_config.2018-05-29@15:39:55
ignw@ignw-jenkins: /var/lib/jenkins/jobs/my_network_as_code/workspace/backup$ cat acme-sea-rtr1
_config.2018-05-29@15\:36\:48 | grep hostname
hostname acme-sea-rtr1
ignw@ignw-jenkins: /var/lib/jenkins/jobs/my_network_as_code/workspace/backup$ cat acme-sea-rtr1
_config.2018-05-29@15\:39\:55 | grep hostname
hostname ****-----CSR
ignw@ignw-jenkins: /var/lib/jenkins/jobs/my_network_as_code/workspace/backup$
```

## Setting up the Archive Framework

Clearly our... less than ideal way of searching for the file to replace things in with a wildcard will not quite work anymore since we have two of the nearly identical files in the same directory. We could register a variable name in Ansible to address this (we can then search for that exact file instead of lazy \* searching), but this presents a decent opportunity to archive our configurations, so we'll take it!

We can have a "dev" and a "prod" folder in a parent "archive" folder to contain things and keep them separated so we don't run into a problem like this. Once we move files around, we can delete the backup directory between the dev and prod stages. We also need a way to keep the backups associated to the particular build ID in Jenkins -- that way we can go back and compare the starting config and the build output if needed. Finally, since we are ensuring our Jenkins workspace is destroyed at the beginning of each build we will need to evacuate the archive directory to somewhere safe before we wrap up. Oh, and one more thing -- if the build fails, we still should be able to archive our configs and pull them off of Jenkins... Lots to do, where to start!?

For now, let's create an archive directory with a dev and a prod child directory on our Jumphost. We may elect to move our archived configurations somewhere more appropriate (S3, GitHub, etc.) in the future, but for now this should work.

```
mkdir archive
mkdir archive/dev
mkdir archive/prod
```

Next we can add a simple function to our Jenkinsfile that will move our artifacts (backups) off of the build server

and into our newly created directory. Since we want to follow the DRY (do not repeat yourself) principle, we can write a single function that takes an argument of the environment so it knows where to move the files to.

```
def archive_configs(environment) {  
    // DO SOMETHING HERE!  
}
```

OK, functions are easy enough, and they take arguments similar to any other programming language so that is good! The simplest way to move these files around will be to SCP or rsync them to our Jumphost. The only challenge with this is authentication -- we want to automate all the things, not pause stuff while we wait for a user to enter a password. We can easily work around this by generating SSH keys on the Jenkins host and installing it as a known host on our Jumphost.

SSH to your Jenkins host and run the following commands:

```
sudo -iu jenkins  
ssh-keygen -t rsa  
ssh-copy-id ignw@10.10.0.254
```

```
ignw@ignw-jumphost: ~
File Edit View Search Terminal Help

ignw@ignw-jenkins:~$ sudo -iu jenkins
jenkins@ignw-jenkins:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/jenkins/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /var/lib/jenkins/.ssh/id_rsa.
Your public key has been saved in /var/lib/jenkins/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:6b0Vg0IUn5ZjI/2Pw5mEYzJfpe/d6M3LExyHQQK+GTU jenkins@ignw-jenkins
The key's randomart image is:
+---[RSA 2048]---+
| . . E.. |
| + = . o. |
| + @ + . o |
| o = * B ... |
| + S B . . o |
| * * * . o |
| o B + . |
| = ..* |
| . .+o*|
+---[SHA256]---+
jenkins@ignw-jenkins:~$ ssh-copy-id ignw@10.10.0.254
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/var/lib/jenkins/.ssh/id_rsa.pub"
The authenticity of host '10.10.0.254 (10.10.0.254)' can't be established.
ECDSA key fingerprint is SHA256:BTp20GFE54EeN57IJI4tldcSH35FuA41L0BqKFot978.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
ignw@10.10.0.254's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'ignw@10.10.0.254'"
and check to make sure that only the key(s) you wanted were added.

jenkins@ignw-jenkins:~$ ssh ignw@10.10.0.254
Welcome to Ubuntu 17.10 (GNU/Linux 4.13.0-43-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

15 packages can be updated.
4 updates are security updates.

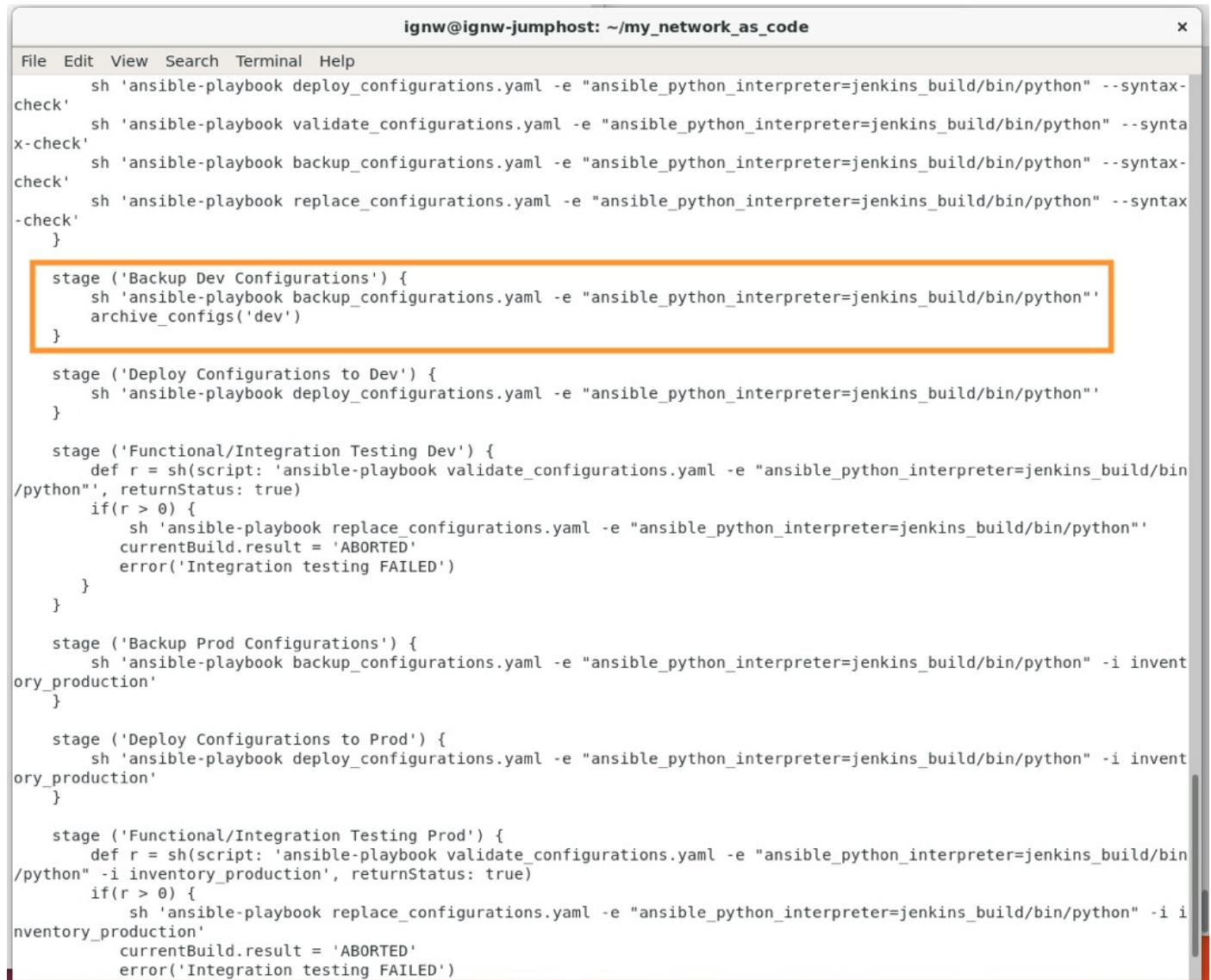
New release '18.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
```

Great, keys are created and we can SSH without being prompted for passwords.

Next we can modify our new function in the Jenkinsfile to rsync the backup configs to our archive directory in the appropriate environment. rsync is the perfect utility for this as it will create the directory for us too. Finally, we can access the build id (the number you see on your builds) with the `BUILD_ID` environment variable.

```
def archive_configs(environment) {
    sh "rsync backup/* ignw@10.10.0.254:/home/ignw/my_network_as_code/archive/$environment/$BUILD_ID"
}
```

When should we invoke this function? We need to do it before we move onto prod (as well as clean the directory so we don't have any more issues), but we also want to do it if our build fails. If we simply do this in the "backup" stage it probably makes the most sense. Modify your Jenkinsfile to export the configs.



```
ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
sh 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" --syntax-check'
}

stage ('Backup Dev Configurations') {
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"' &
        archive_configs('dev')
}

stage ('Deploy Configurations to Dev') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
}

stage ('Functional/Integration Testing Dev') {
    def r = sh(script: 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"', returnStatus: true)
    if(r > 0) {
        sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
        currentBuild.result = 'ABORTED'
        error('Integration testing FAILED')
    }
}

stage ('Backup Prod Configurations') {
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production'
}

stage ('Deploy Configurations to Prod') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production'
}

stage ('Functional/Integration Testing Prod') {
    def r = sh(script: 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production', returnStatus: true)
    if(r > 0) {
        sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production'
        currentBuild.result = 'ABORTED'
        error('Integration testing FAILED')
}
```

Now, before we run into the prod stages, add a stage to cleanup the backup directory, also run the archive function in the Prod backup stage to make sure we snag those configs too!

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
stage ('Backup Dev Configurations') {
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"''
    archive_configs('dev')
}

stage ('Deploy Configurations to Dev') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"''
}

stage ('Functional/Integration Testing Dev') {
    def r = sh(script: 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"', returnStatus: true)
    if(r > 0) {
        sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python"'
        currentBuild.result = 'ABORTED'
        error('Integration testing FAILED')
    }
}

stage ('Cleanup after Dev Tasks') {
    sh 'rm backup/*'
}

stage ('Backup Prod Configurations') {
    sh 'ansible-playbook backup_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production'
    archive_configs('prod')
}

stage ('Deploy Configurations to Prod') {
    sh 'ansible-playbook deploy_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production'
}

stage ('Functional/Integration Testing Prod') {
    def r = sh(script: 'ansible-playbook validate_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production', returnStatus: true)
    if(r > 0) {
        sh 'ansible-playbook replace_configurations.yaml -e "ansible_python_interpreter=jenkins_build/bin/python" -i inventory_production'
        currentBuild.result = 'ABORTED'
        error('Integration testing FAILED')
    }
}

}
ignw@ignw-jumphost:~/my_network_as_code$
```

