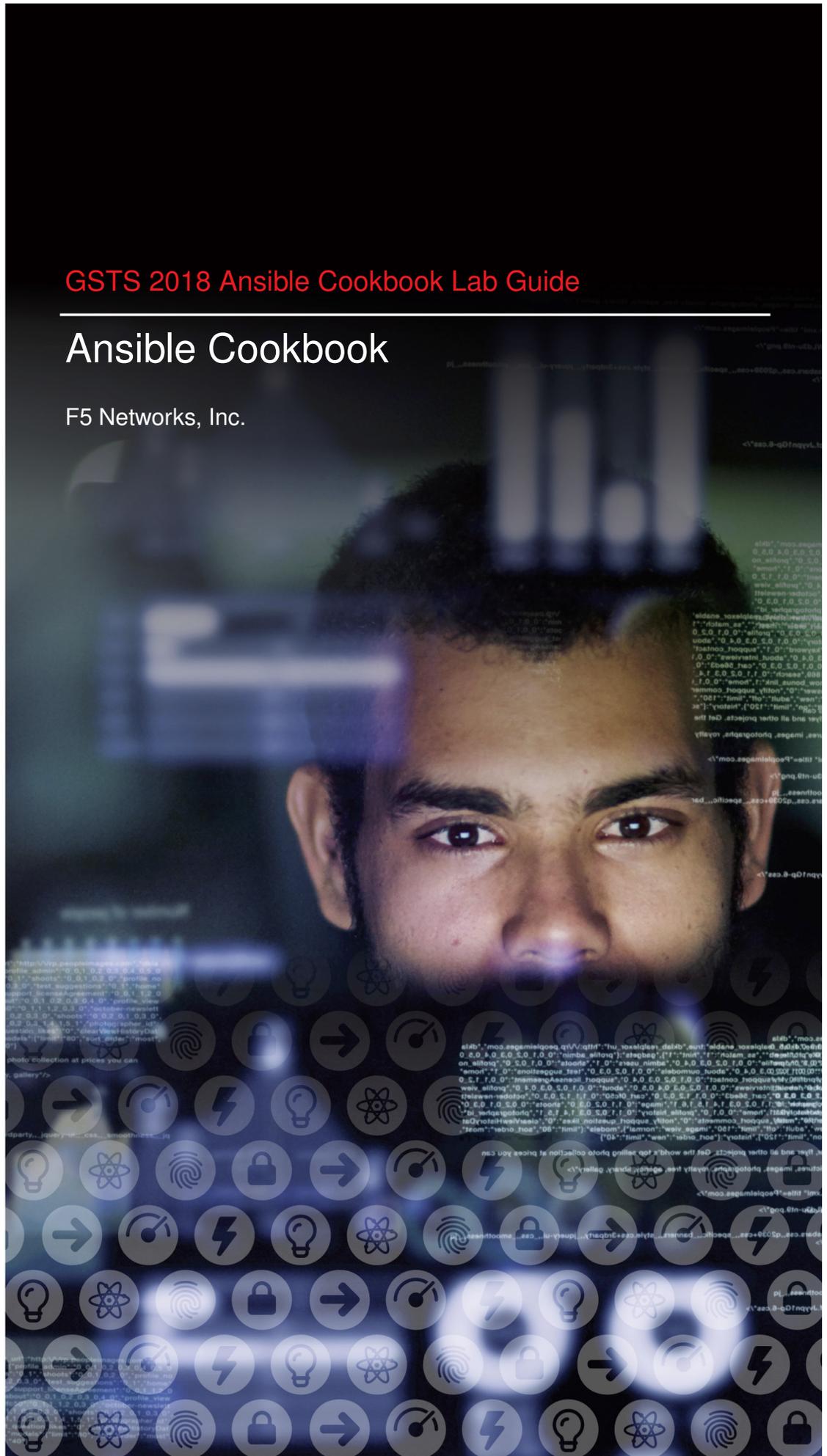




## GSTS 2018 Ansible Cookbook Lab Guide

# Ansible Cookbook

F5 Networks, Inc.





## Contents:

<b>1</b>	<b>BIG-IP Basics (optional)</b>	<b>5</b>
1.1	What is BIG-IP . . . . .	5
1.2	BIG-IP Basic Nomenclature . . . . .	5
1.3	F5 DevCentral BIG-IP Basics Articles . . . . .	5
1.4	Using F5 in Various Environments . . . . .	5
1.5	HA Proxy to BIG-IP Quick Start . . . . .	6
1.6	NGINX to BIG-IP Quick Start . . . . .	6
<b>2</b>	<b>Getting Started</b>	<b>9</b>
2.1	Lab Topology . . . . .	9
<b>3</b>	<b>Class - Ansible Cookbook</b>	<b>11</b>
3.1	Module – Installation and configuration of Ansible . . . . .	11
3.2	Module – Basic BIG-IP administration with Ansible . . . . .	21
3.3	Module – Slightly more advanced Ansible usage . . . . .	39
3.4	Module – Debugging Ansible problems . . . . .	59



## BIG-IP Basics (optional)

Just in case you're new to the F5 BIG-IP platform (or need a refresher) we've included some links and videos below that will help get you started.

### 1.1 What is BIG-IP

Source: <https://devcentral.f5.com/articles/lightboard-lessons-what-is-big-ip-26793>

### 1.2 BIG-IP Basic Nomenclature

Source: <https://devcentral.f5.com/articles/lightboard-lessons-big-ip-basic-nomenclature-26144>

### 1.3 F5 DevCentral BIG-IP Basics Articles

BIG-IP Basics Articles: <https://devcentral.f5.com/articles?tag=devcentral+basics>

### 1.4 Using F5 in Various Environments

- Public Cloud:
  - **AWS/Azure/GCP/etc.:** <http://clouddocs.f5.com/cloud/public/v1/>
- Private Cloud:
  - **OpenStack:** <http://clouddocs.f5.com/cloud/openstack/>
  - **VMware:** <https://f5.com/solutions/technology-alliances/vmware>
- Container Ecosystems:
  - **Cloud Foundry:** <http://clouddocs.f5.com/containers/latest/cloudfoundry/>
  - **Kubernetes:** <http://clouddocs.f5.com/containers/latest/kubernetes>
  - **Mesos Marathon:** <http://clouddocs.f5.com/containers/latest/marathon>

– RedHat OpenShift: <http://clouddocs.f5.com/containers/latest/openshift/>

## 1.5 HA Proxy to BIG-IP Quick Start

If you're already familiar with HA Proxy, learning F5 BIG-IP is straightforward once you learn the associated F5 terminology.

Here is a list of common HA Proxy configuration terminology and its F5 equivalent:

HA Proxy	F5 BIG-IP
Frontend	Virtual Server (VIP)
Backend	Pool
Server	Member
mode http	HTTP Profile
default_backend	Default pool
use_backend	LTM policy
check port	Health monitor

## 1.6 NGINX to BIG-IP Quick Start

If you are already familiar with NGINX, learning F5 BIG-IP will be straightforward once you learn the F5 terminology.

NGINX administrators usually use multiple files and leverage the include command in their config to break down the config and make it easier to manage. F5 leverages *Profiles* which can be applied to a *Virtual Server*.

NGINX uses in-band (passive) health monitors which can be enabled on F5 through the creation of an *inband monitor*. BIG-IP also supports the use of active health monitors, which will poll the pool member periodically. Both can be used together for better monitoring of your services.

F5 BIG-IP supports control-plane and data-plane programmability with:

- Node.js through the use of iRulesLX, iControlLX and iAppsLX
- TCL through the use of iRules and iApp Templates

A lot of the manual configuration and scripting steps that are required with NGINX are supported more easily through various config parameters and profiles in BIG-IP. By leveraging the control-plane programmability features this class covers you can achieve full automation of your services with the BIG-IP platform.

F5 BIG-IP is designed to be a full proxy by default. In most cases there is no need to tune TCP & HTTP buffering like you would on NGINX (i.e. using `proxy_buffering`). This is because the default settings have been optimized and can adapt to most situations.

Here is a list of common NGINX configuration terminology and its F5 equivalent:

<b>NGINX</b>	<b>F5 BIG-IP</b>
listen	Virtual Server Port (VIP)
upstream	Pool
proxy_pass	Default Pool
server	Member
ssl_certificate	SSL Profile Option
return	LTM HTTP Policy Option
proxy_set_header X Forwarded For	HTTP Profile Option Insert X-Forwarded-For
proxy_set_header	LTM HTTP Policy Option
add_header	LTM HTTP Policy Option
location & proxy_pass	LTM HTTP Policy Option
Proxy Cache	Web Acceleration Policy



---

**Note:** All work for this lab will be performed exclusively from the Ansible controller. No installation or interaction with your local system is required.

---

## 2.1 Lab Topology

The following components have been included in your lab environment:

- 1 x F5 BIG-IP VE (v13.0)
- 1 x Linux Server (ubuntu 16.04 LTS)
- 1 x Linux Client (ubuntu 16.04 LTS)
- 1 x Ansible Controller (ubuntu 16.04 LTS)

### 2.1.1 Lab Components

The following table lists VLANs, IP Addresses and Credentials for all components:

Component	VLAN/IP Address(es)	Credentials
big-ip01	<ul style="list-style-type: none"><li>• <b>Management:</b> 10.1.1.4</li><li>• <b>External:</b> 10.1.10.10</li><li>• <b>Internal:</b> 10.1.20.10</li></ul>	admin/admin
client	<ul style="list-style-type: none"><li>• <b>Management:</b> 10.1.1.5</li><li>• <b>External:</b> 10.1.10.11</li></ul>	root/default
server	<ul style="list-style-type: none"><li>• <b>Management:</b> 10.1.1.6</li><li>• <b>Internal:</b> 10.1.20.11</li></ul>	root/default
controller	<ul style="list-style-type: none"><li>• <b>Management:</b> 10.1.1.7</li></ul>	root/default

## 2.1.2 Lab Environments

In order to complete this class you will need to utilize a specific **Lab Environment**. You can consume this training in the following ways:

- Pre-built Environment using the F5 Unified Demo Framework (UDF)
  - This environment is currently available for F5 employees only

Select the Environment from the list below to get started:

### F5 Unified Demo Framework (UDF)

---

**Note:** This environment is currently available for F5 employees only

---

Determine how to start your deployment:

- **Official Events (ISC, SSE Summits):** Please follow the instructions given by your instructor to join the UDF Course.
- **Self-Paced/On Your Own:** Login to UDF, *Deploy* the `Ansible Cookbook` Blueprint and *Start* it.

### Connecting to the Environment

To connect to the lab environment we will use a Web Shell to connect to the Ansible Controller.

#### Connect using Web Shell

1. In the UDF navigate to your *Deployments*
2. Click the *Details* button for your Deployment
3. Click the *Components* tab
4. Find the `Ansible Controller` Component and click the *Access* button. Then click the *WEB SHELL* option. A new browser window/tab will be opened.
5. Ensure that the `f5-gsts-labs-ansible-cookbook` directory in your `/root` directory is up-to-date.

This can be done with the following command

```
cd /root/f5-gsts-labs-ansible-cookbook && git pull
```

6. Select how you would like to continue:
  - Review: *BIG-IP Basics (optional)*
  - Start: *Module – Installation and configuration of Ansible*

## Class - Ansible Cookbook

This class covers the following topics:

- Installation and configuration of Ansible
- Basic BIG-IP administration with Ansible
- Slightly more advanced Ansible usage
- Debugging Ansible problems

These topics are arranged as a series of recipes, in much the same way as the O'Reilly Cookbook series.

Expected time to complete: **2 hours**

### 3.1 Module – Installation and configuration of Ansible

The first step in using Ansible with an F5 product is to ensure that you have done the absolute minimum required to make Ansible work.

The recipes in this chapter look at methods for installing and configuring Ansible. We cover things such as installing the tool, establishing the correct directory layouts, and creating necessary files.

We will focus on using Ansible 2.5 for this class. This specific version of Ansible, at the time of this writing, is not officially released. This will not be a problem for us, and will also serve to prepare you for what is coming in the future.

#### 3.1.1 Installing Ansible

##### Problem

You need to install Ansible in an existing Linux environment

##### Solution

Ansible is distributed in several ways. These include

- Via the system's package manager

- Via the PyPI (pronounced “pie pee eye”) package repository
- Via source tarball

The only proper way to install Ansible is via PyPI using the `pip` command line tool.

```
pip install ansible
```

For the remainder of these labs we will be using the development copy of Ansible.

Since this is not yet available, we’ll install it directly from Github

```
$ pip install --upgrade git+https://github.com/ansible/ansible.git
```

This will include an updated set of modules that will be released in March.

## Discussion

PyPI is considered the only correct way to install Ansible because it is the only method that the Ansible developers themselves can control.

The packages that you find on Linux distributions such as Ubuntu, Fedora, or CentOS are maintained by members of the Ansible community and not by Ansible itself.

Additionally, the packages that ship with your operating system are frequently out-of-date.

True, they may be current at the time of their release, but Ansible’s release cycle is quarterly, and therefore they can become out of date quickly.

The `pip` method of installing is not constrained to the demands of the Linux maintainers; it exists outside of their control. Therefore, it is the easiest way to get the most up-to-date software from Ansible.

One more concern with the Linux packages is that they typically place files in a location different from where `pip` places them. This is totally expected, but it can have frustrating consequences should you choose to switch to the `pip` version (for example, to upgrade to a more recent version).

The differences in file locations can conflict with each other and leave your Ansible installation a complete mess. Best to just stick with `pip`.

## 3.1.2 Installing module dependencies

### Problem

You need to install F5 Ansible module dependencies

### Solution

Each module has different requirements. The F5 Ansible modules require the following PyPI packages

- `f5-sdk`
- `bigsuds`
- `netaddr`
- `objectpath`
- `isoparser`
- `lxml`

- deepdiff

These can be installed with the `pip` command

```
pip install f5-sdk bigsuds netaddr objectpath isoparser lxml deepdiff
```

## Discussion

Unfortunately, there is no way to install all dependencies for all modules out of the box.

Instead, you must find the dependencies for the module you are interested in, and install them manually. This can be done by either,

1. Using the `ansible-doc` command
2. By visiting the Ansible documentation page for the module.

Take `bigip_selfip` for example.

### ansible-doc command

The `ansible-doc` command to view the requirements is,

```
ansible-doc bigip_selfip
```

The requirements are shown in the output of this command.

```
- vlan
    The VLAN that the new self IPs will be on. When creat
    [Default: (null)]

NOTES:
    * Requires the f5-sdk Python package on the host. This
    * Requires the netaddr Python package on the host.
    * For more information on using Ansible to manage F5 Ne
    https://www.ansible.com/ansible-f5.

REQUIREMENTS: netaddr, f5-sdk
AUTHOR: Tim Rupp (@caphrim007)
EXTENDS_DOCUMENTATION_FRAGMENT: f5
METADATA:
```

You may need to scroll to find this information.

## Visiting documentation page

Alternatively you can visit the docs for this module by navigating to [this link](#)

There is a direct link to the requirements list if you mouse over the **Requirements** header

# bigip\_selfip - Manage Self-IPs on a BIG-IP system

New in version 2.2.

- [Synopsis](#)
- [Requirements \(on host that executes module\)](#)
- [Options](#)
- [Examples](#)
- [Return Values](#)
- [Notes](#)
  - [Status](#)

## Synopsis

- Manage Self-IPs on a BIG-IP system



## Requirements (on host that executes module)

- netaddr
- f5-sdk

## Options

parameter	required	default	choices	comment
address	no			The IP addresses for the new self IP. This value is themselves cannot be changed after they are cre:
allow_service	no			Configure port lockdown for the Self IP. By defau This can be changed to allow TCP and UDP ports

Note the chain icon to the right of the header. That link will [lead you here](#).

## Installing a development copy of F5 SDK

One behavior that is frequently done is the installation of a development copy of the F5 Python SDK. This is usually safe to do as the SDK is always in-line with the Ansible modules.

To do this, run the following command:

```
pip install --upgrade git+https://github.com/F5Networks/f5-common-python.git
```

This is usually a required step for Ansible upgrades and future releases of Ansible because we often include new APIs in the SDK that Ansible will make use of.

### 3.1.3 Expected File Layout

#### Problem

You need to know how you should arrange files on disk so that Ansible can find them

#### Solution

You should create the following directory structure when using Ansible.

```
.
?- ansible.cfg
?- inventory
|   ?- group_vars
|   |   ?- all.yaml
|   ?- host_vars
|   |   ?- host1.yaml
|   ?- hosts
?- library
?- playbooks
|   ?- site.yaml
?- files
?- roles
?- scripts
?- templates
```

The above assumes the following,

- you have a single host named `host1`
- you have a single playbook named `site.yaml`

More should be added as necessary. Empty directories are not required.

#### Discussion

Each directory in Ansible has a specific purpose. You may not use all of these directories in your day-to-day work, and that's fine. You can remove empty directories as needed.

In its simplest format, Ansible requires only two files to work; an inventory and a playbook. As indicated in the solution above, we do not recommend you follow that design.

Until you have sufficient knowledge of how Ansible's parts work, it is better that you use the solution above so that any modules you may use (in any place you use them) will work.

Directories that are optional are,

- `files`
- `library`
- `roles`
- `scripts`
- `templates`

The purpose of each directory is the following,

- `files`

- contains non-templates files to be used by the `copy` module
- `library`
  - contains third-party Ansible modules that you want to use in your playbook
- `roles`
  - contains roles that you want to use in your playbook
- `scripts`
  - contains shell scripts that will be referenced by the `script` module
- `templates`
  - contains files that will be treated as templates and referenced by the `template` module.

### 3.1.4 Installing unstable modules

#### Problem

You need to install an unstable F5 Ansible module

#### Solution

The procedure for this is [documented here](#). We will use `bigip_software` for this example.