Finally, push your changes to git and re-run your build. If all goes according to plan the prod devices should be configured and tested!

my\_network\_as\_code #10 Console [Jenkins] - Mozilla Firefox

my\_network\_as\_code #10 Jenkins

10.10.0.253:8080/job/my\_network\_as\_code/10/console

```
acme-sea-rtr1 : ok=2    changed=1    unreachable=0    failed=0

[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Functional/Integration Testing Prod)
[Pipeline] sh
[workspace] Running shell script
+ ansible-playbook validate_configurations.yaml -e ansible_python_interpreter=jenkins_build/bin/python -i inventory_production

PLAY [Validate "Core" to "Internet" Reachability] *****

TASK [nxos_ping] *****
ok: [acme-sea-nxos1]

PLAY [Validate "Core" & "Internet" Reachability from Firewall] *****

TASK [asa_command] *****
ok: [acme-sea-asal] => (item=8.8.8.8)
ok: [acme-sea-asal] => (item=10.255.255.2)
ok: [acme-sea-asal] => (item=10.255.254.1)

PLAY [Validate "Edge" Reachability from Edge Router] *****

TASK [Test Success Pings] *****
ok: [acme-sea-rtr1] => (item=203.0.113.2)

TASK [Test Fail Pings] *****
ok: [acme-sea-rtr1] => (item=10.255.255.2)

PLAY RECAP *****
acme-sea-asal      : ok=1    changed=0    unreachable=0    failed=0
acme-sea-nxos1     : ok=1    changed=0    unreachable=0    failed=0
acme-sea-rtr1      : ok=2    changed=0    unreachable=0    failed=0

[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Page generated: May 30, 2018 2:30:30 PM UTC REST API Jenkins ver. 2.107.2

If you run into any errors, try to run the build once more. In some cases the validation scripts for prod fail being ran immediately after provisioning due to slow lab gear!

# Refactoring

## Overview and Objectives

Up through this point we've been iteratively expanding on our pipeline, adding features, addressing issues, and working toward our end goal of an automated and validated deployment. This is great, however, because we've been iteratively building on this we've not had a chance to step back and look at things holistically. In development it is very common to get code working -- maybe not in an ideal fashion -- and then spend some time to "refactor" or re-write some of the code to improve upon the readability or simplicity. We will be doing just that in this lab.

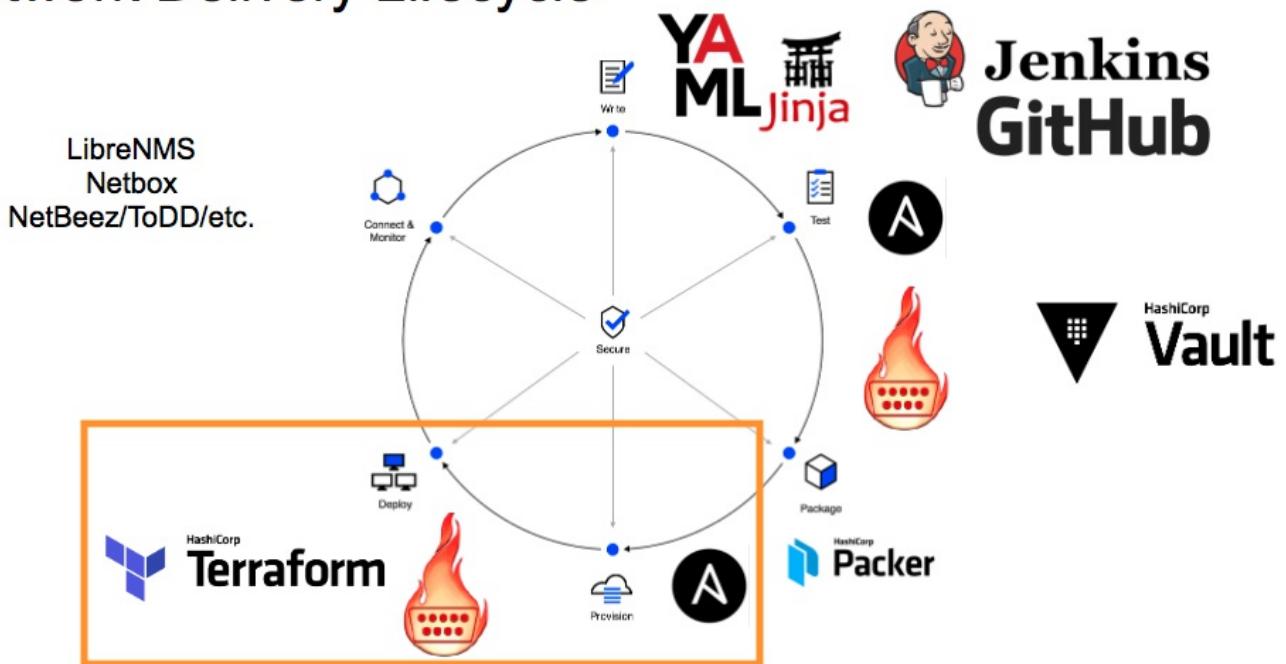
Objectives:

- Eliminate "hacks" from the Pipeline (sed replacement, passing Python paths, etc.)
- Improve the speed of our build (initializing the venv takes quite a bit of time!)
- Clean up/Optimize the Jenkinsfile/Pipeline

## Network Delivery Lifecycle Overview

In this lab we'll be touching a little bit of everything in the Network Delivery Lifecycle Overview -- we'll optimize the deployment by cutting down the time of our build, improve the sanity of our testing, and improve the overall readability/simplicity of our Lifecycle as a whole!

## Network Delivery Lifecycle



## Containerize the Build Environment

As briefly outlined in the "Objectives" section, we want to spend some time going back through what we've created and improving the readability and functionality of our Pipeline as a whole.

The most obvious thing that we can improve is simply speeding up our process as a whole. We can't do much to speed up the NAPALM/Ansible tasks, but we can make some serious improvements to how long our initial setup takes.

We've been creating a new Python venv (virtual environment) each time our build runs. We did this to ensure that each and every time we run our build we have a consistent environment from which to deploy/validate/backup/etc., and we need to ensure that we still have that functionality. A relatively simple way to achieve this is to create a container that already has all of our requirements/dependencies installed into it. Once we've created that, we can simply instantiate an instance of the container -- this process will take about one second... a pretty big improvement over installing all the dependencies each time we build!

Containers are sort of like virtual machines in that they run on a host machine, but unlike virtual machines, containers do not have an operating system installed. Containers share the operating system kernel of the host machine. This makes them super lightweight and very fast to spin up.

Docker is an open source tool that is used to create, deploy, and run containers; Docker is the industry standard container platform (you may be thinking to yourself what about Kubernetes, Rancher, etc.? -- These are container orchestration engines that sit "on top" of Docker/containers)

Docker has already been installed on both the Jumphost and Jenkins servers. If you needed to install Docker on an Ubuntu system such as this, you can install it as simply as:

```
apt-get install docker.io
```

In order to craft a container for our Pipeline we need to first define what we want *in* the container, we do this by creating a Dockerfile. A Dockerfile is kind of like a manifest, or a list of dependencies/requirements that define what should be installed into the Docker image.

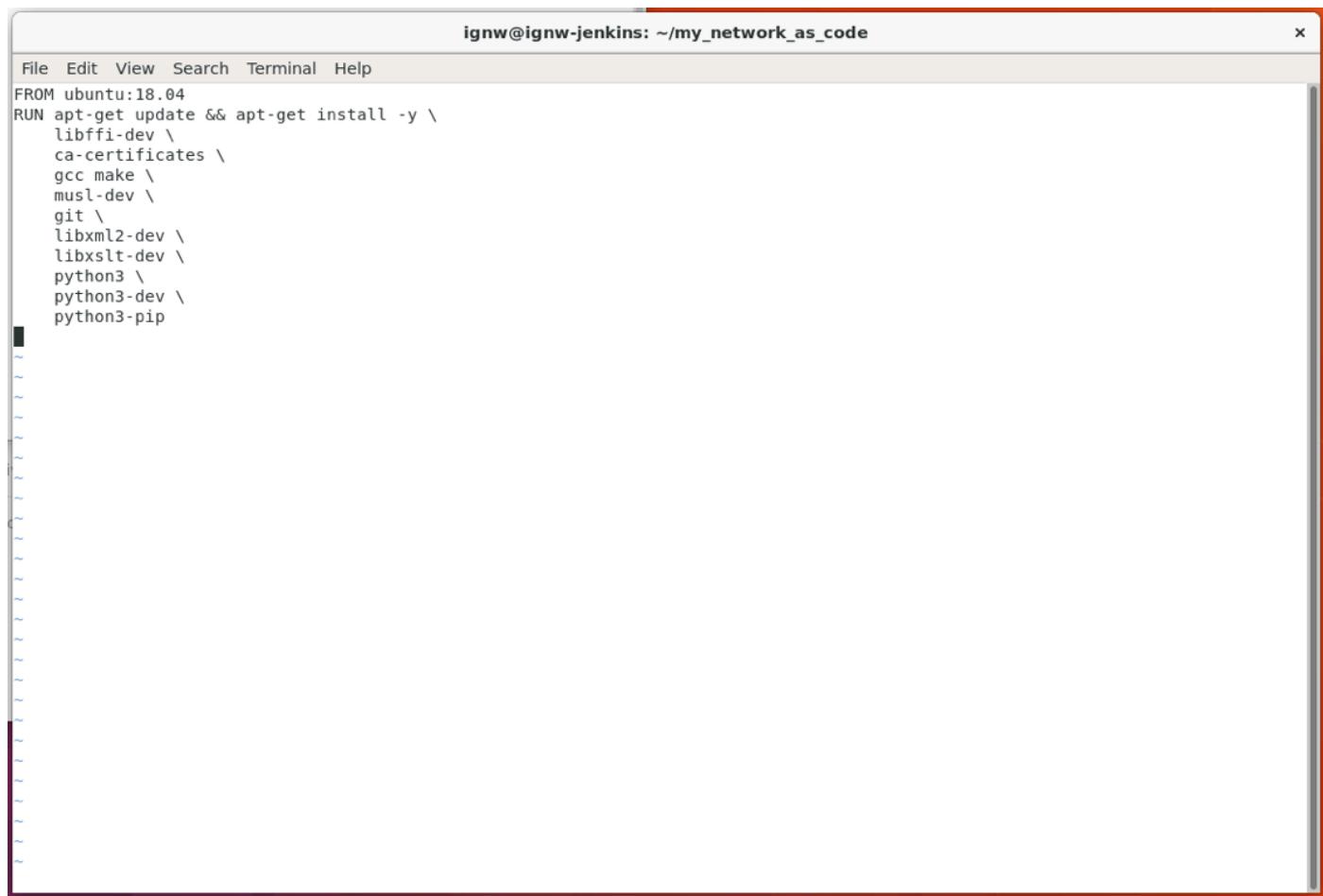
Create a new file in your repository simply called "Dockerfile"

```
touch Dockerfile
```

A Dockerfile generally starts with a "FROM" definition -- this basically allows us to pull dependencies down from other Docker images. In our case, we'll use Ubuntu 18.04 as the "base" for our image. In production you may wish to use something a bit more lightweight such as Alpine, but we'll stick with Ubuntu for consistency sake.

Next we can use the "RUN" command to... run things! We need to install all of our requirements so that we can execute our Playbooks. Copy and paste the following into your Dockerfile:

```
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y \
    libffi-dev \
    ca-certificates \
    gcc make \
    musl-dev \
    git \
    sshpass \
    libxml2-dev \
    libxslt-dev \
    python3 \
    python3-dev \
    python3-pip \
    rsync
```



A screenshot of a terminal window titled "ignw@ignw-jenkins: ~/my\_network\_as\_code". The window has a standard Linux-style interface with a menu bar (File, Edit, View, Search, Terminal, Help) and a title bar. The main area contains the Dockerfile code shown above. The terminal window is set against a light gray background.

```
ignw@ignw-jenkins: ~/my_network_as_code
File Edit View Search Terminal Help
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y \
    libffi-dev \
    ca-certificates \
    gcc make \
    musl-dev \
    git \
    libxml2-dev \
    libxslt-dev \
    python3 \
    python3-dev \
    python3-pip
```

That should get us all sorted on our Python requirements, next we need to install Ansible and NAPALM-Ansible. Due to some outstanding Pull Requests to improve Python 3 support, as well as to ensure we are not pulling from the latest dev branches we'll install/clone specific versions and make a minor update using sed.

```
RUN python3 -m pip install ansible==2.5.2
RUN git clone https://github.com/carlniger/napalm-ansible
RUN cp -r napalm-ansible/napalm_ansible/ /usr/local/lib/python3.6/dist-packages
/
RUN cd napalm-ansible && python3 setup.py install
RUN sed -i 's/for key in result.keys():/for key in list(result):/g' /usr/local/lib/python3.6/dist-packages/ansible/plugins/action/asa_config.py
```

```
ignw@ignw-jenkins: ~/my_network_as_code
File Edit View Search Terminal Help
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y \
    libffi-dev \
    ca-certificates \
    gcc make \
    musl-dev \
    git \
    libxml2-dev \
    libxslt-dev \
    python3 \
    python3-dev \
    python3-pip
RUN python3 -m pip install ansible==2.5.2
RUN git clone https://github.com/carlniger/napalm-ansible
RUN cp -r napalm-ansible/napalm_ansible/ /usr/local/lib/python3.6/dist-packages/
RUN cd napalm-ansible && python3 setup.py install
RUN sed -i 's/for key in result.keys():/for key in list(result):/g' /usr/local/lib/python3.6/dist-packages/ansible/plugins/actio
n/asa_config.py

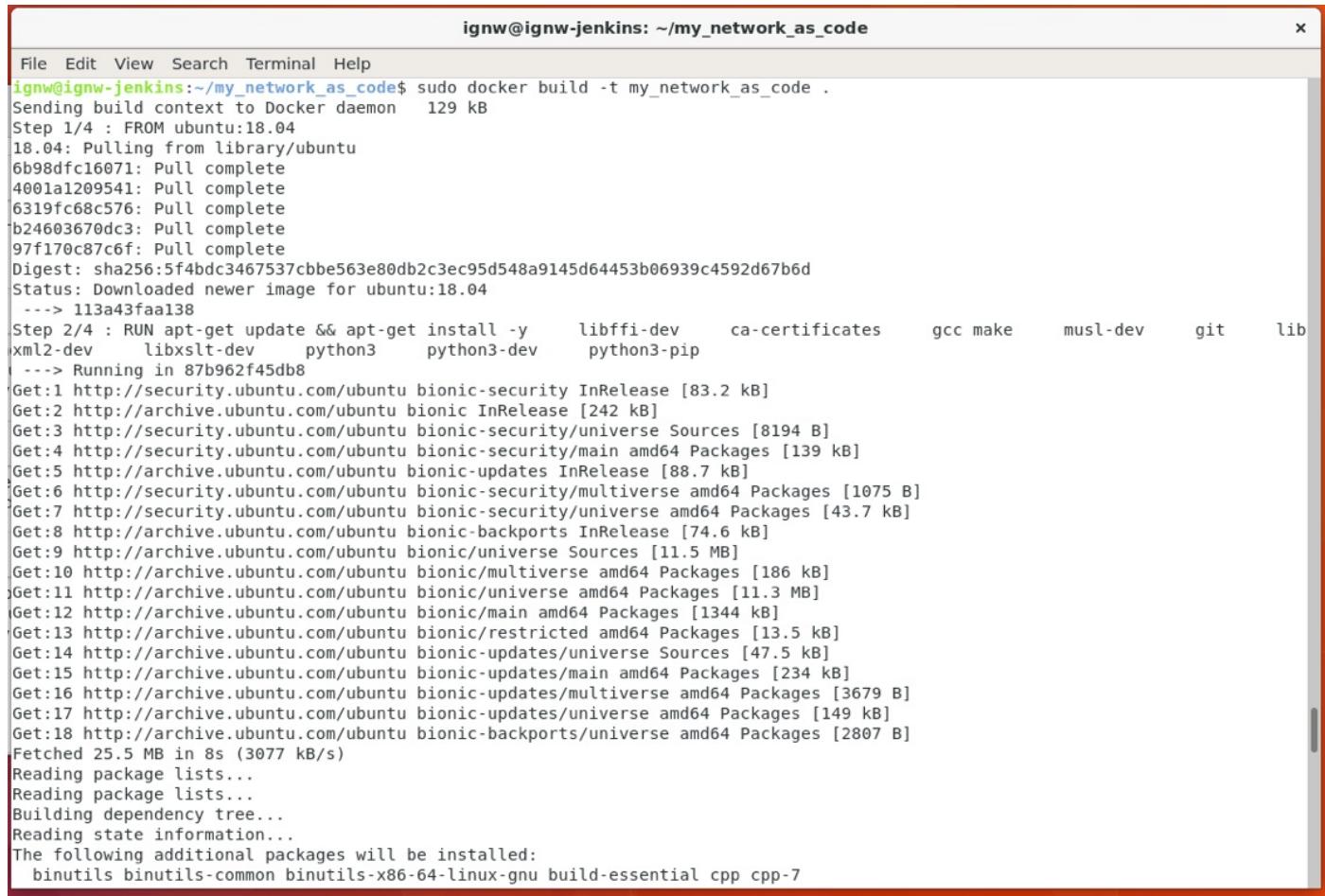
"Dockefile" 18 lines, 600 characters written
```

As you can see we are pinning our Ansible version to 2.5.2 (note that 2.6 contains the fix that eliminates our sed patch, but this has not been fully tested against the lab so we'll stick to 2.5.2 for now!), and we are cloning a fork of NAPALM-Ansible (again, the PR has been merged for NAPALM-Ansible as well, but the rest of the latest dev branch hasn't been lab tested). Finally we'll manually install NAPALM-Ansible, and patch the minor Ansible tweak. This should look pretty familiar to what you've done previously in the Jenkinsfile.

Containers are spawned from images -- now that we have defined what we want in our container, we need to actually build the image itself. Run the following command to do so:

```
sudo docker build -t my network as code .
```

**NOTE THE PERIOD AT THE END!**



The screenshot shows a terminal window titled "ignw@ignw-jenkins: ~/my\_network\_as\_code". The window contains the output of a Docker build command:

```
ignw@ignw-jenkins:~/my_network_as_code$ sudo docker build -t my_network_as_code .
Sending build context to Docker daemon 129 kB
Step 1/4 : FROM ubuntu:18.04
18.04: Pulling from library/ubuntu
6b98dfc16071: Pull complete
4001a1209541: Pull complete
6319fc68c576: Pull complete
b24603670dc3: Pull complete
97f170c87c6f: Pull complete
Digest: sha256:5f4bdcc3467537cbbe563e80db2c3ec95d548a9145d64453b06939c4592d67b6d
Status: Downloaded newer image for ubuntu:18.04
--> 113a43faa138
Step 2/4 : RUN apt-get update && apt-get install -y libffi-dev ca-certificates gcc make musl-dev git libxml2-dev libxslt-dev python3 python3-dev python3-pip
--> Running in 87b962f45db8
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [83.2 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:3 http://security.ubuntu.com/ubuntu bionic-security/universe Sources [8194 B]
Get:4 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [139 kB]
Get:5 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:6 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [1075 B]
Get:7 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [43.7 kB]
Get:8 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:9 http://archive.ubuntu.com/ubuntu bionic/universe Sources [11.5 MB]
Get:10 http://archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [186 kB]
Get:11 http://archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [11.3 MB]
Get:12 http://archive.ubuntu.com/ubuntu bionic/main amd64 Packages [1344 kB]
Get:13 http://archive.ubuntu.com/ubuntu bionic/restricted amd64 Packages [13.5 kB]
Get:14 http://archive.ubuntu.com/ubuntu bionic-updates/universe Sources [47.5 kB]
Get:15 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [234 kB]
Get:16 http://archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Packages [3679 B]
Get:17 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [149 kB]
Get:18 http://archive.ubuntu.com/ubuntu bionic-backports/universe amd64 Packages [2807 B]
Fetched 25.5 MB in 8s (3077 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential cpp cpp-7
```

The build will take a while as it pulls down the Ubuntu image, then installs all of our dependencies....