Ensure that you have created a directory named `library` as shown in [1.3 Expected File Layout](#).

Next, download the source for this module using `curl`

```
curl -o library/bigip_software.py https://raw.githubusercontent.com/F5Networks/f5-  
↪ansible/devel/library/bigip_software.py
```

You can now use the module as documented in its examples.

#### Discussion

Our unstable code exists for the following reasons,

- We do not want to upstream everything. This may be because the underlying product is immature or not fully supported
- Due to the above, we can't put it into upstream Ansible. If we did, this would create a support liability for us.
- It allows us to work independently of anything Ansible does.

Will you need to get unstable code? Probably.

In many cases, the unstable code is just as good as what exists in Ansible today, but you won't know this unless you try to use it.

If you find a module in the unstable branch that is not in the stable (Ansible upstream) product, you will want to let us know about this by [filing an issue](#).

### 3.1.5 Tweaking local ansible.cfg

#### Problem

You need to tell Ansible how to find your unreleased modules

#### Solution

Create, or change, an `ansible.cfg` file that specifies the `library` setting.

I recommend that you put an `ansible.cfg` file at the top level of your Ansible related work.

Then add the following line to the `ansible.cfg` file

```
library = ./library
```

#### Discussion

There are a number of settings that you can change in an `ansible.cfg` file. The entire list is [shown here](#).

Amongst the list of things that I routinely change, are the following

- `retry_files_enabled = False`
- `host_key_checking = False`
- `roles_path = ./roles`
- `library = ./library`

Values for paths (such as `roles_path` and `library` can be separated by a colon. For example,

```
roles_path = ./roles-dir-1:/path/to/absolute-dir2
```

I **never** use the system config found at `/etc/ansible/ansible.cfg`. This is an **anti-pattern**, do not do it. Instead, put your changes for you specific project in a config file found in your project's top-level directory.

If you use the system file, it will affect all the users of the system and all the uses of Ansible on the system. This is almost never what you want.

### 3.1.6 Using static inventory

#### Problem

You need to have Ansible communicate with a predefined list of hosts

#### Solution

Use a static inventory file.

A static inventory file is a INI formatted file. Here is an example

```
server ansible_host=10.1.1.6
bigip ansible_host=10.1.1.4
client ansible_host=10.1.1.5
```

The above text you be put in a file named `hosts` in the `inventory` directory.

You would use the inventory like so,

```
ansible-playbook -i inventory/hosts playbooks/site.yaml
```

1. Create a `lab1.6` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `server` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.6`
4. Add a `client` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.5`
5. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`

## Discussion

Static hosts are the original means of specifying an inventory to Ansible.

The format mentioned in the solution above includes the following information,

1. A host named `bigip`. This value will be put in Ansible's `inventory_hostname` variable.
2. A host *fact* called `ansible_host`. This is a reserved variable in Ansible. It is used by Ansible to connect to the remote host. Its value is `10.1.1.4`.

There are many more forms of inventory than static lists. Indeed, you can also provide dynamic lists that take the form of small programs which output specially formatted JSON.

Static lists work well for demos, ad-hoc play running, and cases when your organizations systems practically never change. Otherwise, a dynamic source is probably better.

Dynamic sources must be written by hand if you require a specific means of getting the host informations (for example, from a local database at your company).

There are also a number of dynamic resources that you can get from Ansible. You can find [Community contributions here](#), and you can find Contributions that [ship with Ansible, here](#).

## 3.1.7 Installing software with apt

### Problem

You need to install apache using the on an Ubuntu host

### Solution

Use the `apt` module.

1. Create a `lab1.7` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Change the `playbooks/site.yaml` file to resemble the following.
4. Add a `server` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.6`

```

---
- name: An example install playbook
  hosts: server

  tasks:
    - name: Install apache
      apt:
        name: apache2
        update_cache: yes

```

Run this playbook, from the `lab1.7` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

There are different package managers for the different distributions of Linux that exist.

In this case, we are using the `apt` package manager because we are on a Debian/Ubuntu based system. On systems such as Fedora or CentOS we would use the `yum`, or `dnf`, module to install similar packages.

Be aware that the name of a package *will* change depending on the package manager being used.

## 3.1.8 Writing general files to a remote device

### Problem

You need to write the contents of a file (literal) to a remote location

### Solution

Use the `copy` module.

1. Create a `lab1.8` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Change the `playbooks/site.yaml` file to resemble the following.
4. Add a `server` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.1.6`

```

---
- name: An example copy playbook
  hosts: server

  tasks:
    - name: Copy a local file to the remote system
      copy:
        src: ../files/sample-download.txt
        dest: /var/www/html/sample-download.txt

```

This playbooks requires a file named `sample-download.txt` be created in the `files` directory of your lab. Therefore, create this file. You can put in it any text you want. How about,

```
This was uploaded by Ansible
```

Run this playbook, from the `lab1.8` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

This module will take a given file, and put it on a remote system at the destination that you specify.

This module is idempotent. That means that if the remote file exists, it will **not** overwrite it upon subsequent runs of the playbook.

This module, like all of the standard Ansible modules, works over SSH. Therefore, the accounts used will be those implicitly used by Ansible unless you specify otherwise.

Ansible will SSH as the user running the playbook (by default) and use the SSH public key for that user (by default).

Default Ansible modules (those that use SSH) will work on BIG-IP versions  $\geq 12.0.0$ . They require though that your SSH user be configured to use the “advanced” shell. They will not work using the `tmsh` shell.

## 3.1.9 Templating a file to a remote device

### Problem

You need to write the contents of a file (containing variables) to a remote location

### Solution

Use the `template` module.

1. Create a `lab1.9` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Change the `playbooks/site.yaml` file to resemble the following.
4. Add a `server` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.6`

```
---
- name: An example template playbook
  hosts: server

  tasks:
    - name: Template a file to disk
      template:
        src: ../templates/sample-template.txt
        dest: /tmp/sample-template.txt
```

This playbooks requires a file named `sample-template.txt` be created in the `templates` directory of your lab. Therefore, create this file. You can put in it any text you want. How about,

```
This was uploaded by Ansible. The remote machine info is,  
name: {{ inventory_hostname }}  
ip: {{ ansible_host }}
```

Run this playbook, from the `lab1.9` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

This module will take a given file, and put it on a remote system at the destination that you specify.

This module is idempotent. That means that if the remote file exists, it will **not** overwrite it upon subsequent runs of the playbook.

This module, like all of the standard Ansible modules, works over SSH. Therefore, the accounts used will be those implicitly used by Ansible unless you specify otherwise.

Ansible will SSH as the user running the playbook (by default) and use the SSH public key for that user (by default).

Default Ansible modules (those that use SSH) will work on BIG-IP versions  $\geq 12.0.0$ . They require though that your SSH user be configured to use the “advanced” shell. They will not work using the `tmsh` shell.

## 3.2 Module – Basic BIG-IP administration with Ansible

Once you have installed and configured Ansible, you will want to move on to doing basic administrative tasks on the BIG-IP.

The following recipes target a common subset of work that people typically undertake when configuring BIG-IP devices. Among the various recipes included in this chapter are means to create pools and virtual servers, provision modules, and manage users and partitions.

We will also see recipes for modules that are used in nearly every playbook.

### 3.2.1 Creating a pool on BIG-IP

#### Problem

You need to create a pool on a BIG-IP

#### Solution

Use the `bigip_pool` module.

1. Create a `lab2.1` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example pool playbook
  hosts: bigip
  connection: local

  tasks:
    - name: Create web servers pool
      bigip_pool:
        name: web-servers
        lb_method: ratio-member
        password: admin
        server: 10.1.1.4
        user: admin
        validate_certs: no
```

Run this playbook, from the `lab2.1` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

The `bigip_pool` module can configure a number of attributes for a pool. At a minimum, the `name` is required.

This module is idempotent. Therefore, you can run it over and over again and so-long as no settings have been changed, this module will report no changes.

Notice how we also included the credentials to log into the device as arguments to the task. This is *not* the preferred way to do this, but it illustrates a way for beginners to get started without needing to know a less obvious way to specify these values.

The module has several more options, all of which can be seen at [this link](#). I have reproduced them below. These are relevant to the 2.5 release of Ansible.

- `description`
- `lb_method`
- `monitor_type`
- `monitors`
- `name`
- `quorum`
- `reselect_tries`
- `service_down_action`
- `slow_ramp_time`

## 3.2.2 Writing once, re-using many times

### Problem

You want to specify the values for `user/pass` and `validate_certs` only once but re-use them throughout your tasks

## Solution

Use variables.

1. Create a `lab2.2` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example copy playbook
  hosts: bigip

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Create many pools
      bigip_pool:
        name: web-servers
        lb_method: ratio-member
        password: "{{ password }}"
        server: 10.1.1.4
        user: "{{ username }}"
        validate_certs: "{{ validate_certs }}"
```

Run this playbook, from the `lab2.2` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

Variables are one of the ways in which you can set a value once and re-use it across many tasks in your Play.

It should be noted that variables **do not** survive across Plays. Therefore, if you need to use them in multiple plays, it is better to put them in a `host_vars` or `group_vars` file.

Variables are identified by their double curly braces (`{{` and `}}`). The value in-between these braces is the variable name.

Notice how we set our variables at the top of the play in the `vars` section. This is a special section of the Playbook where you can specify variable data that will be used across this Play and this Play only.

When using variables, they *must* be wrapped in double quotes. You can see this in the `bigip_pool` task for the `password`, `user`, and `validate_certs` arguments.

## 3.2.3 Creating a physical node

### Problem

You need to create a node which you will assign to a pool.

### Solution

Use the `bigip_node` module.

1. Create a `lab2.3` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. Type the following into the `playbooks/site.yaml` file.

```
---
- name: An example virtual server playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Create node for physical machine
      bigip_node:
        address: 10.1.20.11
        name: server
        password: "{{ password }}"
        server: 10.1.1.4
        user: "{{ username }}"
        validate_certs: "{{ validate_certs }}"
```

Run this playbook, from the `lab2.3` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

### Discussion

The `bigip_node` module can configure physical device addresses that can later be added to pools. At a minimum, the `name` is required. Additionally, either the `address` or `fqdn` parameters are also required when creating new nodes.

This module can take hostnames using the `fqdn` parameter. You may not specify both the `address` and `fqdn`.

## 3.2.4 Adding nodes to a pool

### Problem

You need to assign newly created nodes to a pool

### Solution

Use the `bigip_pool_member` module.

1. Create a `lab2.4` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. Type the following into the `playbooks/site.yaml` file.

```
---
- name: An example pool members playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Add nodes to pool
      bigip_pool_member:
        description: webserver-1
        host: 10.1.20.11
        name: server
        password: "{{ password }}"
        pool: web-servers
        port: 80
        server: 10.1.1.4
        user: "{{ username }}"
        validate_certs: "{{ validate_certs }}"
```

Run this playbook, from the `lab2.4` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

### Discussion

The `bigip_pool_member` module can configure pools with the details of existing nodes. A node that has been placed in a pool is referred to as a “pool member”.

At a minimum, the `name` is required. Additionally, the `host` is required when creating new pool members.

## 3.2.5 Creating a virtual server on BIG-IP

### Problem

You need to create a virtual server, associated with a pool, on a BIG-IP

### Solution

Use the `bigip_virtual_server` module.

1. Create a `lab2.5` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. Type the following into the `playbooks/site.yaml` file.

```
---
- name: An example virtual server playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Create web server VIP
      bigip_virtual_server:
        description: webserver-vip
        destination: 10.1.1.100
        password: "{{ password }}"
        name: vip-1
        pool: web-servers
        port: 80
        server: 10.1.1.4
        snat: Automap
        user: "{{ username }}"
        profiles:
          - http
          - clientssl
        validate_certs: "{{ validate_certs }}"
```

Run this playbook, from the `lab2.5` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

### Discussion

The `bigip_virtual_server` module can configure a number of attributes for a virtual server. At a minimum, the `name` is required.

This module is idempotent. Therefore, you can run it over and over again and so-long as no settings have been changed, this module will report no changes.

Several arguments, such as `policies` and `profiles` take a list of values. If you update this list of values, it will be reflected on the virtual server's configuration. This includes removing items from these lists.

As an example, if you have four items in the `profile` list, and then you remove one, this will cause the virtual server to be reconfigured to only have three profiles.

## 3.2.6 Installing an iApp template on BIG-IP

### Problem

You need to install an App Services Integration iApp

### Solution

Use the `bigip_iapp_template` module.

1. Change into the `lab2.6` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. Type the following into the `playbooks/site.yaml` file.

```
---
- name: An example iApp template playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Add the iApp
      bigip_iapp_template:
        content: "{{ lookup('file', 'appsvcs_integration_v2.0.004.tmpl') }}"
        password: "{{ password }}"
        server: 10.1.1.4
        state: present
        user: admin
```

Run this playbook, from the `lab2.6` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

### Discussion

The `bigip_iapp_template` module can manage the TCL iApps that are installed on the remote BIG-IP.

Most arguments to the module are unnecessary because the module will attempt to parse the iApp itself to determine the necessary values.

Nevertheless, if you do provide the values, they will **override** what is in content of the iApp itself.

## 3.2.7 Creating an HTTP service from the HTTP iApp

### Problem

You need to create a service from the HTTP iApp

### Solution

Use the `bigip_iapp_service` module.

1. Change to `lab2.7` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example iApp service playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Add the iApp template
      bigip_iapp_template:
        content: "{{ lookup('file', '../files/f5.http.v1.2.0rc4.tmpl') }}"
        password: "{{ password }}"
        server: 10.1.1.4
        state: present
        user: admin

    - name: Add the iApp Service
      bigip_iapp_service:
        name: http-iapp1
        template: f5.http.v1.2.0rc4
        password: "{{ password }}"
        server: 10.1.1.4
        validate_certs: "{{ validate_certs }}"
        state: present
        user: "{{ username }}"
        parameters:
          lists:
            - name: irules__irules
              value:
```

```

tables:
  - name: basic__snatpool_members
  - name: net__snatpool_members
  - name: optimizations__hosts
  - name: pool__hosts
    columnNames:
      - name
    rows:
      - row:
          - internal.company.bar
  - name: pool__members
    columnNames:
      - addr
      - port
      - connection_limit
    rows:
      - row:
          - ""
          - 80
          - 0
  - name: server_pools__servers
variables:
  - name: var__vs_address
    value: 1.1.1.1
  - name: pm__apache_servers_for_http
    value: 2.2.2.1:80
  - name: pm__apache_servers_for_https
    value: 2.2.2.2:80
  - name: client__http_compression
    value: "/#create_new#"
  - name: monitor__monitor
    value: "/#create_new#"
  - name: monitor__uri
    value: "/"
  - name: net__client_mode
    value: wan
  - name: net__server_mode
    value: lan
  - name: pool__addr
    value: 10.10.10.10
  - name: pool__pool_to_use
    value: "/#create_new#"
  - name: pool__port
    value: 80
  - name: ssl__mode
    value: no_ssl
  - name: ssl_encryption_questions__advanced
    value: no
  - name: ssl_encryption_questions__help
    value: hide

```

Run this playbook, from the lab2.7 directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

The `bigip_iapp_service` module can manage the iApp services that are on the remote BIG-IP.

The easiest way to provide data to this module is in the form of a content `lookup`, providing the path to a file containing the `parameters` argument.

To use that approach would require a JSON file and a specific format of Task in your Playbook. An example is below.

```
- name: Add the iApp
  bigip_iapp_service:
    name: http-iapp2
    template: f5.http
    password: "{{ password }}"
    server: 10.1.1.4
    validate_certs: "{{ validate_certs }}"
    state: present
    user: "{{ username }}"
    parameters: "{{ lookup('file', '../files/http-iapp-parameters.json') }}"
```

Observe how we changed the parameters to use a lookup instead of providing the YAML format.

The syntax for a lookup is similar to normal Ansible variables, in that it is wrapped in `{{ and }}`. It differs though in its use a the following command.

- `lookup('file', '/path/to/file')`

You can read this in the same way you might read a function in a programming language.

The `lookup` word is the same of a method that Ansible makes available to you. Next, is the word `file` wrapped in quotes. This is a *type* of lookup. There are many types of lookups that you can use. Finally is the path on the filesystem that you want to look up. That is in the `/path/to/file/` value; also wrapped in quotes.

The parentheses ( and ) are also important, and required, in the places that you see them.

Configure the `lab2.7/playbooks/site.yaml` above to replace your existing task with the task in the Discussion. Run the playbook as you did earlier. You should observe similar behavior as before, except a different iApp service, `http-iapp2` should now exist.

Also, yes, in the solution's example, the `parameters` argument really looks like that; the iApp service data structures them self are responsible for that. We (F5 Ansible modules) may be able to improve upon this in the future.

## 3.2.8 Provisioning ASM

### Problem

You need to provision ASM on the BIG-IP

### Solution

Use the `bigip_provision` module.

1. Create a `lab2.8` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.

3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. Type the following into the `playbooks/site.yaml` file.

```
---
- name: An example provision playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Provision ASM
      bigip_provision:
        name: asm
        password: "{{ password }}"
        server: 10.1.1.4
        validate_certs: "{{ validate_certs }}"
        user: "{{ username }}"
```

Run this playbook, from the `lab2.8` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

The `bigip_provision` module can provision and de-provision modules from the system.

This module will wait for a provisioning action to fully complete before it allows the Playbook to proceed to the next task. This includes waiting for the system to reboot and for MCPD to come online and be ready to take new configuration.

All of the above also applies to ASM.

The level that all modules are provisioned at is `nominal` by default. This can be changed using the `level` argument. Valid choices are,

- `dedicated`
- `nominal`
- `minimum`

This module is smart enough to know when certain modules require specific provisioning levels. For example, `vCMP` is always `dedicated`.

### 3.2.9 Applying an ASM policy

#### Problem

You need to apply an ASM policy to the BIG-IP

## Solution

Have on-hand an ASM policy in one of the following formats

- Compact
- Non-compact
- Binary

Use the `bigip_asm_policy` to put the Policy on the BIG-IP and activate it.

---

**Note:** You will still need to add this policy to a virtual server using the `bigip_virtual_server` module.

---

1. Change into the `lab2.9` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example ASM policy playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Create ASM policy, compact XML file
      bigip_asm_policy:
        name: foo-policy
        file: ../files/v2_policy_compact.xml
        active: yes
        user: "{{ username }}"
        password: "{{ password }}"
        server: 10.1.1.4
        validate_certs: "{{ validate_certs }}"
```

Run this playbook, from the `lab2.9` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

Uploading and applying ASM policies is as easy as just specifying the policy you want to put on the device.