```
ignw@ignw-jenkins: ~/my_network_as_code
File Edit View Search Terminal Help
Searching for pycparser==2.18
Best match: pycparser 2.18
Adding pycparser 2.18 to easy-install.pth file

Using /usr/local/lib/python3.6/dist-packages
Searching for pyasn1==0.4.3
Best match: pyasn1 0.4.3
Adding pyasn1 0.4.3 to easy-install.pth file

Using /usr/local/lib/python3.6/dist-packages
Searching for PyNaCl==1.2.1
Best match: PyNaCl 1.2.1
Adding PyNaCl 1.2.1 to easy-install.pth file

Using /usr/local/lib/python3.6/dist-packages
Searching for cryptography==2.1.4
Best match: cryptography 2.1.4
Adding cryptography 2.1.4 to easy-install.pth file

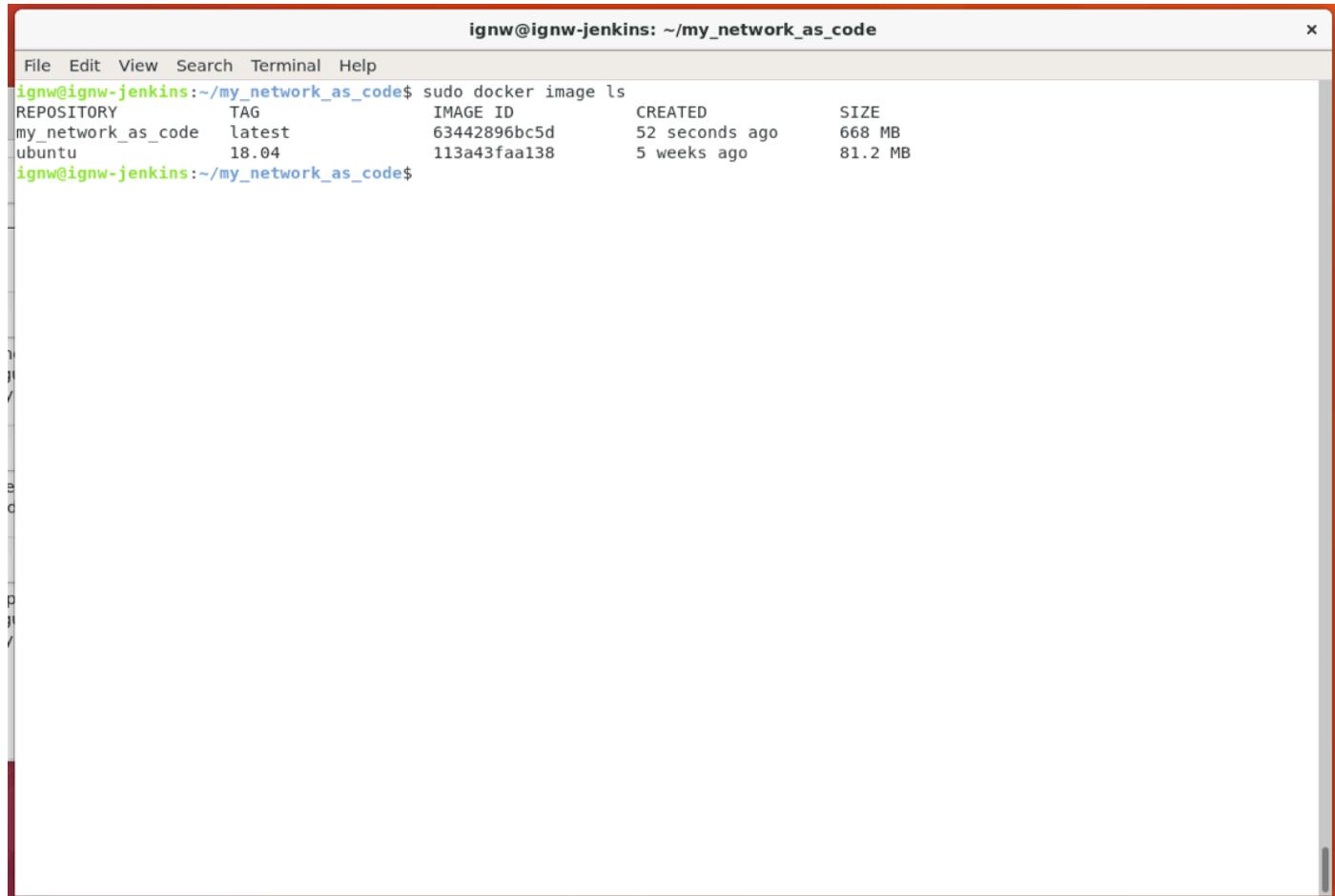
Using /usr/lib/python3/dist-packages
Searching for bcrypt==3.1.4
Best match: bcrypt 3.1.4
Adding bcrypt 3.1.4 to easy-install.pth file

Using /usr/local/lib/python3.6/dist-packages
Searching for idna==2.6
Best match: idna 2.6
Adding idna 2.6 to easy-install.pth file

Using /usr/lib/python3/dist-packages
Finished processing dependencies for napalm-ansible==0.9.1
---> e4c9d4de7429
Removing intermediate container 2aa5f50f1a78
Step 7/7 : RUN sed -i 's/for key in result.keys():/for key in list(result):/g' /usr/local/lib/python3.6/dist-packages/ansible/plugins/action/asa_config.py
---> Running in fe25d3e6808c
---> 0f7b5ff987f9
Removing intermediate container fe25d3e6808c
Successfully built 0f7b5ff987f9
ignw@ignw-jenkins:~/my_network_as_code$
```

When it is finally done, we can check to see what images are available on our system:

```
sudo docker image ls
```

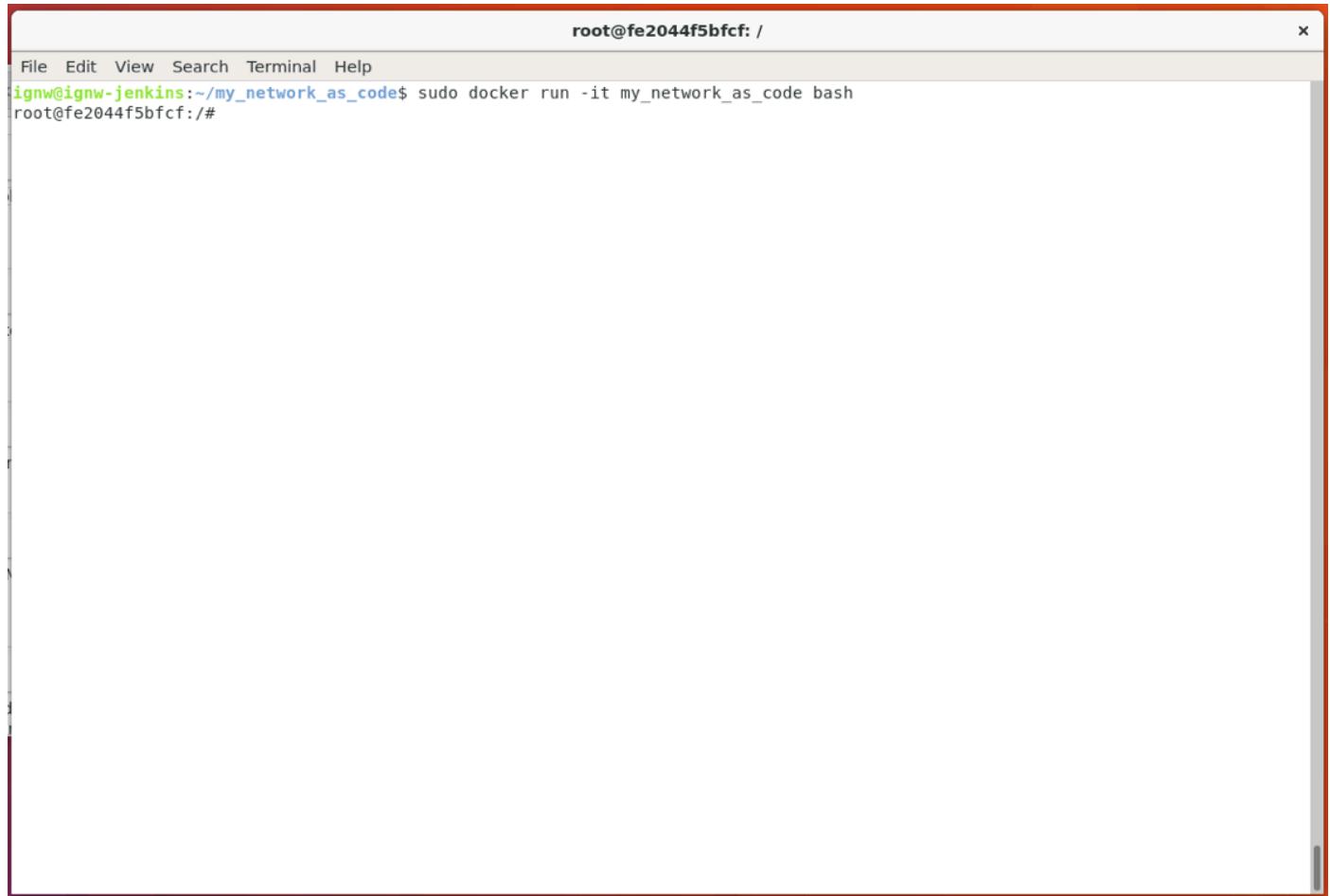


A screenshot of a terminal window titled "ignw@ignw-jenkins: ~/my\_network\_as\_code". The window has a standard OS X-style title bar with icons for close, minimize, and zoom. The main area shows the command "sudo docker image ls" being run, followed by a table of Docker images. The table includes columns for REPOSITORY, TAG, IMAGE ID, CREATED, and SIZE. Two images are listed: "my\_network\_as\_code latest" (image ID 63442896bc5d, created 52 seconds ago, size 668 MB) and "ubuntu 18.04" (image ID 113a43faa138, created 5 weeks ago, size 81.2 MB). The terminal prompt "ignw@ignw-jenkins: ~/my\_network\_as\_code\$" is visible at the bottom.

| REPOSITORY         | TAG    | IMAGE ID     | CREATED        | SIZE    |
|--------------------|--------|--------------|----------------|---------|
| my_network_as_code | latest | 63442896bc5d | 52 seconds ago | 668 MB  |
| ubuntu             | 18.04  | 113a43faa138 | 5 weeks ago    | 81.2 MB |

With our image (hopefully) successfully created, we can now test it out before we begin integrating it into the Pipeline. To launch a container and attach to the shell execute the following command:

```
sudo docker run -it my_network_as_code bash
```



The screenshot shows a terminal window with a red border. The title bar says "root@fe2044f5bfcf: /". The menu bar includes "File Edit View Search Terminal Help". The command line shows: "ignw@ignw-jenkins:~/my\_network\_as\_code\$ sudo docker run -it my\_network\_as\_code bash". The prompt "root@fe2044f5bfcf:/#" is visible at the bottom.

Now that we've built the container, notice how fast it started! You can already see how much faster than installing dependency every time we run a job in our Pipeline!

To test things out we need to clone our repository and run the "backup\_configurations" Playbook -- this Playbook will test to ensure Ansible and NAPALM-Ansible are installed and good to go and should be a good litmus test to ensure the rest of our tasks will run smoothly.

Clone your GitHub repository:

```
git clone [URL OF YOUR GITHUB REPO]
```

*Note* If you are using a private repo you'll need to enter your credentials as shown below, otherwise your repo should simply be cloned without prompting.

```
root@fe2044f5bfcf: /  
File Edit View Search Terminal Help  
ignw@ignw-jenkins:~/my_network_as_code$ sudo docker run -it my_network_as_code bash  
root@fe2044f5bfcf:/#  
root@fe2044f5bfcf:/#  
root@fe2044f5bfcf:/# git clone https://github.com/carlniger/my_network_as_code  
Cloning into 'my_network_as_code'...  
Username for 'https://github.com': carlniger  
Password for 'https://carlniger@github.com':  
remote: Counting objects: 117, done.  
remote: Compressing objects: 100% (102/102), done.  
remote: Total 117 (delta 57), reused 48 (delta 11), pack-reused 0  
Receiving objects: 100% (117/117), 25.43 KiB | 8.48 MiB/s, done.  
Resolving deltas: 100% (57/57), done.  
root@fe2044f5bfcf:/# █
```

Move into the "my\_network\_as\_code" directory and run the backup Playbook:

```
cd my_network_as_code  
ansible-playbook backup_configurations.yaml
```

```
root@fe2044f5bfcf: /my_network_as_code
File Edit View Search Terminal Help

TASK [napalm_get_facts] ****
ok: [acme-sea-rtr1]

TASK [copy] ****
changed: [acme-sea-rtr1]

TASK [replace] ****
changed: [acme-sea-rtr1]

TASK [shell] ****
[WARNING]: Consider using the replace, lineinfile or template module rather than running sed. If you need to use command because replace, lineinfile or template is insufficient you can add warn=False to this command task or set command_warnings=False in ansible.cfg to get rid of this message.

changed: [acme-sea-rtr1]

PLAY [Backup Cisco NXOS Configurations] ****

TASK [Gathering Facts] ****
ok: [acme-sea-nxos1]

TASK [nxos_rollback] ****
changed: [acme-sea-nxos1]

TASK [napalm_get_facts] ****
ok: [acme-sea-nxos1]

TASK [copy] ****
changed: [acme-sea-nxos1]

TASK [replace] ****
changed: [acme-sea-nxos1]

PLAY RECAP ****
acme-sea-asal      : ok=2    changed=1    unreachable=0    failed=0
acme-sea-nxos1     : ok=5    changed=3    unreachable=0    failed=0
acme-sea-rtr1      : ok=5    changed=3    unreachable=0    failed=0

root@fe2044f5bfcf:/my network as code#
```

If your Playbook finished successfully you are good to go! You can exit from your container simply with the 'exit' command. Once out, you can check to see if your container still exists:

```
sudo docker container ls
```

```

ignw@ignw-jenkins: ~/my_network_as_code
File Edit View Search Terminal Help
TASK [copy] ****
changed: [acme-sea-rtr1]

TASK [replace] ****
changed: [acme-sea-rtr1]

TASK [shell] ****
[WARNING]: Consider using the replace, lineinfile or template module rather than running sed. If you need to use command because replace, lineinfile or template is insufficient you can add warn=False to this command task or set command_warnings=False in ansible.cfg to get rid of this message.

changed: [acme-sea-rtr1]

PLAY [Backup Cisco NXOS Configurations] ****

TASK [Gathering Facts] ****
ok: [acme-sea-nxos1]

TASK [nxos_rollback] ****
changed: [acme-sea-nxos1]

TASK [napalm_get_facts] ****
ok: [acme-sea-nxos1]

TASK [copy] ****
changed: [acme-sea-nxos1]

TASK [replace] ****
changed: [acme-sea-nxos1]

PLAY RECAP ****
acme-sea-asal      : ok=2    changed=1    unreachable=0    failed=0
acme-sea-nxos1     : ok=5    changed=3    unreachable=0    failed=0
acme-sea-rtr1      : ok=5    changed=3    unreachable=0    failed=0

root@fe2044f5bfcf:/my_network_as_code# exit
exit
ignw@ignw-jenkins:~/my_network_as_code$ sudo docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS          NAMES
ignw@ignw-jenkins:~/my_network_as_code$ 

```

As you can see the container is already gone! Nice and easy!

Now would be a good time to add the Dockerfile to your repository and push it back up to GitHub.

```

git add Dockerfile
git commit -m 'created Dockerfile'
git push -u origin master

```

## Overhauling the Jenkinsfile

At this point we need to do a pretty significant overhaul to our Jenkinsfile for a few reasons. First and foremost we just want to clean it up and make it a bit easier on the eyes! Perhaps more importantly though we need to modify it to allow Jenkins to execute its steps *in* our container.

Create a backup of the current Jenkinsfile so that we can start more or less from scratch on a new one and just copy/paste in what we need.

```
mv Jenkinsfile orig_Jenkinsfile  
touch Jenkinsfile
```

Starting from the top, copy the "archive\_configs" function you created before -- this is pretty tidy code and we can reuse that.

```
def archive_configs(environment) {  
    sh "rsync backup/* ignw@10.10.0.254:/home/ignw/my_network_as_code/archive/$e  
nvironment/$BUILD_ID"  
}
```

We previously used "node" syntax which is for a "scripted pipeline". This is fine and pretty straight forward which is why we chose it, but there is a more modern syntax called a "declarative pipeline" which we will now switch to. The formatting is very similar, so this will be a pretty easy change.

The basic syntax is as follows:

```
pipeline {  
    agent {  
        XYZ  
    }  
    stages {  
        stage('XYZ') {  
            steps {  
                XYZ  
            }  
        }  
        post {  
            XYZ  
        }  
    }  
}
```

Let's prepare our "checkout" stage in the new format.

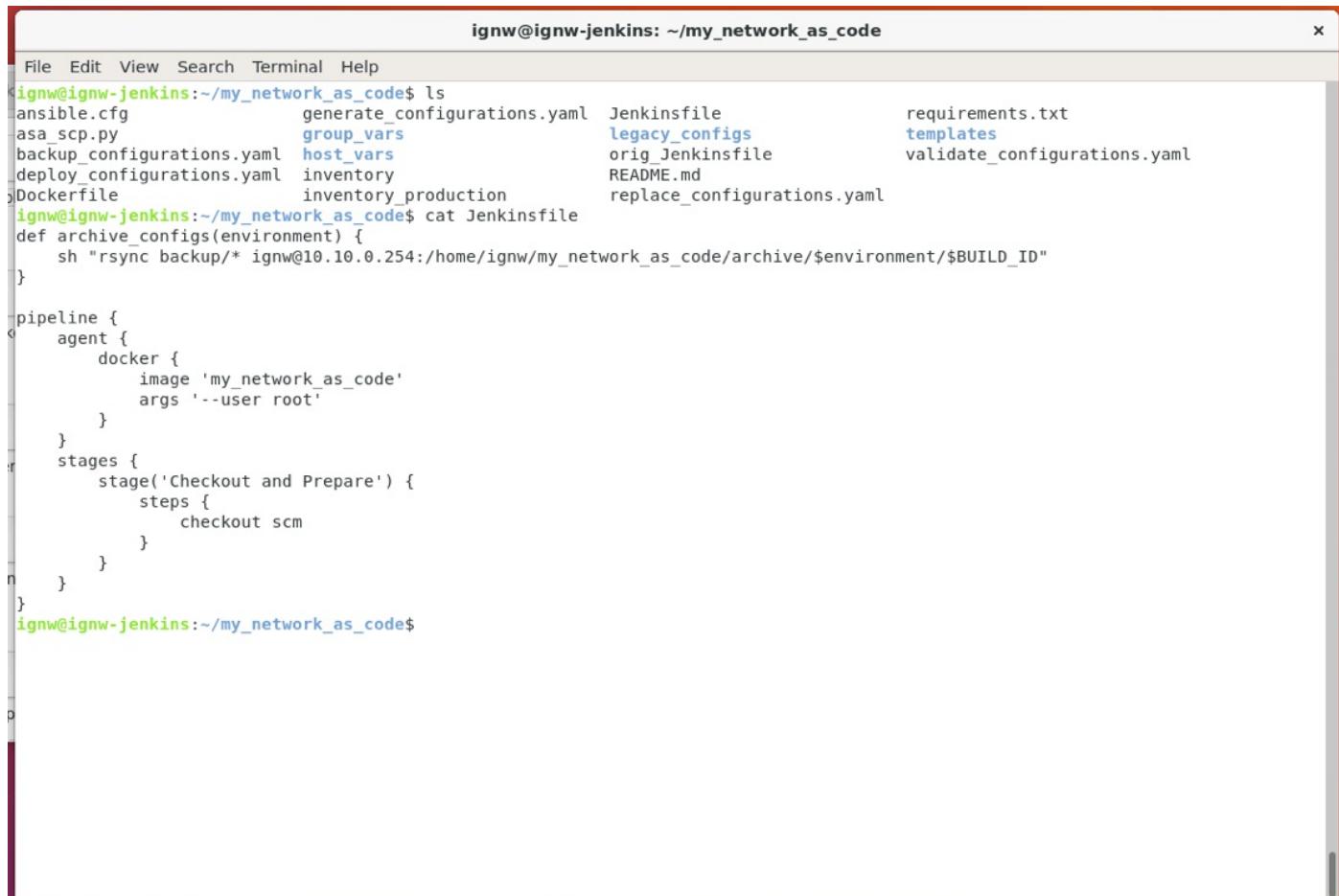
```
pipeline {  
    stages {  
        stage('Checkout and Prepare') {  
            steps {  
                checkout scm  
            }  
        }  
    }  
}
```

Easy enough! Before we go any further though, we need to make sure that our checkout (and everything subsequently) is being executed IN our container. To do that we can set the "agent" that Jenkins is going to use in the "agent" code block:

```
agent {  
    docker {  
        image 'my_network_as_code'  
        args '--user root'  
    }  
}
```

As you can see we are setting the agent to "docker" and pointing to the image we created (we created it on the Jumphost for testing, so we'll need to build it on Jenkins eventually as well), finally we are going to enter into the container as the root user so we don't have to futz with permissions (probably not something you would do in production!).

At this point your new Jenkinsfile should look something like this:



The screenshot shows a terminal window titled "ignw@ignw-jenkins: ~/my\_network\_as\_code". The window contains the Jenkinsfile code. The code defines a pipeline with an agent block using a Docker image and arguments. It also includes stages for checkout and preparation.

```
File Edit View Search Terminal Help  
ignw@ignw-jenkins:~/my_network_as_code$ ls  
ansible.cfg generate_configurations.yaml Jenkinsfile requirements.txt  
asa_scp.py group_vars legacy_configs templates  
backup_configurations.yaml host_vars orig_Jenkinsfile validate_configurations.yaml  
deploy_configurations.yaml inventory README.md  
Dockerfile inventory_production replace_configurations.yaml  
ignw@ignw-jenkins:~/my_network_as_code$ cat Jenkinsfile  
def archive_configs(environment) {  
    sh "rsync backup/* ignw@10.10.0.254:/home/ignw/my_network_as_code/archive/$environment/$BUILD_ID"  
}  
  
pipeline {  
    agent {  
        docker {  
            image 'my_network_as_code'  
            args '--user root'  
        }  
    }  
    stages {  
        stage('Checkout and Prepare') {  
            steps {  
                checkout scm  
            }  
        }  
    }  
}  
ignw@ignw-jenkins:~/my_network_as_code$
```

To confirm that we are indeed executing commands from inside of our container, let's execute a simple command that would help us to validate that. We can run "ip address" to see if the output is the interfaces of the Jenkins host itself, or the Docker bridged interfaces. Add the following to the "Checkout" Stage:

```
sh 'ip address'
```

With that sorted out, push your changes to GitHub. Next we'll go build the container on the Jenkins host and give it a test!

## Initial Testing Jenkins and Docker

On your Jenkins (!!! Important, hop over to your Jenkins host !!!) host, copy your Dockerfile over and create the image again.

The screenshot shows a terminal window titled "ignw@ignw-jenkins: ~". The terminal output is as follows:

```
ignw@ignw-jenkins:~/my_network_as_code$ ssh 10.10.0.253
The authenticity of host '10.10.0.253 (10.10.0.253)' can't be established.
ECDSA key fingerprint is SHA256:QD3ezcnVG69up9cJM00GS4TdzuQtt+PgkLjNUr7Zwbk.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.10.0.253' (ECDSA) to the list of known hosts.
ignw@10.10.0.253's password:
Welcome to Ubuntu 17.10 (GNU/Linux 4.13.0-45-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

New release '18.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Jul 11 14:03:20 2018 from 10.10.10.123
ignw@ignw-jenkins:~$ vi Dockerfile
ignw@ignw-jenkins:~$ ls
ansible Dockerfile get-pip.py
ignw@ignw-jenkins:~$ sudo docker build -t my_network_as_code .
Sending build context to Docker daemon 1.667 MB
Step 1/7 : FROM ubuntu:18.04
18.04: Pulling from library/ubuntu
6b98dfc16071: Downloading [=====>] 15.93 MB/31.26 MB
4001a1209541: Download complete
6319fc68c576: Download complete
b24603670dc3: Download complete
97f170c87c6f: Download complete
```

```
ignw@ignw-jenkins: ~
File Edit View Search Terminal Help

Using /usr/local/lib/python3.6/dist-packages
Searching for PyNaCl==1.2.1
Best match: PyNaCl 1.2.1
Adding PyNaCl 1.2.1 to easy-install.pth file

Using /usr/local/lib/python3.6/dist-packages
Searching for cryptography==2.1.4
Best match: cryptography 2.1.4
Adding cryptography 2.1.4 to easy-install.pth file

Using /usr/lib/python3/dist-packages
Searching for pyasn1==0.4.3
Best match: pyasn1 0.4.3
Adding pyasn1 0.4.3 to easy-install.pth file

Using /usr/local/lib/python3.6/dist-packages
Searching for bcrypt==3.1.4
Best match: bcrypt 3.1.4
Adding bcrypt 3.1.4 to easy-install.pth file

Using /usr/local/lib/python3.6/dist-packages
Searching for idna==2.6
Best match: idna 2.6
Adding idna 2.6 to easy-install.pth file

Using /usr/lib/python3/dist-packages
Finished processing dependencies for napalm-ansible==0.9.1
--> 8b04c21c2dbd
Removing intermediate container e347afc59e74
Step 7/7 : RUN sed -i 's/for key in result.keys():/for key in list(result):/g' /usr/local/lib/python3.6/dist-packages/ansible/plugins/action/asa_config.py
--> Running in e86daf17dbae
--> 8550a4489f78
Removing intermediate container e86daf17dbae
Successfully built 8550a4489f78
ignw@ignw-jenkins:~$ 
ignw@ignw-jenkins:~$ sudo docker image ls
REPOSITORY          TAG           IMAGE ID            CREATED             SIZE
my_network_as_code  latest        8550a4489f78      34 seconds ago    655 MB
ubuntu              18.04         113a43faa138      5 weeks ago       81.2 MB
ignw@ignw-jenkins:~$
```

Once your container is ready (and make sure you pushed your updates to GitHub!) we can attempt to run our build:

Jenkins > my\_network\_as\_code > #2

Status  
Changes  
**Console Output**  
View as plain text  
Edit Build Information  
Git Build Data  
No Tags  
Thread Dump  
Pause/resume  
Replay  
Pipeline Steps  
Previous Build

## Console Output

```
Started by user ignw
Obtained Jenkinsfile from git https://github.com/carlniger/my_network_as_code
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/jobs/my_network_as_code/workspace
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/carlniger/my_network_as_code # timeout=10
Fetching upstream changes from https://github.com/carlniger/my_network_as_code
> git -version # timeout=10
using GIT_ASKPASS to set credentials
> git fetch --tags --progress https://github.com/carlniger/my_network_as_code +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Commit message: "updated jenkinsfile"
Checking out Revision 775a4040f0ab160df50334e098d0a6e37d5f4925 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 775a4040f0ab160df50334e098d0a6e37d5f4925
Commit message: "updated jenkinsfile"
> git rev-list --no-walk 63fb6d0a6b5577571aa6ad3ec5774f47ca71f81a # timeout=10
[Pipeline]
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
[workspace] Running shell script
+ docker inspect -f . my_network_as_code

Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://127.0.0.1:2375/containers/my_network_as_code/json: dial unix /var/run/docker.sock: connect: permission denied
[Pipeline] sh
[workspace] Running shell script
+ docker pull my_network_as_code
Using default tag: latest
Warning: failed to get default registry endpoint from daemon (Get permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://127.0.0.1:2375/containers/my_network_as_code/json: dial unix /var/run/docker.sock: connect: permission denied). Using system default: https://index.docker.io/v1/
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://127.0.0.1:2375/images/create?fromImage=my_network_as_code&tag=latest: dial unix /var/run/docker.sock: connect: permission denied
[Pipeline]
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
```

Uhoh! Looks like Jenkins is having some permissions issues connecting to the Docker daemon. To solve this we can add jenkins to the docker group. Run the following command on your *Jenkins* host and then restart the system.