This module supports all of the types of policies that you can put on a device. It will also support putting ASM policies on older versions of BIG-IP (they changed things in or around 12.1.0)

Obviously, policies created and exported on newer releases of BIG-IP are not backwards compatible with older releases of BIG-IP.

## 3.2.10 Creating an LTM policy with rules

### Problem

You need to create an LTM policy with an ASM rule on a BIG-IP

### Solution

Use the `bigip_policy` module to create a policy with a generic rule. Then use the `bigip_policy_rule` module to modify the `actions` and `conditions` on that rule as needed.

1. Change into the `lab2.10` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example LTM policy playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin
    policy_name1: my-ltm-policy

  tasks:
    - name: Provision ASM
      bigip_provision:
        module: asm
        password: "{{ password }}"
        server: 10.1.1.4
        validate_certs: "{{ validate_certs }}"
        user: "{{ username }}"

    - name: Create ASM policy
      bigip_asm_policy:
        name: foo-policy
        file: ../files/v2_policy_compact.xml
        password: "{{ password }}"
        server: 10.1.1.4
        validate_certs: "{{ validate_certs }}"
        user: "{{ username }}"

    - name: Create published policy with 1 stubbed rule
      bigip_policy:
        name: "{{ policy_name1 }}"
        state: present
        rules:
          - rule1
        password: "{{ password }}"
        server: 10.1.1.4
```

```

    validate_certs: "{{ validate_certs }}"
    user: "{{ username }}"

- name: Attach ASM policy to LTM policy rule
  bigip_policy_rule:
    policy: "{{ policy_name1 }}"
    name: rule1
    actions:
      - type: enable
        asm_policy: foo-policy
    password: "{{ password }}"
    server: 10.1.1.4
    validate_certs: "{{ validate_certs }}"
    user: "{{ username }}"

```

Run this playbook, from the `lab2.10` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

The `bigip_policy` module is used for several purposes.

First, it creates the containers that can actually contain rules.

Second, it is used to stub out lists of rules before you actually configure those rules. This is handy when you need to arrange the rule order of the policy. Since rules can be applied in a specific order, using the `bigip_policy` module can set that order (using stub rules) before you actually go about creating the rules.

If you create rules *later*, they will *always* be appended to the list of current rules. Obviously this may not be what you want, so the `bigip_policy` module can be used to re-arrange them. Just specify the list of rules in the order you want them applied.

At the time of this writing, only a handful of conditions and actions are available for use in the `bigip_policy_rule` module. You may file an issue, if you need a particular condition or action added.

Available conditions types are,

- `http_uri`
- `all_traffic`

Available actions types are

- `forward` (this is used in conjunction with pools)
- `enable` (this is used in conjunction with ASM policies)
- `ignore`

In addition to these types, there is also (usually) a value that you will supply so that a particular type can take effect. These are all documented in the **ansible-doc** for the `bigip_policy_rule` module.

Some of them are

- `path_begins_with_any`
- `asm_policy`
- `pool`

The documentation outlines which values to specify in which cases.

### 3.2.11 Creating a new partition

#### Problem

You need to create separate partitions on the BIG-IP for different tenants or resource management

#### Solution

Use the `bigip_partition` module.

1. Create a `lab2.11` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example partition playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Create partition
      bigip_partition:
        name: my-partition
        password: "{{ password }}"
        server: 10.1.1.4
        validate_certs: "{{ validate_certs }}"
        user: "{{ username }}"
```

Run this playbook, from the `lab2.11` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

#### Discussion

The `bigip_partition` module can manage partitions on the system.

Partitions can be used in other modules after they are created. To use them in modules that support them, provide the `partition` parameter.

Some modules, such as `bigip_selfip` allow you to modify resources that can exist in another partition. In you want to do this, name those resources explicitly using their full path (i.e., `/foo/vlan1`). If you do not name the full path, the module in question will assume the partition that is supplied in the `partition` argument. By default, this is `Common`.

At the time of this writing, partitions can **not** be removed until all of the resources under them have been removed. We realize this is a source of pain, but there is truly no supported way of removing a partition and all of its resources. A future update will provide a workaround.

### 3.2.12 Saving your configuration

#### Problem

You need to save the running configuration of a BIG-IP

#### Solution

Use the `bigip_config` module.

1. Create a `lab2.12` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example configuration saving playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Save running configuration
      bigip_config:
        save: yes
        password: "{{ password }}"
        server: 10.1.1.4
        validate_certs: "{{ validate_certs }}"
        user: "{{ username }}"
```

Run this playbook, from the `lab2.12` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

#### Discussion

The `bigip_config` module has several purposes, one of which is to save your configuration.

In addition to this, you can merge an existing configuration that you might have (in the SCF format) into the running configuration using the `merge_content` argument..

You can also reset the running configuration, should you so desire, using the `reset` argument.

### 3.2.13 Waiting for your device to (re)boot

#### Problem

You need to reboot the BIG-IP and wait for it to come back up

#### Solution

Reboot the device with `bigip_command`, then use `bigip_wait` to wait for the device to come back up and be ready to take configuration.

1. Create a `lab2.13` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. Type the following into the `playbooks/site.yaml` file.

```
---
- name: An example configuration saving playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Reboot BIG-IP
      bigip_command:
        commands: tmsh reboot
        user: "{{ username }}"
        password: "{{ password }}"
        server: 10.1.1.4
        validate_certs: "{{ validate_certs }}"
        ignore_errors: true

    - name: Wait for shutdown to happen
      pause:
        seconds: 90

    - name: Wait for BIG-IP to actually be ready
      bigip_wait:
        user: "{{ username }}"
        password: "{{ password }}"
        server: 10.1.1.4
        validate_certs: "{{ validate_certs }}"
```

Run this playbook, from the `lab2.13` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

Waiting for the BIG-IP to be available is actually a really difficult thing to do. It gets better in later versions of BIG-IP (13.1 and beyond). For those and all the earlier releases (back to 12.0.0) you can use this module.

This module will not return until the BIG-IP is ready to take configuration. This means that it will wait for,

- mcpd
- iControl REST
- ASM
- vCMP

Notice that I mentioned several features that themselves are problematic to wait for. This module will accommodate them.

Once this module returns (and Ansible moves on to the next Task) you will be able to use any F5 Ansible module that would change the configuration.

## 3.2.14 Changing the root password

### Problem

You need to change the password of the root user

### Solution

Use the `bigip_user` module.

1. Create a `lab2.14` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example user modification playbook
  hosts: bigip
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Change root password
      bigip_user:
        username_credential: root
        password_credential: ChangedPassword1234
        password: "{{ password }}"
        server: 10.1.1.4
```

```
validate_certs: "{{ validate_certs }}"
user: "{{ username }}"
```

Run this playbook, from the `lab2.14` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

The `bigip_user` module can manage user accounts across the system.

Specifying the users and credentials that you want to modify are done by providing the `username_credential` or `password_credential` respectively.

In most cases, this module is idempotent. The way it achieves idempotency for password changes is that it will attempt to log in to the REST API as the specified `username_credential`. If this succeeds, the password is assumed to have already been changed, and it will not attempt to change it again.

There is one case where this idempotency for passwords is not supported; the `root` account.

While we can change the root account via the REST API, there is no way to subsequently log into the box as the `root` user to verify the password has already been changed. Therefore, for the root user, and the root user only, a `changed` event will be raised whenever you change its password.

Because of this, it is recommended that you put any Tasks that change the root user account into their own, infrequently used, Playbooks.

## 3.3 Module – Slightly more advanced Ansible usage

Basic administration should serve you well for quite some time.

As you become more seasoned in using the tool and understanding the way that both Ansible and your BIG-IP behave, you will want to begin to brave the world of more advanced playbook and deployment scenarios.

The recipes in this chapter look at even more modules, some of which you may use less often than others.

Throughout the course of this chapter we will also explore topics in Ansible that are a step beyond basic. These include prompts, encrypted files, and interrupting the flows of the a play through the use of custom arguments.

### 3.3.1 Prompting for user input

#### Problem

You need to prompt the user to provide a password to Ansible

#### Solution

Use the `vars_prompt` block in your Playbook.

1. Create a `lab3.1` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.

3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.14`.
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example prompting playbook
  hosts: server

  vars_prompt:
    - name: partition
      prompt: "Enter a partition name"
      default: "Common"

  tasks:
    - name: Print out your input
      debug:
        msg: "You provided the {{ partition }} partition for the 'partition'
        ↪prompt"
```

Run this playbook, from the `lab3.1` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

Prompting is a great way to get input from the user. It can function in both an interactive, and non-interactive way. We will learn later about that [1.3 Expected File Layout](#).

Prompts can also blot out the values that you provide, so they can be useful insinuations where you prompt for a password. This removal of input is done with the `private` keyword to the prompt, such as

```
vars_prompt:
  - name: "some_password"
    prompt: "Enter password"
    private: yes
```

By default, private-ness is disabled.

You may want to use this instead of storing the password credentials in the playbook.

1. *Type* the following into the `playbooks/site2.yaml` file.

```
---
- name: An example pool playbook
  hosts: bigip
  connection: local

  vars_prompt:
    - name: "username"
      prompt: "Enter BIG-IP username"
      private: yes
    - name: "password"
      prompt: "Enter BIG-IP password"
      private: yes
```

```
tasks:
  - name: Create web servers pool
    bigip_pool:
      name: web-servers
      lb_method: ratio-member
      password: "{{ password }}"
      server: 10.1.1.4
      user: "{{ username }}"
      validate_certs: no
```

Run this playbook, from the lab3.1 directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site2.yaml
```

### 3.3.2 Keeping secrets secret

#### Problem

You need to store passwords for use in Ansible

#### Solution

Use `ansible-vault`.

The `ansible-vault` command has three subcommands that are frequently used.

- create
- edit

#### Creating

Use `create` to create the initial files that will be vault encrypted. When you use the `create` subcommand, Vault will prompt you for a password. It will then open up a text editor for you to write data to it. Data of any form can be written, but text is usually the format that is used.

```
$ ansible-vault create foo.bar
New Vault password:
Confirm New Vault password:
$
```

When you save and quit the editor, the file will automatically be encrypted for you. You can look at the encrypted file by `cat`'ing it.

```
$ cat foo.bar
$ANSIBLE_VAULT;1.1;AES256
313665373835356130343064616238663161373930623638653839663732663138393062323266386330643363386534
38390a32373037313030653365613963643237643033663164376264313637
61636134633863356536386133383065376533643864356362653737396632
33373531650a39643034336463326138653439633637643033363735383665
64313134613337
$
```

## Editing

You may edit an existing Vault file by using a similar command

```
$ ansible-vault edit foo.bar
Vault password:
```

This time you will be asked for the password so that you can decrypt the file.

## Discussion

Vault is a tool that comes pre-installed with Ansible. It is a decent way to protect data that is **not publicly available**. If you want to make data publicly available, it is recommended that you use a technology like GPG.

Vault requires that a password be specified so that it can decrypt files. That password can either be specified on the CLI or in a file.

It is recommended that for automation, this information is stored in a file.

If you store the password in a file, you can provide this file with the `--vault-password-file` argument to the `ansible-vault` command. This file does not need to be static though. It can also be a script that gets the password dynamically. For instance, if you stored the password itself in a organization wide password-manager.

### 3.3.3 Local connection versus delegation

#### Problem

You need to know when to use `connection: local` and `delegate_to: localhost`

#### Solution

An explanation of the difference between these two is [here](#). It is reprinted here for your convenience.

There are three major differences between `connection: local` and `delegate_to: localhost`:

- `connection: local` applies to all hosts
- `delegate_to` applies to specific hosts
- `delegate_to` runs your task on one host, in the context of another host

#### Connection: local

First, `connection: local` applies to all hosts in the playbook. If you find yourself mixing and matching BIG-IP hosts with things like web servers, it would cause your legitimate ssh connections to fail.

This is because when you specify `connection: local`, every host is now considered to have 127.0.0.1 as their IP address.

This is likely not what you want.

For example,

1. Create a `lab3.3` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a server host to the ansible inventory and give it,
  - an `ansible_host` fact with the value `10.1.1.6`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: This is my play
  hosts: server
  connection: local

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Disable pool member for upgrading
      bigip_pool_member:
        pool: web-servers
        port: 80
        name: "{{ inventory_hostname }}"
        monitor_state: disabled
        session_state: disabled
        password: "{{ password }}"
        server: 10.1.1.4
        user: "{{ username }}"
        validate_certs: "{{ validate_certs }}"

    - name: Upgrade the webserver
      apt:
        name: apache2
        state: latest

    - name: Re-enable pool member after upgrading
      bigip_pool_member:
        pool: web-servers
        port: 80
        name: "{{ inventory_hostname }}"
        monitor_state: enabled
        session_state: enabled
        password: "{{ password }}"
        server: 10.1.1.4
        user: "{{ username }}"
        validate_certs: "{{ validate_certs }}"
```

Run this playbook, from the `lab3.3` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

This playbook will run, but it's actually not correct. The reason is because of the second task.

The second task is not what you want because it attempts to run the `apt` module on your **local** machine. Your playbook, however, intended to upgrade the **remote** webserver.

So you installed apache on the Ansible controller machine.

You can verify this with the following command

- `dpkg --list | grep apache`

For example, here is the output on my ansible controller

```
$ dpkg --list | grep apache
ii  apache2                2.4.18-2ubuntu3.5
↳  amd64                 Apache HTTP Server
ii  apache2-bin            2.4.18-2ubuntu3.5
↳  amd64                 Apache HTTP Server (modules and other binary files)
ii  apache2-data           2.4.18-2ubuntu3.5
↳  all                    Apache HTTP Server (common files)
ii  apache2-utils          2.4.18-2ubuntu3.5
↳  amd64                 Apache HTTP Server (utility programs for web servers)
```

Whoops.

You can remove apache on the Ansible controller with this command

- `apt-get remove --purge apache2*`

## Delegation

You can remedy this situation with `delegate_to`. For the most part, you will use this feature when the connection line is set to `ssh` (the default).

Delegation allows you to mix and match remote hosts. You continue to use an SSH connection for legitimate purposes, such as connecting to remove servers, but for the devices that don't support this option, you delegate their tasks.

For example, this playbook will correct your problem:

1. Change your `playbooks/site.yaml` file to reflect the changes below.

```
---
- name: This is my play
  hosts: server

  vars:
    validate_certs: no
    username: admin
    password: admin

  tasks:
    - name: Disable pool member for upgrading
      bigip_pool_member:
        pool: web-servers
        port: 80
        name: "{{ inventory_hostname }}"
        monitor_state: disabled
        session_state: disabled
        password: "{{ password }}"
        server: 10.1.1.4
        user: "{{ username }}"
        validate_certs: "{{ validate_certs }}"
        delegate_to: localhost
```

```

- name: Upgrade the webserver
  apt:
    name: apache2
    state: latest

- name: Re-enable pool member after upgrading
  bigip_pool_member:
    pool: web-servers
    port: 80
    name: "{{ inventory_hostname }}"
    monitor_state: enabled
    session_state: enabled
    password: "{{ password }}"
    server: 10.1.1.4
    user: "{{ username }}"
    validate_certs: "{{ validate_certs }}"
    delegate_to: localhost

```

The `delegate_to` parameter delegates the running of the task to some completely different machine.

However, instead of the module having access to that totally different machine's facts, it instead has the facts of the inventory item where the delegation happened. This is using the context of the host.

We also removed the `connection: local` line. This means that Ansible will try to connect over SSH to all of our hosts on the `hosts:` line.

## Discussion

Quiz time.

In the above example, *even though* the first and third tasks are running on the Ansible controller (instead of the remote webserver), what is the value of the `{{ inventory_hostname }}` variable?

1. localhost
2. server
3. something else

If you answered `server` then you are correct.

This is **context**. The task executed on localhost using `server`'s context, and therefore, its `facts`.

## 3.3.4 Starting the playbook at a specific task

### Problem

You need to start at a specific task in a playbook

### Solution

Use the `--start-at-task` argument of `ansible-playbook`

1. Create a `lab3.4` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.

3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example start-at playbook
  hosts: bigip
  connection: local

  environment:
    F5_SERVER: "{{ ansible_host }}"
    F5_USER: admin
    F5_PASSWORD: admin
    F5_SERVER_PORT: 443
    F5_VALIDATE_CERTS: no

  vars:
    send_string1: "GET /bizdev\r\n"
    monitor_name: "monitor1"

  tasks:
    - name: Create HTTP Monitor
      bigip_monitor_http:
        name: "{{ monitor_name }}"
        send: "{{ send_string1 }}"
        register: result

    - name: Assert Create HTTP Monitor
      assert:
        that:
          - result is changed
          - result.send == send_string1

    - name: Create HTTP Monitor - Idempotent check
      bigip_monitor_http:
        name: "{{ monitor_name }}"
        send: "{{ send_string1 }}"
        register: result

    - name: Assert Create HTTP Monitor - Idempotent check
      assert:
        that:
          - result is not changed
```

You can see that we have 4 Tasks in this Playbook.

You can run this Playbook once and it will do its thing. Then, assume that you want to run the playbook again, but you want to start at the *Create HTTP Monitor - Idempotent check* Task.

You can do this by specifying the Task name to the `---start-at-task` argument.

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml --start-at-task
↪ "Create HTTP Monitor - Idempotent check"
```

The Play will run, but will start at the third Task this time.

But there's an error that's raised. Why?

The answer, is because you started at the task which is intended to be the idempotent check. Run the playbook again. Does the result change?

## Discussion

This argument is extremely valuable when it comes to debugging or running specific blocks of a Playbook over.

There are certain things that you need to be aware of when using this argument though.

1. It will **not** run any prior tasks. Therefore, if you will start at (or have a future) Task that relies on some information from *before* the Task you are starting it, it will *not* be available. This will cause your Play to fail when it reaches the Task that needs this information
2. If you have multiple Tasks with the same name, the **first Task found** is the one that will be used.
3. **ALWAYS NAME YOUR TASKS!!!1!!!1!!!!1** if you do not, it makes it incredibly difficult to start-at them in the future.
4. If the Task you are starting at is in a role, prefix the role name to the task followed by spacing and a colon. For example, `--start-at-task "role1 : This is my roles task"`

Despite the constraints, this is a go-to feature that you will use all the time. Remember it.

## 3.3.5 Stepping through a playbook

### Problem

You need to step through each task because, by default, Ansible will fire off tasks as fast as possible

### Solution

Use the `--step` argument of `ansible-playbook`

1. Create a `lab3.5` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example stepped playbook
  hosts: bigip
  connection: local

  environment:
    F5_SERVER: "{{ ansible_host }}"
    F5_USER: admin
    F5_PASSWORD: admin
    F5_SERVER_PORT: 443
    F5_VALIDATE_CERTS: no

  vars:
    send_string1: "GET /hr\r\n"
```

```
monitor_name: "monitor2"

tasks:
  - name: Create HTTP Monitor
    bigip_monitor_http:
      name: "{{ monitor_name }}"
      send: "{{ send_string1 }}"
      register: result

  - name: Assert Create HTTP Monitor
    assert:
      that:
        - result is changed
        - result.send == send_string1

  - name: Create HTTP Monitor - Idempotent check
    bigip_monitor_http:
      name: "{{ monitor_name }}"
      send: "{{ send_string1 }}"
      register: result

  - name: Assert Create HTTP Monitor - Idempotent check
    assert:
      that:
        - result is not changed

  - name: Remove HTTP Monitor
    bigip_monitor_http:
      name: "{{ monitor_name }}"
      state: absent
      register: result
```

You can see that we have 5 Tasks in this Playbook.

You have this test playbook, but you are not sure if they Tasks are actually doing their work because the last Task removes the monitor. How do you check that 1 actually changed the remote device? Sure, it may report changed, but did it *really* change?

You can do this by specifying the `---step` argument to your Playbook.

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml --step
```

The Play will run, but will Ansible will prompt you to either do the Task, Skip the Task, or Continue on with all Tasks.

For example,

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml --step

PLAY [An example partition playbook]_
↪*****
Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: y

Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue:_
↪*****

TASK [Gathering Facts]_
↪*****
ok: [bigip1]
```

```
Perform task: TASK: Create HTTP Monitor (N)o/(y)es/(c)ontinue:
^C
$
```

## Discussion

Stepping is something I use frequently when I am writing a Playbook initially. Between each step, Ansible will pause indefinitely and let you do something out-of-band of the Playbook.

Often, I will do a task, then do either a series of debug work, or configuration validation. For example, if I am using a new module, did the module actually change my BIG-IP as I expected it would?

For debugging, I can pause right before a Task and make sure that,

- the device is indeed ready for my config
- any log files I am going to tail are empty so I don't need to go look through them
- Any debug-level logging is configured on any remote devices
- etc

I can then run the Task, and proceed with the other future Tasks as needed. Once I am ready to quit, I can `ctrl+c` the Playbook to stop all execution. Or, I can press `c` to tell Ansible to proceed on with the entire rest of the Playbook.

## 3.3.6 Sending arguments to your playbook

### Problem

You need to specify “vars” values automatically, such as via a command line.

### Solution

Use the `-e`, or `--extra-vars` argument of `ansible-playbook`

Remember the Playbook we had back in [Lab 3.1](#)? That Playbook prompted us for variables every time we ran it. Now we want to run the same playbook without getting those prompts.

We can supply the prompt variable names, and their values, on the command line.

1. Change into the `lab3.1` directory.

Run this playbook, from the `lab3.1` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml -e "username=admin_
↪password=admin"
```

## Discussion

This method of specifying values is not reserved for credentials.

In most cases, it **should not** be used for credentials in fact. This is because the Ansible command (including the extra arguments) will show in the running process list of your Ansible controller.

The more common situations are when you are prompting for specific configuration related to something on your network. For example, your Playbook may be flexible enough to take a given `region` or `cell`.

This would look like the following

```
$ ansible-playbook -i inventory/hosts bootstrap.yaml -e "region=ord_
↪cell=c0006"
```

The Playbook would not need to change, but you could continually provide values to variables in the Playbook to keep from writing them into the actual Playbook itself.

### 3.3.7 Creating iRules from a list, with loop

#### Problem

You need to upload a series of iRules to a BIG-IP

#### Solution

Use the `bigip_irule` module and the `loop` keyword.

1. Change into the `lab3.7` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Change the `playbooks/site.yaml` file to resemble the following.
4. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`

```
---
- name: An example looping iRule playbook
  hosts: bigip
  connection: local

  environment:
    F5_SERVER: "{{ ansible_host }}"
    F5_USER: admin
    F5_PASSWORD: admin
    F5_SERVER_PORT: 443
    F5_VALIDATE_CERTS: no

  tasks:
    - name: Create iRule in LTM
      bigip_irule:
        content: "{{ lookup('file', item.file) }}"
        module: ltm
        name: "{{ item.name }}"
      loop:
        - name: irule1
          file: ../files/irule-01.tcl
        - name: irule2
          file: ../files/irule-02.tcl
        - name: irule3
          file: ../files/irule-03.tcl
```

Run this playbook, from the `lab3.7` directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

## Discussion

iRules are managed on the remote system using the `bigip_irule` module. Since the possibility exists though that there may be many iRules you want to upload, one way of accomplishing that is to use the `loop` keyword in Ansible.

Notice that the `loop` keyword itself is not a parameter to the module because it is not vertically aligned with the parameters underneath the `bigip_irule` YAML above.

Instead, this keyword is internal to Ansible itself. It's available to nearly every module. Therefore you can loop with things like pools, virtual servers, nodes, etc.

The way to correctly read the above is, "run the `bigip_irule` module for each item in the `loop` list".

There are also variables in the above playbook that we haven't seen before; `item.name` and `item.file`. What do these mean?

When you use the `loop` construct, it will **automatically** create a variable for you called `item`. The value in this variable will change with each iteration of the loop to match the value in the loop.

The dot that separates `item` from the other words is Ansible lingo for a method of referring to subkeys.

In our `loop` list, we specified a *list of dictionaries*. A dictionary has key names, and those names can have values of any type. In our case, the key names for each item in the list are `name` and `file`.

Therefore, when we refer to the variable `item.name` we are referring to the `name` key's value of the current `item` in the list.

The above loop causes the task to run three times; one for each item in the loop.

## 3.3.8 The fallback F5 module for when there is no idempotent module

### Problem

You need to use a `tmsh` command that does not have an Ansible module equivalent

### Solution

Use the `bigip_command` module

1. Create a `lab3.8` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`
4. *Type* the following into the `playbooks/site.yaml` file.

```
---
- name: An example command playbook
  hosts: bigip
  connection: local
```

```
environment:
  F5_SERVER: "{{ ansible_host }}"
  F5_USER: admin
  F5_PASSWORD: admin
  F5_SERVER_PORT: 443
  F5_VALIDATE_CERTS: no

tasks:
  - name: Create a datagroup using tmsh
    bigip_command:
      commands: "create /ltm data-group internal applicationIdRealm type_
↵string records add { epc.foo.bar.org { data 16777264 } }"
```

Run this playbook, from the lab3.8 directory like so

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

### Discussion

This module is what we recommend for all situations where you need to do something that a current module does not support.

This module will **always** warn you when you use it for things that change configuration. These warnings will inform you to file an issue on our Github Issue tracker for a feature enhancement.

Ultimately, the goal we want to get to is to have a suite of modules that meets all the needs of customers that use Ansible. Since that is not yet possible, the `bigip_command` is there to accommodate.

This module can also be used over SSH, but password SSH is the only method known to work at this time.

### 3.3.9 Running in a virtualenv, and the associated problems

#### Problem

You need to run Ansible from a Python virtualenv environment

#### Solution

This is possible, but it requires a keen understanding of how Ansible works, as well as a change to the `host_vars` for a single host (or `group_vars` if you want to apply this to multiple hosts)

1. Change into the lab3.9 directory in the labs directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Change the `playbooks/site.yaml` file to resemble the following.
4. Add a `bigip` host to the ansible inventory and give it an `ansible_host` fact with the value `10.1.1.4`

```
---
- name: An example command playbook
  hosts: bigip
  connection: local
```

```
environment:
  F5_SERVER: "{{ ansible_host }}"
  F5_USER: admin
  F5_PASSWORD: admin
  F5_SERVER_PORT: 443
  F5_VALIDATE_CERTS: no

tasks:
  - name: Create a datagroup using tmsh
    bigip_command:
      commands: "tmsh show sys version"
```

Next, we will uninstall our `f5-sdk` package from the system. Most people consider this to be an OK thing to do because, after all, they will be running Ansible from a virtualenv.

```
pip uninstall --yes f5-sdk bigsuds
```

There is a virtualenv pre-installed on your Ansible Controller. You can activate it with the following command

```
$ source ~/.virtualenvs/lab3.9/bin/activate
```

You will know that you are in the virtualenv, because your prompt will change. It should look similar to this, where the word *ansible* prefixes the CLI prompt.

```
(lab3.9) $
```

You can verify that the necessary *pip* libraries are installed with the following command.

```
$ pip freeze
```

You should see in this list, an entry for `f5-sdk`.

Let's now run the Ansible Playbook.

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

If your playbook fails, that is to be expected.

Now, change the `inventory/hosts` file and add the following fact to the *bigip* line.

```
ansible_python_interpreter=~/.virtualenvs/lab3.9/bin/python
```

Re-run the `ansible-playbook` command from above.

If your playbook succeeds, that is expected. Proceed to the Discussion for a deeper answer as to what is happening.

Be sure to re-install the F5 Ansible dependencies that you removed as we will use them in future labs

```
$ pip install f5-sdk bigsuds
```

## Discussion

Why does it fail the first time? The answer is because Ansible is **not** running your module in the virtualenv. It's running it on the system's Python.

That doesn't make sense though, it should be running in the virtualenv. Wrong.

## A brief segue is necessary

Ansible's default behavior is that it **always** SSH's to the remote host. Always. Even when `connection: local` is set, it is running...in a sense...on the "remote" host; only this time, localhost is considered the remote host.

Setting `connection: local` only eliminates the SSH protocol, it does **not**, however, eliminate the fact that Ansible is going to always run your module using `/usr/bin/python`.

By default, modules point at `/usr/bin/python`. Period.

So Ansible itself runs just fine in a virtualenv. The problem is that when it communicates with the "remote" host, the **module** is going to run with `/usr/bin/python`. That means that the F5 dependency libraries are also going to be looked up according to `/usr/bin/python`. If you installed your dependencies in a virtualenv, that virtualenv's python is not `/usr/bin/python`.

This is why you **must** set the `ansible_python_interpreter` for any hosts, or groups of hosts, where the python interpreter differs. We did this in our solution for a single host when we changed the `inventory/hosts` file. We could have also created a file at `inventory/group_vars/all.yaml` and those facts would apply to **all** hosts in your playbook.

## 3.3.10 Creating roles

### Problem

You need to reuse the work you have just done in other playbooks without repeatedly writing tasks.

### Solution

Use Roles.

*You may combine any set of Tasks that we have used previously in this role.*

A role is an abstraction in which a *directory named after the role* is created in the `roles` directory. In the first module, we learned about the *expected file layout*. Part of this layout is a `roles` directory. It is that directory in which you put the role directory.

1. Change into the `lab3.10` directory in the `labs` directory.
2. Setup the filesystem layout to mirror the one *described in lab 1.3*.
3. Make a role named `app1`

```
$ mkdir -p roles/app1
```

A role directory has the same directory structure that we created in the first module. There are exceptions though. They are

- the `playbooks` directory has been replaced with `tasks`
- there is no `roles` directory in a role
- there is no `inventory` directory in a role
- When a role is included, Ansible only calls a file named `main.yaml` in the `tasks` directory
- Several new directories, such as `vars` and `defaults` are available to add

With these constraints in place, we create the following directory structure in the `app1` directory.

```
.
?- defaults
|   ?- main.yaml
?- files
?- tasks
|   ?- main.yaml
?- templates
```

With this structure in place, we can cherry pick Tasks from any of our other Playbook we have written and add them to the `tasks/main.yaml` file.

Our `app1` role will do the following

1. consume a `tenant` variable, defaulting to `Common`
2. consume a `bigip_port` variable, defaulting to `443`
3. consume a `validate_certs` variable, defaulting to `no`
4. consume a `bigip_username` variable
5. consume a `bigip_password` variable
6. consume a `bigip_server` variable
7. fail if any of the variables above are not defined
8. create a partition using the name of the `tenant` variable
9. create a pool named `app1-pool` on the `tenant` partition. Use the round-robin load balancing method
10. create a single iRule using one of the same iRule files we used from [the earlier lab](#). Name it `irule1`
11. create a virtual server named `app1-vs` on the `tenant` partition. Assign it the iRule and pool you created. It should have a destination of `10.1.10.240` and a port of `80`. Finally, set `snat` to `Automap`
12. create a node for each host in the playbook using the current `ansible_host` IP address
13. add the node to the `app1-pool` pool
14. Save the running configuration

To accomplish the above, let's do the following

### Construct a playbook to use your role

Create the file `playbooks/site.yaml` in the `lab3.10` directory; **not** the role directory. Put the following in it.

```
---

- name: Use app1 role
  hosts: app1
  connection: local

  vars_prompt:
    - name: bigip_username
      prompt: "Enter the BIG-IP username"
      private: no
    - name: bigip_password
      prompt: "Enter the BIG-IP password"
```

```
private: yes
- name: bigip_server
  prompt: "Enter the BIG-IP server address"
  private: no

roles:
- appl
```

This is the playbook we will use.

### Create default variables

In the `appl` role directory, edit the `defaults/main.yaml` file, add the following

```
---

tenant: Common
bigip_port: 443
validate_certs: no
```

This accomplishes bullets #1 to #3

### Create a setup task list

Create the file `tasks/setup.yaml`

In this file, put the following

```
---

- name: Check to see if bigip username credential missing
  fail:
    msg: "You must provide a 'bigip_username' variable"
    when: bigip_username is not defined

- name: Check to see if bigip passwd credential missing
  fail:
    msg: "You must provide a 'bigip_password' variable"
    when: bigip_password is not defined

- name: Check to see if bigip server credential missing
  fail:
    msg: "You must provide a 'bigip_server' variable"
    when: bigip_server is not defined
```

This accomplishes bullets #4 to #7

### Create a main task list

Edit the `tasks/main.yaml` file to include the following

```
---

- import_tasks: setup.yaml
```

```

- name: Create tenant partition
bigip_partition:
  name: "{{ tenant }}"
  user: "{{ bigip_username }}"
  password: "{{ bigip_password }}"
  validate_certs: "{{ validate_certs }}"
  server: "{{ bigip_server }}"
  server_port: "{{ bigip_port }}"
  delegate_to: localhost

- name: Create pool
bigip_pool:
  name: "{{ tenant }}-pool1"
  lb_method: round-robin
  partition: "{{ tenant }}"
  user: "{{ bigip_username }}"
  password: "{{ bigip_password }}"
  validate_certs: "{{ validate_certs }}"
  server: "{{ bigip_server }}"
  server_port: "{{ bigip_port }}"
  delegate_to: localhost

- name: Create iRule
bigip_irule:
  content: "{{ lookup('file', 'irule-01.tcl') }}"
  module: ltm
  name: irule1
  partition: "{{ tenant }}"
  user: "{{ bigip_username }}"
  password: "{{ bigip_password }}"
  validate_certs: "{{ validate_certs }}"
  server: "{{ bigip_server }}"
  server_port: "{{ bigip_port }}"
  delegate_to: localhost

- name: Create virtual server
bigip_virtual_server:
  name: appl-vs
  destination: "{{ vs_destination }}"
  port: 80
  irules:
    - irule1
  profiles:
    - http
  snat: Automap
  partition: "{{ tenant }}"
  user: "{{ bigip_username }}"
  password: "{{ bigip_password }}"
  validate_certs: "{{ validate_certs }}"
  server: "{{ bigip_server }}"
  server_port: "{{ bigip_port }}"
  delegate_to: localhost

- name: Create node for physical machine
bigip_node:
  address: "{{ node_destination }}"
  name: "{{ inventory_hostname }}"

```

```
    user: "{{ bigip_username }}"
    password: "{{ bigip_password }}"
    validate_certs: "{{ validate_certs }}"
    server: "{{ bigip_server }}"
    server_port: "{{ bigip_port }}"
    delegate_to: localhost

- name: Add node to pool
  bigip_pool_member:
    pool: "{{ tenant }}-pool1"
    partition: "{{ tenant }}"
    host: "{{ node_destination }}"
    port: 80
    user: "{{ bigip_username }}"
    password: "{{ bigip_password }}"
    validate_certs: "{{ validate_certs }}"
    server: "{{ bigip_server }}"
    server_port: "{{ bigip_port }}"
    delegate_to: localhost