```
sudo usermod -aG docker jenkins
```

With that out of the way, run the build again to see what happens.

This is actually a great error for us to get, and confirms that Jenkins is actually executing our steps inside of the container. From the output we can see that the command "ip address" is not available. If you executed this command on either the Jumphost or Jenkins hosts you would see the IP addresses of the devices. The container, due to its lightweight nature, doesn't have this executable installed, and thus this command would (and did!) fail inside of the container.

Remove the line that executes the "ip address" command from your Jenkinsfile.

## Re-establishing our Pipeline

One of the things we had to deal with previously was to setup SSH keys to allow the Jenkins host to export the configuration backups over to the Jumphost. We can't just do this once and be done with it since we are now instantiating a new container each and every time we run a job.

To address this SSH keys issue, add the following to the "Checkout" stage steps:

```
sh 'ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa'
sh 'sshpass -p "ignw" ssh-copy-id -f -i ~/.ssh/id_rsa.pub -o StrictHostKeyChecking=no ignw@10.10.0.254'
```

Next we can build out our stages once again:

```
stage('Run Syntax Checks') {
    steps {
    }
}
stage ('Render Configurations') {
    steps {
    }
}
stage ('Backup Configurations') {
    steps {
    }
}
stage ('Deploy to Dev') {
    steps {
    }
}
stage ('Validate Dev') {
    steps {
    }
}
stage ('Cleanup after Dev') {
    steps {
    }
}
```

One easy way to we can improve our Jenkins Pipeline would be to move all of the syntax checking into its own stage. This not only makes more sense logically, but it will also cause the build to fail very early on into the process if there are any issues. This may save us some time, and it may prevent us from trying to push configurations if there is an issue in a later Playbook.

Add a syntax check for all of the Playbooks:

```
stage('Run Syntax Checks') {
    steps {
        sh 'ansible-playbook generate_configurations.yaml --syntax-check
        sh 'ansible-playbook deploy_configurations.yaml --syntax-check
        sh 'ansible-playbook validate_configurations.yaml --syntax-check
        sh 'ansible-playbook backup_configurations.yaml --syntax-check
        sh 'ansible-playbook replace_configurations.yaml --syntax-check
    }
}
```

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ cat Jenkinsfile
def archive_configs(environment) {
    sh "rsync backup/* ignw@10.10.0.254:/home/ignw/my_network_as_code/archive/$environment/$BUILD_ID"
}

pipeline {
    agent {
        docker {
            image 'my_network_as_code'
            args '--user root'
        }
    }
    stages {
        stage('Checkout and Prepare') {
            steps {
                checkout scm
                sh 'ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa'
                sh 'sshpass -p "ignw" ssh-copy-id -f -i ~/.ssh/id_rsa.pub -o StrictHostKeyChecking=no ignw@10.10.0.254'
            }
        }
        stage('Run Syntax Checks') {
            steps {
                sh 'ansible-playbook generate_configurations.yaml --syntax-check'
                sh 'ansible-playbook deploy_configurations.yaml --syntax-check'
                sh 'ansible-playbook validate_configurations.yaml --syntax-check'
                sh 'ansible-playbook backup_configurations.yaml --syntax-check'
                sh 'ansible-playbook replace_configurations.yaml --syntax-check'
            }
        }
        stage ('Render Configurations') {
            steps {
            }
        }
        stage ('Backup Configurations') {
            steps {
            }
        }
        stage ('Deploy to Dev') {
            steps {
            }
        }
        stage ('Validate Dev') {
            steps {
            }
        }
        stage ('Cleanup After Dev') {
            steps {
            }
        }
    }
}
ignw@ignw-jumphost:~/my_network_as_code$ █

```

Notice anything different/simpler? Because we are using Ubuntu as the base of our container, and because we don't need to worry about Python virtual environments anymore, there is no need to pass in a Python interpreter to Ansible -- the path in our Ansible.cfg file will already be pointing to the appropriate Python binaries in the container!

Let's continue re-building our stages by adding the render and backup steps. Here is another opportunity for us to make a simple, but logical improvement: in the previous Jenkinsfile we only backed up "prod" after the "dev" stages completed successfully -- we may as well just back them up all at once for consistency sake:

```

stage ('Render Configurations') {
    steps {
        sh 'ansible-playbook generate_configurations.yaml'
    }
}
stage ('Backup Configurations') {
    steps {
        sh 'ansible-playbook backup_configurations.yaml'
        archive_configs('dev')
        sh 'ansible-playbook backup_configurations.yaml -i inventory_production'
        archive_configs('prod')
    }
}

```

If we leave our "backup" steps as outlined above we will run into some interesting issues if our build fails and we need to try to roll back. We will have nearly identically named backup files for prod and dev, so we need to address this. We can do this pretty simply by just moving our dev backups into a dev folder and the prod backups into a prod folder.

```

stage ('Backup Configurations') {
    steps {
        sh 'ansible-playbook backup_configurations.yaml'
        archive_configs('dev')
        sh 'mkdir backup_dev'
        sh 'mv backup/* backup_dev/'
        sh 'ansible-playbook backup_configurations.yaml -i inventory_production'
        archive_configs('prod')
        sh 'mkdir backup_prod'
        sh 'mv backup/* backup_prod/'
    }
}

```

That should set us up nicely for our rollback tasks if/when we need them!

Next, our "Deploy to Dev" stage is also very straightforward:

```

stage ('Deploy to Dev') {
    steps {
        sh 'ansible-playbook deploy_configurations.yaml'
    }
}

```

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
}
}
stages {
stage('Checkout and Prepare') {
steps {
    checkout scm
    sh 'ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa'
    sh 'sshpass -p "ignw" ssh-copy-id -f -i ~/.ssh/id_rsa.pub -o StrictHostKeyChecking=no ignw@10.10.0.254'
}
}
stage('Run Syntax Checks') {
steps {
    sh 'ansible-playbook generate_configurations.yaml --syntax-check'
    sh 'ansible-playbook deploy_configurations.yaml --syntax-check'
    sh 'ansible-playbook validate_configurations.yaml --syntax-check'
    sh 'ansible-playbook backup_configurations.yaml --syntax-check'
    sh 'ansible-playbook replace_configurations.yaml --syntax-check'
}
}
stage ('Render Configurations') {
steps {
    sh 'ansible-playbook generate_configurations.yaml'
}
}
stage ('Backup Configurations') {
steps {
    sh 'ansible-playbook backup_configurations.yaml'
    archive_configs('dev')
    sh 'mkdir backup_dev'
    sh 'mv backup/* backup_dev/'
    sh 'ansible-playbook backup_configurations.yaml -i inventory_production'
    archive_configs('prod')
    sh 'mkdir backup_prod'
    sh 'mv backup/* backup_prod/'
}
}
stage ('Deploy to Dev') {
steps {
    sh 'ansible-playbook deploy_configurations.yaml'
}
}
stage ('Validate Dev') {
steps {
}
}
stage ('Cleanup After Dev') {
steps {
}
}
}
}
ignw@ignw-jumphost:~/my_network_as_code$
```

Previously, we handled the validation stages with a simple "if" statement. We can do almost exactly the thing in the declarative syntax, but we need to enclose it in a "script" block. We can also tidy up our script by moving the validation tasks into a function of its own. This could allow for adding of "stage" "qa" "acceptance" or any number of other zones without requiring us to duplicate more code!

To start, create a new function at the top of the Jenkinsfile (right after the archive function). This function should take "environment" as a variable. IF the environment is dev, it should validate the dev, otherwise (for now) it should simply validate the prod environment, finally, we can return the results.

```

def validate_deployment(environment) {
    if (environment == 'dev') {
        result = sh(script: 'ansible-playbook validate_configurations.yaml', returnStatus: true)
    } else {
        result = sh(script: 'ansible-playbook validate_configurations.yaml -i inventory_production', returnStatus: true)
    }
    return result
}

```

Finally, just as before we can check to see what the return value of the script is, and if it is greater than zero (i.e. a bad return code), then we should fail our build:

```

stage ('Validate Dev') {
    steps {
        script {
            result = validate_deployment('dev')
            if (result > 0) {
                currentBuild.result = 'ABORTED'
                error('Integration testing FAILED')
            }
        }
    }
}

```



```

ignw@ignw-jenkins: ~/my_network_as_code
File Edit View Search Terminal Help
}
}
ignw@ignw-jenkins:~/my_network_as_code$ ignw@ignw-jenkins:~/my_network_as_code$ cat Jenkinsfile
def archive_configs(environment) {
    sh "rsync backup/* ignw@10.10.0.254:/home/ignw/my_network_as_code/archive/$environment/$BUILD_ID"
}

def validate_deployment(environment) {
    if (environment == 'dev') {
        result = sh(script: 'ansible-playbook validate_configurations.yaml', returnStatus: true)
    } else {
        result = sh(script: 'ansible-playbook validate_configurations.yaml -i inventory_production', returnStatus: true)
    }
    return result
}

pipeline {
    agent {
        docker {
            image 'my_network_as_code'
            args '--user root'
        }
    }
}

```

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
        sh 'sshpass -p "ignw" ssh-copy-id -f -i ~/.ssh/id_rsa.pub -o StrictHostKeyChecking=no ignw@10.10.0.254'
    }
}
stage('Run Syntax Checks') {
    steps {
        sh 'ansible-playbook generate_configurations.yaml --syntax-check'
        sh 'ansible-playbook deploy_configurations.yaml --syntax-check'
        sh 'ansible-playbook validate_configurations.yaml --syntax-check'
        sh 'ansible-playbook backup_configurations.yaml --syntax-check'
        sh 'ansible-playbook replace_configurations.yaml --syntax-check'
    }
}
stage ('Render Configurations') {
    steps {
        sh 'ansible-playbook generate_configurations.yaml'
    }
}
stage ('Backup Configurations') {
    steps {
        sh 'ansible-playbook backup_configurations.yaml'
        archive_configs('dev')
        sh 'mkdir backup_dev'
        sh 'mv backup/* backup_dev/'
        sh 'ansible-playbook backup_configurations.yaml -i inventory_production'
        archive_configs('prod')
        sh 'mkdir backup_prod'
        sh 'mv backup/* backup_prod/'
    }
}
stage ('Deploy to Dev') {
    steps {
        sh 'ansible-playbook deploy_configurations.yaml'
    }
}
stage ('Validate Dev') {
    steps {
        script {
            result = validate_deployment('dev')
            if (result > 0) {
                currentBuild.result = 'ABORTED'
                error('Integration testing FAILED')
            }
        }
    }
}
stage ('Cleanup After Dev') {
    steps {
    }
}
}
ignw@ignw-jumphost:~/my_network_as_code$ █

```

One last thing before we test this out -- add a simple 'echo' to the "Clean after Dev" task -- Jenkins won't like it if we leave an empty step in our Jenkinsfile!

```

stage ('Clean Up After Dev') {
    steps {
        echo 'clean stuff up here!'
    }
}

```

Since we've made so many changes, let's push the updates to GitHub and check to see if our job is still executing successfully...

```
git add -u  
git commit -m 'updating jenkinsfile'  
git push -u origin master
```

Recall that we were "cleaning" the working directory in our previous Jenkinsfile -- we need to do the same thing here as Jenkins is writing the files to a directory on the host that is mounted in the container.

Due to the fact that we are upgrading our Jenkinsfile syntax we need to go about this in a slightly different way, but the idea is exactly the same. Add the following to the end of your Jenkinsfile, then commit your updates to GitHub.

```
post {  
    always {  
        cleanWs()  
    }  
}
```

You can also delete the "Cleanup After Dev" stage as we won't need it. By moving our backups into "dev" and "prod" folders, we got around the issues we previously faced there.

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
}
}
stage('Run Syntax Checks') {
    steps {
        sh 'ansible-playbook generate_configurations.yaml --syntax-check'
        sh 'ansible-playbook deploy_configurations.yaml --syntax-check'
        sh 'ansible-playbook validate_configurations.yaml --syntax-check'
        sh 'ansible-playbook backup_configurations.yaml --syntax-check'
        sh 'ansible-playbook replace_configurations.yaml --syntax-check'
    }
}
stage ('Render Configurations') {
    steps {
        sh 'ansible-playbook generate_configurations.yaml'
    }
}
stage ('Backup Configurations') {
    steps {
        sh 'ansible-playbook backup_configurations.yaml'
        archive_configs('dev')
        sh 'mkdir backup_dev'
        sh 'mv backup/* backup_dev'
        sh 'ansible-playbook backup_configurations.yaml -i inventory_production'
        archive_configs('prod')
        sh 'mkdir backup_prod'
        sh 'mv backup/* backup_prod'
    }
}
stage ('Deploy to Dev') {
    steps {
        sh 'ansible-playbook deploy_configurations.yaml'
    }
}
stage ('Validate Dev') {
    steps {
        script {
            result = validate_deployment('dev')
            if (result > 0) {
                currentBuild.result = 'ABORTED'
                error('Integration testing FAILED')
            }
        }
    }
}
post {
    always {
        cleanWs()
    }
}
ignw@ignw-jumphost:~/my_network_as_code$ 

```

Re-run your build. It will fail one more time since the clean up action is a *post* action instead of a pre-build action as we had previously. Finally, run the build one more time -- this time things should work a bit better!

```

Jenkins > my_network_as_code > #15
PLAY [Validate "Core" & "Internet" Reachability from Firewall] ****
TASK [asa_command] ****
ok: [acme-sea-asal] => (item=8.8.8)
ok: [acme-sea-asal] => (item=10.255.255.2)
ok: [acme-sea-asal] => (item=10.255.254.1)

PLAY [Validate "Edge" Reachability from Edge Router] ****
TASK [Test Success Pings] ****
ok: [acme-sea-rtr1] => (item=203.0.113.2)

TASK [Test Fail Pings] ****
ok: [acme-sea-rtr1] => (item=10.255.255.2)

PLAY RECAP ****
acme-sea-asal : ok=1    changed=0   unreachable=0   failed=0
acme-sea-nxos1: ok=1    changed=0   unreachable=0   failed=0
acme-sea-rtr1 : ok=2    changed=0   unreachable=0   failed=0

[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Cleanup after Dev)
[Pipeline] sh
[workspace] Running shell script
+ echo tacocat
tacocat
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] cleanWs
[WS-CLEANUP] Deleting project workspace...[WS-CLEANUP] done
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
$ docker stop --time=1 19455dc4ac5a435c0755db106053622d5e104533c409d5243d6800132dfd564a
$ docker rm -f 19455dc4ac5a435c0755db106053622d5e104533c409d5243d6800132dfd564a
[Pipeline] // withDockerContainer
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

Page generated: Jul 12, 2018 3:26:16 PM UTC [REST API](#) [Jenkins ver. 2.107.2](#)

We've got a few things left to add to our Jenkinsfile at this point. Now that we have successfully pushed all the dev configurations out, we need to figure out our rollback setup, as well as add in our stages for prod.

First let's tackle our rollbacks -- keeping in mind we want to be able to have reusable code that can handle more stages than just "dev" and "prod". We can create another function very similar to the one created for the validation step:

```

def replace_configs(environment) {
    if (environment == 'dev') {
        DO STUFF
    } else {
        DO STUFF
    }
}

```

This should look pretty familiar! Next we just need to add in some basic logic to get our replace Playbook ran against the appropriate inventory file. We also need to ensure that we get the backup configurations for the

particular environment back into the "backup" directory (since we moved them earlier so we can more easily distinguish between dev/prod).

```
def replace_configs(environment) {
    if (environment == 'dev') {
        sh 'mv backup_dev/* backup/'
        sh 'ansible-playbook replace_configurations.yaml'
    } else {
        sh 'mv backup_dev/* backup/'
        sh 'ansible-playbook replace_configurations.yaml'
        sh 'rm backup/*'
        sh 'mv backup_prod/* backup/'
        sh 'ansible-playbook replace_configurations.yaml -i inventory_production
    }
}
```

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
ignw@ignw-jumphost:~/my_network_as_code$ cat Jenkinsfile
def archive_configs(environment) {
    sh "rsync backup/* ignw@10.10.0.254:/home/ignw/my_network_as_code/archive/$environment/$BUILD_ID"
}

def validate_deployment(environment) {
    if (environment == 'dev') {
        result = sh(script: 'ansible-playbook validate_configurations.yaml', returnStatus: true)
    } else {
        result = sh(script: 'ansible-playbook validate_configurations.yaml -i inventory_production', returnStatus: true)
    }
    return result
}

def replace_configs(environment) {
    if (environment == 'dev') {
        sh 'mv backup_dev/* backup/'
        sh 'ansible-playbook replace_configurations.yaml'
    } else {
        sh 'mv backup_dev/* backup/'
        sh 'ansible-playbook replace_configurations.yaml'
        sh 'rm backup/*'
        sh 'mv backup_prod/* backup/'
        sh 'ansible-playbook replace_configurations.yaml -i inventory_production'
    }
}

pipeline {
    agent {
        docker {
            image 'my_network_as_code'
            args '--user root'
        }
    }
    stages {
        stage('Checkout and Prepare') {
            steps {
                checkout scm
                sh 'ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa'
                sh 'sshpass -p "ignw" ssh-copy-id -f -i ~/.ssh/id_rsa.pub -o StrictHostKeyChecking=no ignw@10.10.0.254'
            }
        }
        stage('Run Syntax Checks') {
            steps {
                sh 'ansible-playbook generate_configurations.yaml --syntax-check'
                sh 'ansible-playbook deploy_configurations.yaml --syntax-check'
                sh 'ansible-playbook validate_configurations.yaml --syntax-check'
                sh 'ansible-playbook backup_configurations.yaml --syntax-check'
                sh 'ansible-playbook replace_configurations.yaml --syntax-check'
            }
        }
    }
}

```

With that function built, we can simply call the function and pass in the environment if our validation Playbook/script fails:

```
stage ('Validate Dev') {
    steps {
        script {
            result = validate_deployment('dev')
            if (result > 0) {
                replace_configs('dev')
                currentBuild.result = 'ABORTED'
                error('Integration testing FAILED')
            }
        }
    }
}
```

```

ignw@ignw-jumphost: ~/my_network_as_code
File Edit View Search Terminal Help
}
stage('Run Syntax Checks') {
    steps {
        sh 'ansible-playbook generate_configurations.yaml --syntax-check'
        sh 'ansible-playbook deploy_configurations.yaml --syntax-check'
        sh 'ansible-playbook validate_configurations.yaml --syntax-check'
        sh 'ansible-playbook backup_configurations.yaml --syntax-check'
        sh 'ansible-playbook replace_configurations.yaml --syntax-check'
    }
}
stage ('Render Configurations') {
    steps {
        sh 'ansible-playbook generate_configurations.yaml'
    }
}
stage ('Backup Configurations') {
    steps {
        sh 'ansible-playbook backup_configurations.yaml'
        archive_configs('dev')
        sh 'mkdir backup_dev'
        sh 'mv backup/* backup_dev/'
        sh 'ansible-playbook backup_configurations.yaml -i inventory_production'
        archive_configs('prod')
        sh 'mkdir backup_prod'
        sh 'mv backup/* backup_prod/'
    }
}
stage ('Deploy to Dev') {
    steps {
        sh 'ansible-playbook deploy_configurations.yaml'
    }
}
stage ('Validate Dev') {
    steps {
        script {
            result = validate_deployment('dev')
            if (result > 0) {
                replace_configs('dev')
                currentBuild.result = 'ABORTED'
                error('Integration testing FAILED')
            }
        }
    }
}
post {
    always {
        cleanWs()
    }
}
}
ignw@ignw-jumphost:~/my_network_as_code$ █

```

Finally, add in a Deploy and Validate stage for Prod:

```
stage ('Deploy to Prod') {
    steps {
        sh 'ansible-playbook deploy_configurations.yaml -i inventory_production'
    }
}
stage ('Validate Prod') {
    steps {
        script {
            result = validate_deployment('prod')
            if (result > 0) {
                replace_configs('prod')
                currentBuild.result = 'ABORTED'
                error('Integration testing FAILED')
            }
        }
    }
}
```

Commit all of your changes to GitHub and re-execute your build. At this point it should complete successfully!

```

PLAY [Validate "Core" to "Internet" Reachability] ****
TASK [nxos_ping] ****
ok: [acme-sea-nxos1]

PLAY [Validate "Core" & "Internet" Reachability from Firewall] ****
TASK [asa_command] ****
ok: [acme-sea-asal] => (item=8.8.8)
ok: [acme-sea-asal] => (item=10.255.255.2)
ok: [acme-sea-asal] => (item=10.255.254.1)

PLAY [Validate "Edge" Reachability from Edge Router] ****
TASK [Test Success Pings] ****
ok: [acme-sea-rtr1] => (item=203.0.113.2)

TASK [Test Fail Pings] ****
ok: [acme-sea-rtr1] => (item=10.255.255.2)

PLAY RECAP ****
acme-sea-asal : ok=1    changed=0    unreachable=0    failed=0
acme-sea-nxos1: ok=1    changed=0    unreachable=0    failed=0
acme-sea-rtr1 : ok=2    changed=0    unreachable=0    failed=0

[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] cleanWs
[WS-CLEANUP] Deleting project workspace...[WS-CLEANUP] done
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
$ docker stop --time=1 78c463cfa90af22e24bcfda3f8e5bb7c4b6f2572822d45136ad3b3fba5f81625
$ docker rm -f 78c463cfa90af22e24bcfda3f8e5bb7c4b6f2572822d45136ad3b3fba5f81625
[Pipeline] // withDockerContainer
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

There is still a LOT of room for improvement, but this has been a big step toward making a more re-usable and readable Jenkinsfile, as well as making HUGE improvements to how quickly everything executes thanks to our handy Docker container!