- name: Save running config
  bigip_config:
    save: yes
    user: "{{ bigip_username }}"
    password: "{{ bigip_password }}"
    validate_certs: "{{ validate_certs }}"
    server: "{{ bigip_server }}"
    server_port: "{{ bigip_port }}"
    delegate_to: localhost
```

This accomplishes bullets #8 to #14

**Move files to the appropriate directories**

In the task list above, we use an iRule file. To make use of it in this role, we need to put it in the files directory because we used the file lookup.

From the lab3.10 directory, issue the following command

```
cp files/irule-01.tcl roles/app1/files/
```

**Run the playbook**

With the above in place, you can run the playbook as you normally would

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

Your play, and role, should execute as expected and configure the device.

**Discussion**

As you can see from the solution above, a role is a way to encapsulate a body of work. This role could have been zipped up and given to anyone else and they could have extracted it and run it the same way that you did.

Roles can include their own files, templates, and even custom modules. They should be your go-to solution for all your work that is beyond a single playbook.

With our solution in place, our directory structure now looks like this

```
.
?- defaults
|   ?- main.yaml
?- files
|   ?- irule-01.tcl
?- tasks
|   ?- main.yaml
|   ?- setup.yaml
?- templates
```

Earlier I said that Ansible will **only** call the `tasks/main.yaml` file. That's perfectly ok though because we can include as many other files as we need.

We did just take with the `import_tasks` action in the `tasks/main.yaml` file. This action will cause Ansible to read in this file and replace the import line with the content of the file.

The `defaults` directory we made use of stores default variables. These variables may be overridden via the CLI as we learned [in an earlier lab](#).

Notice also how when we used the file lookup, we didn't need to refer to the full path to the file. This is because, in roles, if you used the file lookup, Ansible assumes the file being looked up is in the `files` directory of the role.

The `template` lookup works much the same way. If you use the following in a role

```
lookup('template', 'file.txt')
```

Ansible will implicitly look in the `templates` directory of your role.

## 3.4 Module – Debugging Ansible problems

An advanced, but equally necessary topic is debugging. It is inevitable that things will go wrong.

The problem may be in the F5 modules, or even Ansible itself. When a problem arises, it is less important that you know **what** the problem is, than know **how to diagnose** what the problem is.

These recipes will expose you to diagnosing problems with modules and how you can peel back the Ansible layer to gain better insight into what went wrong.

### 3.4.1 Enable verbose debugging

#### Problem

You want to get more context about what is happening when a Playbook runs, because right now you have none

#### Solution

Provide the `-vvvv` argument to the `ansible-playbook` command.

Enabling verbose output can be done as follows,

## 1. Change into the lab4.1 directory in the labs directory.

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml -vvvv
```

Running this will output more output than you would normally get. This playbook includes an artificial module with an error message that would not normally be displayed if you had not included the verbose output.

This is normal output

```
TASK [Raises an error] *****
An exception occurred during task execution. To see the full traceback, use
-vvv. The error was: Exception: An error occurred
fatal: [localhost]: FAILED! => {"changed": false, "module_stderr": "Traceback
(most recent call last):
  File \"/Users/trupp/.ansible/tmp/ansible-tmp-1512
284216.7-97617236630854/foo41.py\"", line 24, in <module>
    al()
  File
\="/Users/trupp/.ansible/tmp/ansible-tmp-1512284216.7-97617236630854/foo41.py\"
↪",
line 13, in al
    bl()
  File \"/Users/trupp/.ansible/tmp/ansible-tmp-
↪15122
84216.7-97617236630854/foo41.py\"", line 16, in bl
    cl()
  File \"/Users/
trupp/.ansible/tmp/ansible-tmp-1512284216.7-97617236630854/foo41.py\"", line
↪19,
in cl
    dl()
  File \"/Users/trupp/.ansible/tmp/ansible-tmp-1512284216.7-
↪9
7617236630854/foo41.py\"", line 22, in dl
    raise Exception(\"An error
↪occur
red\")
Exception: An error occurred\n", "module_stdout": "", "msg": "MODULE
FAILURE", "rc": 0}
```

This is verbose output

```
TASK [Raises an error] *****
task path: /Users/trupp/src/f5-gsts-labs-ansible-cookbook/docs/labs/
↪playbooks/lab4.1.yaml:8
Using module file /Users/trupp/src/f5-gsts-labs-ansible-cookbook/docs/labs/
↪library/foo41.py
<localhost> ESTABLISH LOCAL CONNECTION FOR USER: trupp
<localhost> EXEC /bin/sh -c 'echo ~ && sleep 0'
<localhost> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo /Users/trupp/
.ansible-tmp-1512284240.61-66631414390058 ` " && echo ansible-
tmp-1512284240.61-66631414390058="` echo /Users/trupp/.ansible/tmp/ansibl
e-tmp-1512284240.61-66631414390058 ` " ) && sleep 0'
<localhost> PUT /var/folders/jc/9d1188j962931rhqrlm4173w5j5m45/T/tmpOT27vx TO
/Users/trupp/.ansible/tmp/ansible-tmp-1512284240.61-66631414390058/foo41.py
<localhost> PUT /var/folders/jc/9d1188j962931rhqrlm4173w5j5m45/T/tmpZwW0ZP TO
/Users/trupp/.ansible/tmp/ansible-tmp-1512284240.61-66631414390058/args
<localhost> EXEC /bin/sh -c 'chmod u+x /Users/trupp/.ansible/tmp/ansible-tmp-
1512284240.61-66631414390058/ /Users/trupp/.ansible/tmp/ansible-tmp-1512284
240.61-66631414390058/foo41.py /Users/trupp/.ansible/tmp/ansible-tmp-1512
284240.61-66631414390058/args && sleep 0'
<localhost> EXEC /bin/sh -c '/usr/bin/python /Users/trupp/.ansible/tmp/ansi
ble-tmp-1512284240.61-66631414390058/foo41.py /Users/trupp/.ansible/tmp/a
nsible-tmp-1512284240.61-66631414390058/args; rm -rf "/Users/trupp/.ansib
le/tmp/ansible-tmp-1512284240.61-66631414390058/" > /dev/null 2>&1 && sle
ep 0'
The full traceback is:
Traceback (most recent call last):
  File "/Users/trupp/.ansible/tmp/ansible-tmp-1512284240.61-66631414390058/
↪foo41.py", line 24, in <module>
```

```

    a1()
    File "/Users/trupp/.ansible/tmp/ansible-tmp-1512284240.61-66631414390058/
↳foo41.py", line 13, in a1
        b1()
    File "/Users/trupp/.ansible/tmp/ansible-tmp-1512284240.61-66631414390058/
↳foo41.py", line 16, in b1
        c1()
    File "/Users/trupp/.ansible/tmp/ansible-tmp-1512284240.61-66631414390058/
↳foo41.py", line 19, in c1
        d1()
    File "/Users/trupp/.ansible/tmp/ansible-tmp-1512284240.61-66631414390058/
↳foo41.py", line 22, in d1
        raise Exception("An error occurred")
Exception: An error occurred

fatal: [localhost]: FAILED! => {
    "changed": false,
    "module_stderr": "Traceback (most recent call last):\n File \"/Users/
↳trupp/
↳.ansible/tmp/ansible-tmp-1512284240.61-66631414390058/foo41.py\", line
↳24,
↳in <module>\n    a1()\n File \"/Users/trupp/.ansible/tmp/ansible-tmp-
↳1512
↳284240.61-66631414390058/foo41.py\", line 13, in a1\n    b1()\n File \
↳"/U
↳sers/trupp/.ansible/tmp/ansible-tmp-1512284240.61-66631414390058/foo41.
↳py\
↳", line 16, in b1\n    c1()\n File \"/Users/trupp/.ansible/tmp/
↳ansible-tm
↳p-1512284240.61-66631414390058/foo41.py\", line 19, in c1\n    d1()\n
↳Fil
↳e \"/Users/trupp/.ansible/tmp/ansible-tmp-1512284240.61-66631414390058/
↳foo4
↳1.py\", line 22, in d1\n    raise Exception(\"An error occurred\
↳")\nExcepti
↳on: An error occurred\n",
    "module_stdout": "",
    "msg": "MODULE FAILURE",
    "rc": 0
}

```

## Discussion

I don't take my chances when running playbooks. I always use verbose logging.

You will find over time, that if you do not do this, that you will miss out on some of the more critical information that may be required to track down a problem.

The verbose information that is shown is typically the first step in debugging a problem, and the F5 Ansible developers will want it from you when you report a problem.

Verbose output includes several key pieces of information that will be used to debug problems even further. These include

- The connection information
- Delegation information

- Remote playbook execution files
- Structured failure output

We will discuss the third bullet in more detail in a lab in the next lab.

### 3.4.2 Save and view remote module execution code

#### Problem

You need to get the actual contents of a module that are run on the remote machine

#### Solution

The solution to this problem is a series of steps that can be near impossible to guess at. Let's follow these steps to show you.

First, set the `ANSIBLE_KEEP_REMOTE_FILES` variable to 1 when you run a playbook. Additionally, run the playbook with `-vvv`. Using the playbook in **lab4.2/playbooks/site.yaml** run the following command,

```
$ ANSIBLE_KEEP_REMOTE_FILES=1 ansible-playbook -i inventory/hosts playbooks/
→site.yaml -vvv
```

After the playbook has finished execution, note the location of the module file that was copied to the remote machine.

The module file is buried in the verbose output that the playbook generates. Refer to the image below for an example.

```
<10.1.1.6> PUT /tmp/tmp_QQdcy TO /root/.ansible/tmp/ansible-tmp-1512367718.13-21522411002
<10.1.1.6> SSH: EXEC sftp -b - -C -o ControlMaster=auto -o ControlPersist=60s -o KbdInter
publickey -o PasswordAuthentication=no -o ConnectTimeout=10 -o ControlPath=/root/.ansible
<10.1.1.6> (0, 'sftp> put /tmp/tmp_QQdcy /root/.ansible/tmp/ansible-tmp-1512367718.13-215
<10.1.1.6> ESTABLISH SSH CONNECTION FOR USER: None
<10.1.1.6> SSH: EXEC ssh -C -o ControlMaster=auto -o ControlPersist=60s -o KbdInteractive
key -o PasswordAuthentication=no -o ConnectTimeout=10 -o ControlPath=/root/.ansible/cp/06
15224110025969/ /root/.ansible/tmp/ansible-tmp-1512367718.13-215224110025969/apt.py && sl
<10.1.1.6> (0, '', '')
<10.1.1.6> ESTABLISH SSH CONNECTION FOR USER: None
<10.1.1.6> SSH: EXEC ssh -C -o ControlMaster=auto -o ControlPersist=60s -o KbdInteractive
key -o PasswordAuthentication=no -o ConnectTimeout=10 -o ControlPath=/root/.ansible/cp/06
367718.13-215224110025969/apt.py && sleep 0''''''
<10.1.1.6> (0, '\r\n{"invocation": {"module_args": {"force_apt_get": false, "autoclean":
state": "present", "autoremove": false, "purge": false, "update_cache": true, "name": "re
install_recommends": null, "upgrade": null, "force": false, "allow_unauthenticated": fals
d connection to 10.1.1.6 closed.\r\n')
ok: [server] => {
  "cache_update_time": 1512367718,
  "cache_updated": true,
  "changed": false,
  "invocation": {
    "module_args": {
      "allow_unauthenticated": false,
      "autoclean": false,
      "autoremove": false,
```

With this file found, we can ssh to the remote host in which we were running this playbook on; `server`

```
$ ssh 10.1.1.6
```

and `ls` the file to make sure it exists

```
$ ls -l /root/.ansible/tmp/ansible-tmp-1512367718.13-215224110025969/apt.py
-rwx----- 1 root root 102974 Dec  4 06:08 /root/.ansible/tmp/ansible-tmp-
↪1512367718.13-215224110025969/apt.py
$
```

This file is a copy of the module and the libraries that it includes from Ansible. It can be extracted with the `explode` argument

```
$ /root/.ansible/tmp/ansible-tmp-1512367718.13-215224110025969/apt.py explode
Module expanded into:
/root/.ansible/tmp/ansible-tmp-1512367718.13-215224110025969/debug_dir
$
```

It provides you with the directory where the content of the module was extracted to.

```
$ ls -l
total 48
drwxr-xr-x 3 root root  4096 Dec  4 06:13 ansible
-rw-r--r-- 1 root root 38495 Dec  4 06:13 ansible_module_apt.py
-rw-r--r-- 1 root root   441 Dec  4 06:13 args
$
```

The file named `ansible_module_apt.py` is the copy of module used in your task. You can edit it directly and re-run the changed module and associated files by using the `execute` argument to the same script you provided the `explode` argument to.

```
$ /root/.ansible/tmp/ansible-tmp-1512367718.13-215224110025969/apt.py execute
```

The module will be run as if it were being run directly from the Ansible controller.

## Discussion

The method you've just learned is used extensively in the beginning stages of how to debug modules. Even to this day I use it for extreme cases where I am unable to diagnose a problem and need to execute the exact module code on a remote machine.

This method requires no remote debuggers (like may be used in typical module development or debugging) and it's a rather straight-forward method once you experience the usage pattern.

- `ANSIBLE_KEEP_REMOTE_FILES`
- `/path/to/module.py explode`
- change directory and edit
- `/path/to/module.py execute`

The reason that we need to do the `explode` part in particular is because Ansible compresses the files that are part of the module, before it sends it to the remote host. This sacrifices some CPU time on the Ansible controller for what can often be a longer time transporting data over the network.

You can actually look at the compressed form if you `less` the file,

```
$ less /root/.ansible/tmp/ansible-tmp-1512367718.13-215224110025969/apt.py
```

It will produce the self-extracting script; a large portion of which will be the compressed module data

```
import shutil
import zipfile
import tempfile
import subprocess

if sys.version_info < (3,):
    bytes = str
    PY3 = False
else:
    unicode = str
    PY3 = True
try:
    # Python-2.6+
    from io import BytesIO as IOStream
except ImportError:
    # Python < 2.6
    from StringIO import StringIO as IOStream

rys9VKMh0Ly3JzFHIZC3ILypRSK0oSc1LiS9ILMngigdT8fG2SIIaMEGd+Pi8xNzU+HhNoLqy1KLizPw8oFILLIz0TPSM9
AyWgaGJpSUZ+EUjQMa84MyknVUFbMy9ZT4kLAFBLAwQUAAACAATMYRLncXxazcAAABIAAAIAAAAGFuc2libGUvbW9kd
WxLX3V0aWxzL19faW5pdF9fLnB5SvKz1UoyE4vLcnMUcjMLcgvKlFIrShJzUuJL0gsyeCKB1Px8bZIGHOWQZ34+LzE3N
T4eE0uAFBLAwQUAAACAATMYRLejKhxHAoAABflgAAFQAAAGFuc2libGVfbW9kdWxLX2FwdC5wec19+3fjtpXw7/orEPn
MITWRZHsmyaY+Ubu0x5P67MTjtT3Jdj1ehpIgm7VEKnzY4+3X//27DwAESFCyJ2m7aj0WS0Di4uLiPvHY+Wk3KvLdaZLu
rh/L2yzt7YjRy5GYZfMkvTkQVbkYfYtPevAinA3Eq739V0Pxdpk95FLVSvHu3RG8+TlPyLkmYvoofozL8LY+iJ+T5TKJV
4X4bsVP/n2hK41n2eqPU0v7uJBzkaXisVqJVTavllI81JAuZHkrfkrm8VJ8V9zd05e4FAs5z/J4nWd/lbNyn0U3AAQA/X
D6QfwgU5lDqbNquxm4l0yk2khxf3r8d6XIiykFEfvz/5ycvqDyHJxW5br4mB39+HhYXyTVghpd8k1it2b9XIEtcb1p3L
Q6y3ybCWiaFGVVS6jSCSrdZaXIp4W2RL6E/HvoZgn90mRZ0lQrPMkLaFC0ivhdy+KvRkMZ8u4KKD6RJSPa9nr9Q5PL06+
```

Near the bottom of the self-extractor is also a blurb about how to use the code should you get hung up. Here is an excerpt

```

def debug(command, zipped_mod, json_params):
    # The code here normally doesn't run. It's only used for debugging on the
    # remote machine.
    #
    # The subcommands in this function make it easier to debug ansiblez
    # modules. Here's the basic steps:
    #
    # Run ansible with the environment variable: ANSIBLE_KEEP_REMOTE_FILES=1 and -vvv
    # to save the module file remotely::
    # $ ANSIBLE_KEEP_REMOTE_FILES=1 ansible host1 -m ping -a 'data=october' -vvv
    #
    # Part of the verbose output will tell you where on the remote machine the
    # module was written to::
    # [...]
    # <host1> SSH: EXEC ssh -C -q -o ControlMaster=auto -o ControlPersist=60s -o KbdInterac
    # tiveAuthentication=no -o
    # PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey -o Password
    # Authentication=no -o ConnectTimeout=10 -o
    # ControlPath=/home/badger/.ansible/cp/ansible-ssh-%h-%p-%r -tt rhel7 '/bin/sh -c ''''
    LANG=en_US.UTF-8 LC_ALL=en_US.UTF-8
    # LC_MESSAGES=en_US.UTF-8 /usr/bin/python /home/badger/.ansible/tmp/ansible-tmp-1461173
    # 013.93-9076457629738/ping''''''
    # [...]
    #
    # Login to the remote machine and run the module file via from the previous
    # step with the explode subcommand to extract the module payload into
    # source files::
    # $ ssh host1
    # $ /usr/bin/python /home/badger/.ansible/tmp/ansible-tmp-1461173013.93-9076457629738/p
    # ing explode
    # Module expanded into:
    # /home/badger/.ansible/tmp/ansible-tmp-1461173408.08-279692652635227/ansible
    #
    # You can now edit the source files to instrument the code or experiment with
    # different parameter values. When you're ready to run the code you've modified
    # (instead of the code from the actual zipped module), use the execute subcommand like th
    # is::
    # $ /usr/bin/python /home/badger/.ansible/tmp/ansible-tmp-1461173013.93-9076457629738/p
    # ing execute

    # Okay to use __file__ here because we're running from a kept file
    basedir = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'debug_dir')
    args_path = os.path.join(basedir, 'args')
    script_path = os.path.join(basedir, 'ansible_module_apt.py')

    if command == 'explode':
        # transform the ZIPDATA into an exploded directory of code and then
        # print the path to the code. This is an easy way for people to look
        # at the code on the remote machine for debugging it in that
        # environment

```

There are three commands, but only two that are frequently used, they are

- extract
- execute
- excommunicate (almost never used)

One last thing. It is not recommended that you run **all** your playbook with `ANSIBLE_KEEP_REMOTE_FILES` **all** the time. This is because keeping these remote files causes a number of temporary files to build up on the remote host.

This can lead to disk space errors, filesystem errors, and even Ansible errors if too many temp files exist (name collisions can happen for instance).

So it is best that you reserve the usage of this method for the times when you need to do serious squirrel levels of debugging in either your own code, or the code of others.

### 3.4.3 Filing bugs

#### Problem

You need to report a bug in an F5 Ansible module

#### Solution

Create an issue on our [Github issue tracker here](#)

When creating issues, you will be asked to fill out a number of fields.

- Issue type
- Component name
- Ansible version
- BIGIP version
- Library versions
- Configuration
- OS/Environment
- Summary
- Steps to reproduce
- Expected results
- Actual results

It is **critical** that you provide as much information as possible if you hope to get your bug fixed.

#### Discussion

All customers, F5'ers, partners, everyone... needs to file issues on Github. This is a publicly operated project and we use the publicly available Github issue tracker to manage it.

**It is not acceptable** to directly email Tim with a bug you found. He will direct you to the Issue tracker.

Why is it so important that you log bugs in the issue tracker? Because of the way that we manage fixes in our source tree. Every issue has a dedicated test file that is created for it. Therefore, if you do not create an issue, we cannot create a file in our source tree to test a fix.

I repeat, because I cannot stress this enough,

**File bug reports on the F5 Ansible Github Issue Tracker**

### 3.4.4 Getting assistance with a problem

#### Problem

You need to ask for help in using an F5 Ansible module or debugging a problem in a module

## Solution

Use either of the following channels

- [f5CloudSolutions Slack team](#) in the `#ansible` channel
- Join, and send email to, `*sme_ansible`

Both channels are monitored with roughly equal consistency. However, if you do not receive a timely response on one channel, consider asking on another channel

## Discussion

You have the option of getting real-time interactive help (via Slack), or, semi-realtime help (via email).

The maintainers of the F5 Ansible modules are located in Seattle at the time of this writing. Therefore, it is not always realistic to expect to get a response immediately in our Slack channel.

On both communication channels, there is a growing body of tribal knowledge being accumulated among the channel participants. We are beginning to see several non-F5 participants helping answer questions about our modules.

We fully encourage this to continue.

## 3.4.5 Enable debug output

### Problem

You want to see more complete debugging when running playbooks

### Solution

Set the environment variable `ANSIBLE_DEBUG` to `1` when you run the `ansible-playbook` command.

1. Change into the `lab4.5` directory in the `labs` directory.

```
$ ANSIBLE_DEBUG=1 ansible-playbook -i inventory/hosts playbooks/site.yaml
```

Running this will output *a lot* more output than even the verbose output gives you. None of the debug output is what you would normally get. This playbook is a contrived example, but illustrates debug's output.

This is normal output

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml

PLAY [Labb 4.5] *****

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

TASK [Run a task] *****
ok: [localhost]

TASK [Run a second task] *****
skipping: [localhost]

TASK [Run a third task] *****
```

```

ok: [localhost] => {
  "fact1": "foo"
}

PLAY RECAP *****
localhost                : ok=3    changed=0    unreachable=0    failed=0

```

This is verbose output (truncated for readability)

```

$ ANSIBLE_DEBUG=1 ansible-playbook -i inventory/hosts playbooks/site.yaml
88233 1512328569.77688: starting run
88233 1512328569.87126: Added group all to inventory
88233 1512328569.87134: Added group ungrouped to inventory
88233 1512328569.87139: Group all now contains ungrouped
88233 1512328569.87326: Loading InventoryModule 'host_list' from /Users/tru.
↪...
88233 1512328569.87484: assigned :doc
88233 1512328569.87490: assigned :plainexamples
88233 1512328569.87513: Loading InventoryModule 'script' from /Users/trupp/.
↪...
88233 1512328569.89141: assigned :doc
88233 1512328569.89160: Loaded config def from plugin (inventory/script)
88233 1512328569.89193: Loading InventoryModule 'yaml' from /Users/trupp/sr.
↪...
88233 1512328569.89762: assigned :doc
88233 1512328569.89769: assigned :plainexamples
88233 1512328569.89780: Loaded config def from plugin (inventory/yaml)
88233 1512328569.89823: Loading InventoryModule 'ini' from /Users/trupp/src.
↪...
88233 1512328569.90388: assigned :doc
88233 1512328569.90394: assigned :plainexamples
88233 1512328569.90425: Loading InventoryModule 'auto' from /Users/trupp/sr.
↪...
88233 1512328569.90644: assigned :doc
88233 1512328569.90649: assigned :plainexamples
88233 1512328569.90696: Examining possible inventory source: /Users/trupp/s.
↪...
88233 1512328569.90705: Attempting to use plugin host_list (/Users/trupp/sr.
↪...
88233 1512328569.90713: /Users/trupp/src/f5-gsts-labs-ansible-cookbook/docs.
↪...
88233 1512328569.90717: Attempting to use plugin script (/Users/trupp/src/e.
↪...
88233 1512328569.90729: /Users/trupp/src/f5-gsts-labs-ansible-cookbook/docs.
↪...
88233 1512328569.90734: Attempting to use plugin yaml (/Users/trupp/src/env.
↪...
88233 1512328569.90804: Loading data from /Users/trupp/src/f5-gsts-labs-ans.
↪...
88233 1512328569.90854: /Users/trupp/src/f5-gsts-labs-ansible-cookbook/docs.
↪...
88233 1512328569.90860: Attempting to use plugin ini (/Users/trupp/src/envs.
↪...
88233 1512328569.90993: set inventory_file for localhost
88233 1512328569.91004: set inventory_dir for localhost
88233 1512328569.91009: Added host localhost to inventory
88233 1512328569.91015: Added host localhost to group ungrouped
88233 1512328569.91020: Reconcile groups and hosts in inventory.

```

```

88233 1512328569.91025: Group all now contains localhost
88233 1512328569.91166: Loading CacheModule 'memory' from /Users/trupp/src/.
↪...
88233 1512328569.93963: Loading data from /Users/trupp/src/f5-gsts-labs-ans.
↪...
88233 1512328570.14469: Loading CallbackModule 'default' from /Users/trupp/.
↪...
88233 1512328570.14924: assigned :doc
88233 1512328570.14989: Loading CallbackModule 'actionable' from /Users/tru.
↪...
88233 1512328570.15173: assigned :doc
88233 1512328570.15194: Loading CallbackModule 'context_demo' from /Users/t.
↪...
88233 1512328570.15406: assigned :doc
88233 1512328570.15427: Loading CallbackModule 'debug' from /Users/trupp/sr.
↪...
88233 1512328570.15599: assigned :doc
88233 1512328570.15609: Loading CallbackModule 'default' from /Users/trupp/.
↪...
88233 1512328570.15980: assigned :doc
88233 1512328570.16029: Loading CallbackModule 'dense' from /Users/trupp/sr.
↪...
88233 1512328570.16451: assigned :doc
88233 1512328570.20526: Loading CallbackModule 'foreman' from /Users/trupp/.
↪...
88233 1512328570.21163: assigned :doc
88233 1512328570.21185: Loaded config def from plugin (callback/foreman)
88233 1512328570.21223: Loading CallbackModule 'full_skip' from /Users/trup.
↪...
88233 1512328570.21458: assigned :doc
88233 1512328570.22018: Loading CallbackModule 'hipchat' from /Users/trupp/.
↪...
88233 1512328570.22699: assigned :doc
88233 1512328570.22725: Loaded config def from plugin (callback/hipchat)
88233 1512328570.22780: Loading CallbackModule 'jabber' from /Users/trupp/s.
↪...
88233 1512328570.23387: assigned :doc
88233 1512328570.23440: Loaded config def from plugin (callback/jabber)
88233 1512328570.23524: Loading CallbackModule 'json' from /Users/trupp/src.
↪...
88233 1512328570.23921: assigned :doc
88233 1512328570.24611: Loading CallbackModule 'junit' from /Users/trupp/sr.
↪...
88233 1512328570.25312: assigned :doc
88233 1512328570.25339: Loaded config def from plugin (callback/junit)
88233 1512328570.25366: Loading CallbackModule 'log_plays' from /Users/trup.
↪...
88233 1512328570.25624: assigned :doc
88233 1512328570.25681: Loading CallbackModule 'logentries' from /Users/tru.
↪...
88233 1512328570.27404: assigned :doc
88233 1512328570.27414: assigned :plainexamples
88233 1512328570.27440: Loaded config def from plugin (callback/logentries)
88233 1512328570.27493: Loading CallbackModule 'logstash' from /Users/trupp.
↪...
88233 1512328570.28007: assigned :doc
88233 1512328570.28025: Loaded config def from plugin (callback/logstash)
88233 1512328570.28143: Loading CallbackModule 'mail' from /Users/trupp/src.
↪...

```

```

88233 1512328570.28534: assigned :doc
88233 1512328570.28553: Loaded config def from plugin (callback/mail)
88233 1512328570.28576: Loading CallbackModule 'minimal' from /Users/trupp/.
↳...
88233 1512328570.28733: assigned :doc
88233 1512328570.28762: Loading CallbackModule 'null' from /Users/trupp/src.
↳...
88233 1512328570.28914: assigned :doc
88233 1512328570.28943: Loading CallbackModule 'oneline' from /Users/trupp/.
↳...
88233 1512328570.29117: assigned :doc
88233 1512328570.29147: Loading CallbackModule 'osx_say' from /Users/trupp/.
↳...
88233 1512328570.29348: assigned :doc
88233 1512328570.29375: Loading CallbackModule 'profile_roles' from /Users/.
↳...
88233 1512328570.29630: assigned :doc
88233 1512328570.29702: Loading CallbackModule 'profile_tasks' from /Users/.
↳...
88233 1512328570.30465: assigned :doc
88233 1512328570.30476: assigned :plainexamples
88233 1512328570.30505: Loaded config def from plugin (callback/profile_
↳tasks)
88233 1512328570.30542: Loading CallbackModule 'selective' from /Users/trup.
↳...
88233 1512328570.31090: assigned :doc
88233 1512328570.31097: assigned :plainexamples
88233 1512328570.31118: Loaded config def from plugin (callback/selective)
88233 1512328570.31139: Loading CallbackModule 'skippy' from /Users/trupp/s.
↳...
88233 1512328570.31305: assigned :doc
88233 1512328570.31328: Loading CallbackModule 'slack' from /Users/trupp/sr.
↳...
88233 1512328570.32250: assigned :doc
88233 1512328570.32289: Loaded config def from plugin (callback/slack)
88233 1512328570.32337: Loading CallbackModule 'stderr' from /Users/trupp/s.
↳...
88233 1512328570.32782: assigned :doc
88233 1512328570.33061: Loading CallbackModule 'syslog_json' from /Users/tr.
↳...
88233 1512328570.34292: assigned :doc
88233 1512328570.34341: Loaded config def from plugin (callback/syslog_json)
88233 1512328570.34432: Loading CallbackModule 'timer' from /Users/trupp/sr.
↳...
88233 1512328570.34699: assigned :doc
88233 1512328570.34755: Loading CallbackModule 'tree' from /Users/trupp/src.
↳...
88233 1512328570.35199: assigned :doc
88233 1512328570.35319: Loading CallbackModule 'unixy' from /Users/trupp/sr.
↳...
88233 1512328570.35797: assigned :doc
88233 1512328570.35953: Loading CallbackModule 'yaml' from /Users/trupp/src.
↳...
88233 1512328570.36363: assigned :doc
88233 1512328570.36411: in VariableManager get_vars()
88233 1512328570.37284: Loading FilterModule 'core' from /Users/trupp/src/e.
↳...
88233 1512328570.39222: Loading FilterModule 'ipaddr' from /Users/trupp/src.
↳...

```

```

88233 1512328570.39913: Loading FilterModule 'json_query' from /Users/trupp.
↳...
88233 1512328570.40022: Loading FilterModule 'mathstuff' from /Users/trupp/.
↳...
88233 1512328570.40233: Loading FilterModule 'network' from /Users/trupp/sr.
↳...
88233 1512328570.40287: Loading FilterModule 'urlsplit' from /Users/trupp/s.
↳...
88233 1512328570.40499: Loading TestModule 'core' from /Users/trupp/src/env.
↳...
88233 1512328570.40560: Loading TestModule 'files' from /Users/trupp/src/en.
↳...
88233 1512328570.40642: Loading TestModule 'mathstuff' from /Users/trupp/sr.
↳...
88233 1512328570.41209: done with get_vars()
88233 1512328570.41286: in VariableManager get_vars()
88233 1512328570.41373: Loading FilterModule 'core' from /Users/trupp/src/e.
↳...
88233 1512328570.41384: Loading FilterModule 'ipaddr' from /Users/trupp/src.
↳...
88233 1512328570.41394: Loading FilterModule 'json_query' from /Users/trupp.
↳...
88233 1512328570.41402: Loading FilterModule 'mathstuff' from /Users/trupp/.
↳...
88233 1512328570.41410: Loading FilterModule 'network' from /Users/trupp/sr.
↳...
88233 1512328570.41418: Loading FilterModule 'urlsplit' from /Users/trupp/s.
↳...
88233 1512328570.41479: Loading TestModule 'core' from /Users/trupp/src/env.
↳...
88233 1512328570.41487: Loading TestModule 'files' from /Users/trupp/src/en.
↳...
88233 1512328570.41500: Loading TestModule 'mathstuff' from /Users/trupp/sr.
↳...
88233 1512328570.41917: done with get_vars()

PLAY [Labb 4.5] *****
88233 1512328570.43407: Loading StrategyModule 'linear' from /Users/trupp/s.
↳...
88233 1512328570.43460: getting the remaining hosts for this loop
88233 1512328570.43472: done getting the remaining hosts for this loop
88233 1512328570.43483: building list of next tasks for hosts
88233 1512328570.43491: getting the next task for host localhost
88233 1512328570.43504: done getting next task for host localhost
88233 1512328570.43514: ^ task is: TASK: Gathering Facts
88233 1512328570.43522: ^ state is: HOST STATE: block=0, task=0, rescue=0,.
↳...
88233 1512328570.43529: done building task lists
88233 1512328570.43535: counting tasks in each state of execution
88233 1512328570.43541: done counting tasks in each state of execution:
    num_setups: 1
    num_tasks: 0
    num_rescue: 0
    num_always: 0
88233 1512328570.43545: advancing hosts in ITERATING_SETUP
88233 1512328570.43549: starting to advance hosts
88233 1512328570.43553: getting the next task for host localhost
88233 1512328570.43558: done getting next task for host localhost

```

```

88233 1512328570.43562: ^ task is: TASK: Gathering Facts
88233 1512328570.43566: ^ state is: HOST STATE: block=0, task=0, rescue=0,
↪...
88233 1512328570.43571: done advancing hosts to next task
88233 1512328570.43578: getting variables
88233 1512328570.43583: in VariableManager get_vars()
88233 1512328570.43621: Loading FilterModule 'core' from /Users/trupp/src/e.
↪...
88233 1512328570.43630: Loading FilterModule 'ipaddr' from /Users/trupp/src.
↪...
88233 1512328570.43641: Loading FilterModule 'json_query' from /Users/trupp.
↪...
88233 1512328570.43650: Loading FilterModule 'mathstuff' from /Users/trupp/.
↪...
88233 1512328570.43658: Loading FilterModule 'network' from /Users/trupp/sr.
↪...
88233 1512328570.43667: Loading FilterModule 'urlsplit' from /Users/trupp/s.
↪...
88233 1512328570.43696: Loading TestModule 'core' from /Users/trupp/src/env.
↪...
88233 1512328570.43708: Loading TestModule 'files' from /Users/trupp/src/en.
↪...
88233 1512328570.43716: Loading TestModule 'mathstuff' from /Users/trupp/sr.
↪...
88233 1512328570.43867: Calling all_inventory to load vars for localhost
88233 1512328570.43888: Calling groups_inventory to load vars for localhost
88233 1512328570.43901: Calling all_plugins_inventory to load vars for_
↪localhost
88233 1512328570.44240: Loading VarsModule 'host_group_vars' from /Users/tr.
↪...
88233 1512328570.44275: Calling all_plugins_play to load vars for localhost
88233 1512328570.44303: Loading VarsModule 'host_group_vars' from /Users/tr.
↪...
88233 1512328570.44325: Calling groups_plugins_inventory to load vars for_
↪localhost
88233 1512328570.44354: Loading VarsModule 'host_group_vars' from /Users/tr.
↪...
88233 1512328570.44383: Calling groups_plugins_play to load vars for_
↪localhost
88233 1512328570.46982: Loading VarsModule 'host_group_vars' from /Users/tr.
↪...
88233 1512328570.47033: Loading VarsModule 'host_group_vars' from /Users/tr.
↪...
88233 1512328570.47063: Loading VarsModule 'host_group_vars' from /Users/tr.
↪...
88233 1512328570.47104: done with get_vars()
88233 1512328570.47132: done getting variables
88233 1512328570.47143: sending task start callback, copying the task so we.
↪...
88233 1512328570.47154: done copying, going to template now
88233 1512328570.47164: done templating
88233 1512328570.47171: here goes the callback...

TASK [Gathering Facts] *****
88233 1512328570.47183: sending task start callback
88233 1512328570.47190: entering _queue_task() for localhost/setup
88233 1512328570.47339: worker is 1 (out of 1 available)
88233 1512328570.47410: exiting _queue_task() for localhost/setup

```

```

88233 1512328570.47435: done queuing things up, now waiting for results_
↳queue to drain
88233 1512328570.47451: waiting for pending results...
88247 1512328570.47777: running TaskExecutor() for localhost/TASK:_
↳Gathering Facts
88247 1512328570.47883: in run() - task 8c85904d-91d6-70e5-2197-000000000011
88247 1512328570.48303: calling self._execute()
88247 1512328570.49597: Loading Connection 'local' from /Users/trupp/src/
↳env...
88247 1512328570.49687: Loading ShellModule 'csh' from /Users/trupp/src/
↳envs...
88247 1512328570.49787: Loading ShellModule 'fish' from /Users/trupp/src/
↳env...
88247 1512328570.49806: Loading ShellModule 'powershell' from /Users/trupp/
↳s...
88247 1512328570.49822: Loading ShellModule 'sh' from /Users/trupp/src/envs/
↳...
88247 1512328570.49917: Loading ShellModule 'sh' from /Users/trupp/src/envs/
↳...
88247 1512328570.50658: assigned :doc
88247 1512328570.50814: Loading ActionModule 'normal' from /Users/trupp/src/
↳...
88247 1512328570.50831: starting attempt loop
88247 1512328570.50838: running the handler
88247 1512328570.50930: ANSIBALLZ: Using lock for setup
88247 1512328570.50939: ANSIBALLZ: Acquiring lock
88247 1512328570.50950: ANSIBALLZ: Lock acquired: 4534992464
88247 1512328570.50962: ANSIBALLZ: Creating module
88247 1512328570.85142: ANSIBALLZ: Writing module
88247 1512328570.85214: ANSIBALLZ: Renaming module
88247 1512328570.85245: ANSIBALLZ: Done creating module
88247 1512328570.85407: _low_level_execute_command(): starting
88247 1512328570.85415: _low_level_execute_command(): executing: /bin/sh -c
↳'echo ~ && sleep 0'
88247 1512328570.85429: in local.exec_command()
88247 1512328570.85435: opening command with Popen()
88247 1512328570.85823: done running command with Popen()
88247 1512328570.85842: getting output with communicate()
88247 1512328570.86905: done communicating
88247 1512328570.86927: done with local.exec_command()
88247 1512328570.86946: _low_level_execute_command() done: rc=0, stdout=/
↳Users/trupp
, stderr=
88247 1512328570.86958: _low_level_execute_command(): starting
88247 1512328570.86967: _low_level_execute_command(): executing: /bin/sh -c
↳'(...
88247 1512328570.86979: in local.exec_command()
88247 1512328570.86985: opening command with Popen()
88247 1512328570.87401: done running command with Popen()
88247 1512328570.87426: getting output with communicate()
88247 1512328570.89015: done communicating
88247 1512328570.89025: done with local.exec_command()
88247 1512328570.89042: _low_level_execute_command() done: rc=0,_
↳stdout=ansibl...
, stderr=
88247 1512328570.89055: transferring module to remote /Users/trupp/.ansible/
↳tm...
88247 1512328570.89245: done transferring module to remote

```

```

88247 1512328570.89266: _low_level_execute_command(): starting
88247 1512328570.89273: _low_level_execute_command(): executing: /bin/sh -c
↪ 'c...
88247 1512328570.89283: in local.exec_command()
88247 1512328570.89288: opening command with Popen()
88247 1512328570.89634: done running command with Popen()
88247 1512328570.89665: getting output with communicate()
88247 1512328570.91161: done communicating
88247 1512328570.91171: done with local.exec_command()
88247 1512328570.91192: _low_level_execute_command() done: rc=0, stdout=,
↪ stderr=
88247 1512328570.91200: _low_level_execute_command(): starting
88247 1512328570.91211: _low_level_execute_command(): executing: /bin/sh -c
↪ '...
88247 1512328570.91223: in local.exec_command()
88247 1512328570.91229: opening command with Popen()
88247 1512328570.91581: done running command with Popen()
88247 1512328570.91614: getting output with communicate()
88247 1512328571.28618: done communicating
88247 1512328571.28630: done with local.exec_command()
88247 1512328571.28655: _low_level_execute_command() done: rc=0, stdout=
{"invocation": {"module_args": {"filter": "*", "gather_subset": ["all"],
↪ "fact...
, stderr=
88247 1512328571.29273: done with _execute_module (setup, {'_ansible_version
↪ ':...
88247 1512328571.29291: handler run complete
88247 1512328571.34550: attempt loop complete, returning result
88247 1512328571.34576: _execute() done
88247 1512328571.34583: dumping result to json
88247 1512328571.34671: done dumping result, returning
88247 1512328571.34683: done running TaskExecutor() for localhost/TASK:
↪ Gather...
88247 1512328571.34699: sending task result for task 8c85904d-91d6-70e5-
↪ 2197-0...
88247 1512328571.34737: done sending task result for task 8c85904d-91d6-
↪ 70e5-2...
88247 1512328571.35092: WORKER PROCESS EXITING
ok: [localhost]
88233 1512328571.36570: no more pending results, returning what we have
88233 1512328571.36579: results queue empty
88233 1512328571.36583: checking for any_errors_fatal
88233 1512328571.36588: done checking for any_errors_fatal
88233 1512328571.36592: checking for max_fail_percentage
88233 1512328571.36596: done checking for max_fail_percentage
88233 1512328571.36600: checking to see if all hosts have failed and the
↪ runn...
88233 1512328571.36604: done checking to see if all hosts have failed
88233 1512328571.36608: getting the remaining hosts for this loop
88233 1512328571.36616: done getting the remaining hosts for this loop
88233 1512328571.36626: building list of next tasks for hosts
88233 1512328571.36631: getting the next task for host localhost
88233 1512328571.36638: done getting next task for host localhost
88233 1512328571.36644: ^ task is: TASK: meta (flush_handlers)
88233 1512328571.37533: ^ state is: HOST STATE: block=1, task=1, rescue=0,
↪ alw...
88233 1512328571.37544: done building task lists
88233 1512328571.37549: counting tasks in each state of execution

```

```

88233 1512328571.37555: done counting tasks in each state of execution:
    num_setups: 0
    num_tasks: 1
    num_rescue: 0
    num_always: 0
88233 1512328571.37567: advancing hosts in ITERATING_TASKS
88233 1512328571.37572: starting to advance hosts
88233 1512328571.37576: getting the next task for host localhost
88233 1512328571.37583: done getting next task for host localhost
88233 1512328571.37589: ^ task is: TASK: meta (flush_handlers)
88233 1512328571.37594: ^ state is: HOST STATE: block=1, task=1, rescue=0,
↳alwa...
88233 1512328571.37600: done advancing hosts to next task
88233 1512328571.37619: done queuing things up, now waiting for results
↳queue to...
88233 1512328571.37626: results queue empty
88233 1512328571.37631: checking for any_errors_fatal
88233 1512328571.37636: done checking for any_errors_fatal
88233 1512328571.37641: checking for max_fail_percentage
88233 1512328571.37646: done checking for max_fail_percentage
88233 1512328571.37650: checking to see if all hosts have failed and the
↳running result is not ok
88233 1512328571.37655: done checking to see if all hosts have failed
88233 1512328571.37660: getting the remaining hosts for this loop
88233 1512328571.37669: done getting the remaining hosts for this loop
88233 1512328571.37680: building list of next tasks for hosts
88233 1512328571.37686: getting the next task for host localhost
88233 1512328571.37698: done getting next task for host localhost
88233 1512328571.37705: ^ task is: TASK: Run a task
88233 1512328571.37710: ^ state is: HOST STATE: block=2, task=1, rescue=0,
↳always=0,...
88233 1512328571.37715: done building task lists
88233 1512328571.37720: counting tasks in each state of execution
88233 1512328571.37725: done counting tasks in each state of execution:
    num_setups: 0
    num_tasks: 1
    num_rescue: 0
    num_always: 0
88233 1512328571.37732: advancing hosts in ITERATING_TASKS
88233 1512328571.37736: starting to advance hosts
88233 1512328571.37741: getting the next task for host localhost
88233 1512328571.37748: done getting next task for host localhost
88233 1512328571.37754: ^ task is: TASK: Run a task
88233 1512328571.37759: ^ state is: HOST STATE: block=2, task=1, rescue=0,
↳always=0,...
88233 1512328571.37764: done advancing hosts to next task
88233 1512328571.37964: Loading ActionModule 'set_fact' from /Users/trupp/
↳src/envs/f5...
88233 1512328571.37977: getting variables
88233 1512328571.37985: in VariableManager get_vars()
88233 1512328571.38064: Loading FilterModule 'core' from /Users/trupp/src/
↳envs/f5ansi...
88233 1512328571.38074: Loading FilterModule 'ipaddr' from /Users/trupp/src/
↳envs/f5an...
88233 1512328571.38082: Loading FilterModule 'json_query' from /Users/trupp/
↳src/envs/...
88233 1512328571.38088: Loading FilterModule 'mathstuff' from /Users/trupp/
↳src/envs/f...

```

```

88233 1512328571.38095: Loading FilterModule 'network' from /Users/trupp/
↳src/envs/f5a...
88233 1512328571.38102: Loading FilterModule 'urlsplit' from /Users/trupp/
↳src/envs/f5...
88233 1512328571.38135: Loading TestModule 'core' from /Users/trupp/src/
↳envs/f5ansibl...
88233 1512328571.38142: Loading TestModule 'files' from /Users/trupp/src/
↳envs/f5ansib...
88233 1512328571.38148: Loading TestModule 'mathstuff' from /Users/trupp/
↳src/envs/f5a...
88233 1512328571.38235: Calling all_inventory to load vars for localhost
88233 1512328571.38246: Calling groups_inventory to load vars for localhost
88233 1512328571.38253: Calling all_plugins_inventory to load vars for
↳localhost
88233 1512328571.38277: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/en...
88233 1512328571.38305: Calling all_plugins_play to load vars for localhost
88233 1512328571.38323: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/en...
88233 1512328571.38344: Calling groups_plugins_inventory to load vars for
↳localhost
88233 1512328571.38365: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/en...
88233 1512328571.38386: Calling groups_plugins_play to load vars for
↳localhost
88233 1512328571.38405: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/en...
88233 1512328571.38440: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/en...
88233 1512328571.38472: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/en...
88233 1512328571.39665: done with get_vars()
88233 1512328571.39684: done getting variables
88233 1512328571.39691: sending task start callback, copying the task so we
↳can template
88233 1512328571.39696: done copying, going to template now
88233 1512328571.39702: done templating
88233 1512328571.39706: here goes the callback...

TASK [Run a task]
↳*****
88233 1512328571.39718: sending task start callback
88233 1512328571.39723: entering _queue_task() for localhost/set_fact
88233 1512328571.39728: Creating lock for set_fact
88233 1512328571.39884: worker is 1 (out of 1 available)
88233 1512328571.39947: exiting _queue_task() for localhost/set_fact
88233 1512328571.39969: done queuing things up, now waiting for results
↳queue to drain
88233 1512328571.39976: waiting for pending results...
88286 1512328571.40364: running TaskExecutor() for localhost/TASK: Run a
↳task
88286 1512328571.40509: in run() - task 8c85904d-91d6-70e5-2197-000000000008
88286 1512328571.40615: calling self._execute()
88286 1512328571.41878: Loading Connection 'local' from /Users/trupp/src/
↳envs/f5ansible/...
88286 1512328571.41995: Loading ShellModule 'csh' from /Users/trupp/src/
↳envs/f5ansible/l...
88286 1512328571.42067: Loading ShellModule 'fish' from /Users/trupp/src/
↳envs/f5ansible/...

```

```

88286 1512328571.42078: Loading ShellModule 'powershell' from /Users/trupp/
↳src/envs/f5ans...
88286 1512328571.42086: Loading ShellModule 'sh' from /Users/trupp/src/envs/
↳f5ansible/li...
88286 1512328571.42133: Loading ShellModule 'sh' from /Users/trupp/src/envs/
↳f5ansible/li...
88286 1512328571.42792: assigned :doc
88286 1512328571.42850: Loading ActionModule 'set_fact' from /Users/trupp/
↳src/envs/f5ans...
88286 1512328571.42863: starting attempt loop
88286 1512328571.42872: running the handler
88286 1512328571.42889: handler run complete
88286 1512328571.43127: attempt loop complete, returning result
88286 1512328571.43133: _execute() done
88286 1512328571.43137: dumping result to json
88286 1512328571.43142: done dumping result, returning
88286 1512328571.43148: done running TaskExecutor() for localhost/TASK: Run_
↳a task [8c85...
88286 1512328571.43160: sending task result for task 8c85904d-91d6-70e5-
↳2197-0000000000008
88286 1512328571.43188: done sending task result for task 8c85904d-91d6-
↳70e5-2197-0000000000008
88286 1512328571.43216: WORKER PROCESS EXITING
ok: [localhost]
88233 1512328571.43625: no more pending results, returning what we have
88233 1512328571.43638: results queue empty
88233 1512328571.43643: checking for any_errors_fatal
88233 1512328571.43650: done checking for any_errors_fatal
88233 1512328571.43655: checking for max_fail_percentage
88233 1512328571.43660: done checking for max_fail_percentage
88233 1512328571.43665: checking to see if all hosts have failed and the_
↳running result is not ok
88233 1512328571.43669: done checking to see if all hosts have failed
88233 1512328571.43674: getting the remaining hosts for this loop
88233 1512328571.43685: done getting the remaining hosts for this loop
88233 1512328571.43699: building list of next tasks for hosts
88233 1512328571.43706: getting the next task for host localhost
88233 1512328571.43717: done getting next task for host localhost
88233 1512328571.43724: ^ task is: TASK: Run a second task
88233 1512328571.43731: ^ state is: HOST STATE: block=2, task=2, rescue=0,
↳always=0, ru...
88233 1512328571.43742: done building task lists
88233 1512328571.43747: counting tasks in each state of execution
88233 1512328571.43753: done counting tasks in each state of execution:
    num_setups: 0
    num_tasks: 1
    num_rescue: 0
    num_always: 0
88233 1512328571.43771: advancing hosts in ITERATING_TASKS
88233 1512328571.43776: starting to advance hosts
88233 1512328571.43781: getting the next task for host localhost
88233 1512328571.43788: done getting next task for host localhost
88233 1512328571.43794: ^ task is: TASK: Run a second task
88233 1512328571.43800: ^ state is: HOST STATE: block=2, task=2, rescue=0,
↳always=0, ru...
88233 1512328571.43805: done advancing hosts to next task
88233 1512328571.44173: Loading ActionModule 'debug' from /Users/trupp/src/
↳envs/f5ansibl...

```

```

88233 1512328571.44187: getting variables
88233 1512328571.44193: in VariableManager get_vars()
88233 1512328571.44265: Loading FilterModule 'core' from /Users/trupp/src/
↳envs/f5ansible...
88233 1512328571.44279: Loading FilterModule 'ipaddr' from /Users/trupp/src/
↳envs/f5ansib...
88233 1512328571.44288: Loading FilterModule 'json_query' from /Users/trupp/
↳src/envs/f5a...
88233 1512328571.44297: Loading FilterModule 'mathstuff' from /Users/trupp/
↳src/envs/f5an...
88233 1512328571.44305: Loading FilterModule 'network' from /Users/trupp/
↳src/envs/f5ansi...
88233 1512328571.44313: Loading FilterModule 'urlsplit' from /Users/trupp/
↳src/envs/f5ans...
88233 1512328571.44355: Loading TestModule 'core' from /Users/trupp/src/
↳envs/f5ansible/l...
88233 1512328571.44364: Loading TestModule 'files' from /Users/trupp/src/
↳envs/f5ansible/...
88233 1512328571.44371: Loading TestModule 'mathstuff' from /Users/trupp/
↳src/envs/f5ansi...
88233 1512328571.44496: Calling all_inventory to load vars for localhost
88233 1512328571.44510: Calling groups_inventory to load vars for localhost
88233 1512328571.44519: Calling all_plugins_inventory to load vars for_
↳localhost
88233 1512328571.44550: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.44577: Calling all_plugins_play to load vars for localhost
88233 1512328571.44596: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.44618: Calling groups_plugins_inventory to load vars for_
↳localhost
88233 1512328571.44639: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.44660: Calling groups_plugins_play to load vars for_
↳localhost
88233 1512328571.44679: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.44716: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.44748: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.46020: done with get_vars()
88233 1512328571.46053: done getting variables
88233 1512328571.46063: sending task start callback, copying the task so we_
↳can template...
88233 1512328571.46068: done copying, going to template now
88233 1512328571.46075: done templating
88233 1512328571.46080: here goes the callback...

TASK [Run a second task]_
↳*****
88233 1512328571.46094: sending task start callback
88233 1512328571.46101: entering _queue_task() for localhost/debug
88233 1512328571.46107: Creating lock for debug
88233 1512328571.46271: worker is 1 (out of 1 available)
88233 1512328571.46329: exiting _queue_task() for localhost/debug
88233 1512328571.46354: done queuing things up, now waiting for results_
↳queue to drain

```

```

88233 1512328571.46360: waiting for pending results...
88287 1512328571.46821: running TaskExecutor() for localhost/TASK: Run a
↳second task
88287 1512328571.46936: in run() - task 8c85904d-91d6-70e5-2197-00000000000a
88287 1512328571.47046: calling self._execute()
88287 1512328571.47350: Loading FilterModule 'core' from /Users/trupp/src/
↳envs/f5ansible...
88287 1512328571.47365: Loading FilterModule 'ipaddr' from /Users/trupp/src/
↳envs/f5ansib...
88287 1512328571.47375: Loading FilterModule 'json_query' from /Users/trupp/
↳src/envs/f5a...
88287 1512328571.47383: Loading FilterModule 'mathstuff' from /Users/trupp/
↳src/envs/f5an...
88287 1512328571.47391: Loading FilterModule 'network' from /Users/trupp/
↳src/envs/f5ansi...
88287 1512328571.47399: Loading FilterModule 'urlsplit' from /Users/trupp/
↳src/envs/f5ans...
88287 1512328571.47735: Loading TestModule 'core' from /Users/trupp/src/
↳envs/f5ansible/l...
88287 1512328571.47745: Loading TestModule 'files' from /Users/trupp/src/
↳envs/f5ansible/...
88287 1512328571.47753: Loading TestModule 'mathstuff' from /Users/trupp/
↳src/envs/f5ansi...
88287 1512328571.48550: when evaluation is False, skipping this task
88287 1512328571.48563: _execute() done
88287 1512328571.48571: dumping result to json
88287 1512328571.48580: done dumping result, returning
88287 1512328571.48588: done running TaskExecutor() for localhost/TASK: Run
↳a second tas...
88287 1512328571.48607: sending task result for task 8c85904d-91d6-70e5-
↳2197-00000000000a
88287 1512328571.48641: done sending task result for task 8c85904d-91d6-
↳70e5-2197-00000000000a
88287 1512328571.48649: WORKER PROCESS EXITING
skipping: [localhost]
88233 1512328571.49095: no more pending results, returning what we have
88233 1512328571.49106: results queue empty
88233 1512328571.49111: checking for any_errors_fatal
88233 1512328571.49131: done checking for any_errors_fatal
88233 1512328571.49140: checking for max_fail_percentage
88233 1512328571.49146: done checking for max_fail_percentage
88233 1512328571.49151: checking to see if all hosts have failed and the
↳running result is not ok
88233 1512328571.49157: done checking to see if all hosts have failed
88233 1512328571.49162: getting the remaining hosts for this loop
88233 1512328571.49190: done getting the remaining hosts for this loop
88233 1512328571.49200: building list of next tasks for hosts
88233 1512328571.49208: getting the next task for host localhost
88233 1512328571.49220: done getting next task for host localhost
88233 1512328571.49228: ^ task is: TASK: Run a third task
88233 1512328571.49239: ^ state is: HOST STATE: block=2, task=3, rescue=0,
↳always=0, ru...
88233 1512328571.49249: done building task lists
88233 1512328571.49254: counting tasks in each state of execution
88233 1512328571.49260: done counting tasks in each state of execution:
    num_setups: 0
    num_tasks: 1
    num_rescue: 0

```

```

num_always: 0
88233 1512328571.49273: advancing hosts in ITERATING_TASKS
88233 1512328571.49278: starting to advance hosts
88233 1512328571.49283: getting the next task for host localhost
88233 1512328571.49289: done getting next task for host localhost
88233 1512328571.49295: ^ task is: TASK: Run a third task
88233 1512328571.49302: ^ state is: HOST STATE: block=2, task=3, rescue=0,
↳always=0, ru...
88233 1512328571.49308: done advancing hosts to next task
88233 1512328571.49899: Loading ActionModule 'debug' from /Users/trupp/src/
↳envs/f5ansibl...
88233 1512328571.49921: getting variables
88233 1512328571.49930: in VariableManager get_vars()
88233 1512328571.50084: Loading FilterModule 'core' from /Users/trupp/src/
↳envs/f5ansible...
88233 1512328571.50096: Loading FilterModule 'ipaddr' from /Users/trupp/src/
↳envs/f5ansib...
88233 1512328571.50104: Loading FilterModule 'json_query' from /Users/trupp/
↳src/envs/f5a...
88233 1512328571.50111: Loading FilterModule 'mathstuff' from /Users/trupp/
↳src/envs/f5an...
88233 1512328571.50118: Loading FilterModule 'network' from /Users/trupp/
↳src/envs/f5ansi...
88233 1512328571.50124: Loading FilterModule 'urlsplit' from /Users/trupp/
↳src/envs/f5ans...
88233 1512328571.50166: Loading TestModule 'core' from /Users/trupp/src/
↳envs/f5ansible/l...
88233 1512328571.50174: Loading TestModule 'files' from /Users/trupp/src/
↳envs/f5ansible/...
88233 1512328571.50181: Loading TestModule 'mathstuff' from /Users/trupp/
↳src/envs/f5ansi...
88233 1512328571.50273: Calling all_inventory to load vars for localhost
88233 1512328571.50283: Calling groups_inventory to load vars for localhost
88233 1512328571.50291: Calling all_plugins_inventory to load vars for
↳localhost
88233 1512328571.50321: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.50370: Calling all_plugins_play to load vars for localhost
88233 1512328571.50407: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.50438: Calling groups_plugins_inventory to load vars for
↳localhost
88233 1512328571.50469: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.50494: Calling groups_plugins_play to load vars for
↳localhost
88233 1512328571.50516: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.50558: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.50594: Loading VarsModule 'host_group_vars' from /Users/
↳trupp/src/envs/...
88233 1512328571.51987: done with get_vars()
88233 1512328571.52010: done getting variables
88233 1512328571.52019: sending task start callback, copying the task so we
↳can template...
88233 1512328571.52025: done copying, going to template now
88233 1512328571.52033: done templating

```

```

88233 1512328571.52047: here goes the callback...

TASK [Run a third task]_
↳*****
88233 1512328571.52066: sending task start callback
88233 1512328571.52073: entering _queue_task() for localhost/debug
88233 1512328571.52246: worker is 1 (out of 1 available)
88233 1512328571.52320: exiting _queue_task() for localhost/debug
88233 1512328571.52345: done queuing things up, now waiting for results_
↳queue to drain
88233 1512328571.52351: waiting 88288 1512328571.52817: running_
↳TaskExecutor() for loca...
for pending results...
88288 1512328571.53010: in run() - task 8c85904d-91d6-70e5-2197-00000000000c
88288 1512328571.53117: calling self._execute()
88288 1512328571.53492: Loading FilterModule 'core' from /Users/trupp/src/
↳envs/f5ansible...
88288 1512328571.53523: Loading FilterModule 'ipaddr' from /Users/trupp/src/
↳envs/f5ansib...
88288 1512328571.53539: Loading FilterModule 'json_query' from /Users/trupp/
↳src/envs/f5a...
88288 1512328571.53551: Loading FilterModule 'mathstuff' from /Users/trupp/
↳src/envs/f5an...
88288 1512328571.53562: Loading FilterModule 'network' from /Users/trupp/
↳src/envs/f5ansi...
88288 1512328571.53576: Loading FilterModule 'urlsplit' from /Users/trupp/
↳src/envs/f5ans...
88288 1512328571.53665: Loading TestModule 'core' from /Users/trupp/src/
↳envs/f5ansible/l...
88288 1512328571.53676: Loading TestModule 'files' from /Users/trupp/src/
↳envs/f5ansible/...
88288 1512328571.53683: Loading TestModule 'mathstuff' from /Users/trupp/
↳src/envs/f5ansi...
88288 1512328571.55223: Loading Connection 'local' from /Users/trupp/src/
↳envs/f5ansible/...
88288 1512328571.55376: Loading ShellModule 'csh' from /Users/trupp/src/
↳envs/f5ansible/l...
88288 1512328571.55476: Loading ShellModule 'fish' from /Users/trupp/src/
↳envs/f5ansible/...
88288 1512328571.55497: Loading ShellModule 'powershell' from /Users/trupp/
↳src/envs/f5an...
88288 1512328571.55509: Loading ShellModule 'sh' from /Users/trupp/src/envs/
↳f5ansible/li...
88288 1512328571.55560: Loading ShellModule 'sh' from /Users/trupp/src/envs/
↳f5ansible/li...
88288 1512328571.56124: assigned :doc
88288 1512328571.56194: Loading ActionModule 'debug' from /Users/trupp/src/
↳envs/f5ansibl...
88288 1512328571.56211: starting attempt loop
88288 1512328571.56218: running the handler
88288 1512328571.56362: handler run complete
88288 1512328571.56372: attempt loop complete, returning result
88288 1512328571.56380: _execute() done
88288 1512328571.56385: dumping result to json
88288 1512328571.56392: done dumping result, returning
88288 1512328571.56405: done running TaskExecutor() for localhost/TASK: Run_
↳a third task...
88288 1512328571.56423: sending task result for task 8c85904d-91d6-70e5-
↳2197-00000000000c

```

```

88288 1512328571.56471: done sending task result for task 8c85904d-91d6-
↳70e5-2197-000000000000c
88288 1512328571.56505: WORKER PROCESS EXITING
ok: [localhost] => {
    "fact1": "foo"
}
88233 1512328571.57041: no more pending results, returning what we have
88233 1512328571.57072: results queue empty
88233 1512328571.57086: checking for any_errors_fatal
88233 1512328571.57113: done checking for any_errors_fatal
88233 1512328571.57123: checking for max_fail_percentage
88233 1512328571.57132: done checking for max_fail_percentage
88233 1512328571.57138: checking to see if all hosts have failed and the
↳running result is not ok
88233 1512328571.57149: done checking to see if all hosts have failed
88233 1512328571.57159: getting the remaining hosts for this loop
88233 1512328571.57201: done getting the remaining hosts for this loop
88233 1512328571.57223: building list of next tasks for hosts
88233 1512328571.57235: getting the next task for host localhost
88233 1512328571.57270: done getting next task for host localhost
88233 1512328571.57284: ^ task is: TASK: meta (flush_handlers)
88233 1512328571.57306: ^ state is: HOST STATE: block=3, task=1, rescue=0,
↳always=0, ru...
88233 1512328571.57320: done building task lists
88233 1512328571.57326: counting tasks in each state of execution
88233 1512328571.57335: done counting tasks in each state of execution:
    num_setups: 0
    num_tasks: 1
    num_rescue: 0
    num_always: 0
88233 1512328571.57356: advancing hosts in ITERATING_TASKS
88233 1512328571.57372: starting to advance hosts
88233 1512328571.57378: getting the next task for host localhost
88233 1512328571.57389: done getting next task for host localhost
88233 1512328571.57396: ^ task is: TASK: meta (flush_handlers)
88233 1512328571.57402: ^ state is: HOST STATE: block=3, task=1, rescue=0,
↳always=0, run...
88233 1512328571.57418: done advancing hosts to next task
88233 1512328571.57485: done queuing things up, now waiting for results
↳queue to drain
88233 1512328571.57492: results queue empty
88233 1512328571.57497: checking for any_errors_fatal
88233 1512328571.57503: done checking for any_errors_fatal
88233 1512328571.57507: checking for max_fail_percentage
88233 1512328571.57512: done checking for max_fail_percentage
88233 1512328571.57517: checking to see if all hosts have failed and the
↳running result is not ok
88233 1512328571.57521: done checking to see if all hosts have failed
88233 1512328571.57526: getting the remaining hosts for this loop
88233 1512328571.57532: done getting the remaining hosts for this loop
88233 1512328571.57540: building list of next tasks for hosts
88233 1512328571.57545: getting the next task for host localhost
88233 1512328571.57552: done getting next task for host localhost
88233 1512328571.57558: ^ task is: TASK: meta (flush_handlers)
88233 1512328571.57563: ^ state is: HOST STATE: block=4, task=1, rescue=0,
↳always=0, run_st...
88233 1512328571.57569: done building task lists
88233 1512328571.57573: counting tasks in each state of execution

```

```

88233 1512328571.57578: done counting tasks in each state of execution:
    num_setups: 0
    num_tasks: 1
    num_rescue: 0
    num_always: 0
88233 1512328571.57584: advancing hosts in ITERATING_TASKS
88233 1512328571.57589: starting to advance hosts
88233 1512328571.57594: getting the next task for host localhost
88233 1512328571.57600: done getting next task for host localhost
88233 1512328571.57606: ^ task is: TASK: meta (flush_handlers)
88233 1512328571.57611: ^ state is: HOST STATE: block=4, task=1, rescue=0,
↳always=0, run_st...
88233 1512328571.57616: done advancing hosts to next task
88233 1512328571.57626: done queuing things up, now waiting for results
↳queue to drain
88233 1512328571.57632: results queue empty
88233 1512328571.57637: checking for any_errors_fatal
88233 1512328571.57642: done checking for any_errors_fatal
88233 1512328571.57646: checking for max_fail_percentage
88233 1512328571.57651: done checking for max_fail_percentage
88233 1512328571.57656: checking to see if all hosts have failed and the
↳running result is not ok
88233 1512328571.57660: done checking to see if all hosts have failed
88233 1512328571.57665: getting the remaining hosts for this loop
88233 1512328571.57671: done getting the remaining hosts for this loop
88233 1512328571.57678: building list of next tasks for hosts
88233 1512328571.57683: getting the next task for host localhost
88233 1512328571.57689: done getting next task for host localhost
88233 1512328571.57694: ^ task is: None
88233 1512328571.57700: ^ state is: HOST STATE: block=5, task=0, rescue=0,
↳always=0, run_st...
88233 1512328571.57705: done building task lists
88233 1512328571.57710: counting tasks in each state of execution
88233 1512328571.57714: done counting tasks in each state of execution:
    num_setups: 0
    num_tasks: 0
    num_rescue: 0
    num_always: 0
88233 1512328571.57720: all hosts are done, so returning None's for all
↳hosts
88233 1512328571.57725: done queuing things up, now waiting for results
↳queue to drain
88233 1512328571.57730: results queue empty
88233 1512328571.57735: checking for any_errors_fatal
88233 1512328571.57739: done checking for any_errors_fatal
88233 1512328571.57744: checking for max_fail_percentage
88233 1512328571.57749: done checking for max_fail_percentage
88233 1512328571.57753: checking to see if all hosts have failed and the
↳running result is not ok
88233 1512328571.57758: done checking to see if all hosts have failed
88233 1512328571.57764: getting the next task for host localhost
88233 1512328571.57771: done getting next task for host localhost
88233 1512328571.57776: ^ task is: None
88233 1512328571.57781: ^ state is: HOST STATE: block=5, task=0, rescue=0,
↳always=0, run_st...
88233 1512328571.57787: running handlers

```

```

PLAY RECAP
↳*****

```

```
localhost : ok=3 changed=0 unreachable=0 failed=0
```

## Discussion

Debug output is not very useful unless you are debugging a core problem with Ansible. It is also useful, in some cases, when you need to debug a module.

The reason we are showing it to you here is because it may be requested of you when you report problems to the F5 Ansible developers.

Debug output shows the detailed execution of the Ansible engine as it processes the playbook and the modules.

## 3.4.6 Dealing with “bigsuds/f5-sdk not found” errors

### Problem

Your playbook is failing with an error about being “unable to find bigsuds/f5-sdk”, but you’re SURE they are installed

### Solution

There are three potential causes for this that we’ll cover. They are

- They’re not installed in the right spot
- You’re not using connection local
- You’re not using delegation

### Wrong installation location

If you are using a virtualenv, or a system that does not have python found at `/usr/bin/python`, you *must* set the python interpreter of that system for Ansible.

This might be `/usr/bin/python3` or the path to the virtualenv python like `ansible_python_interpreter=/.virtualenvs/lab4.6/bin/python`

See the discussion below for more information regarding this issue.

### Not using connection local

You have not specified `connection: local`. Therefore, all remote hosts will be connected to over SSH. Nine times out of ten the “not found” error is being raised because Ansible is connecting to a remote BIG-IP and the F5 module dependencies are not installed on the BIG-IP

*The F5 Ansible modules can not be installed on a BIG-IP*

Use `connection: local` for the play, or, `delegate_to: localhost` **for each BIG-IP task**

## Not using delegation

See the solution above as the reason and solution are the same, only this time you are missing `delegate_to: localhost` instead of `connection: local`

## Discussion

In an earlier lab we had, we learned that Ansible considers all your hosts to be remote. This includes when you running it in a virtualenv or use `connection: local`.

This “wrong installation location” problem rears its head primarily when you run Ansible in a virtualenv.

As a general rule of thumb, don’t do that unless you know what you’re doing. If you’re reading this lab, you probably don’t know what you’re doing.

To experience the error, we’ll use a contrived example with two virtualenvs; one with the dependencies installed, one without. These labs can be found in the **lab4.6** directory.

We have configured our hosts to reference two different python interpreters. The `broken` host, references a python interpreter that is intended to mimic your non-virtualenv remote system (I know its referencing a virtualenv, imagine with me here).

The `working` python interpreter is intended to mimic the virtualenv that you have installed Ansible in.

Change into the working virtualenv.

```
$ workon lab4.6
```

When you are in the virtualenv and the following shows `f5-sdk`,

```
(lab4.6) $ pip freeze | grep f5-sdk
f5-sdk==3.0.5
$
```

It is irrelevant. It is what is **in the remote system’s python installation that matters** because that is the python that is invoked by default when Ansible connects to a machine.

Let’s run the **lab4.6/playbooks/site.yaml** playbook now

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

It seems the playbook fails (output truncated)

```
TASK [Create a pool]
↳*****
An exception occurred during task execution. To see the full traceback, use -vvv. The error was: ImportError: No module named netaddr
fatal: [broken]: FAILED! => {"changed": false, "module_stderr": "Traceback
↳(most recent call last):\n  File \"/tmp/ansible__3fdUX/ans
  to retry, use: --limit @/root/f5-gsts-labs-ansible-cookbook/labs/
↳lab4.6/playbooks/site.retry
```

To fix this, change the `ansible_python_interpreter` line in `inventory/hosts` file to read

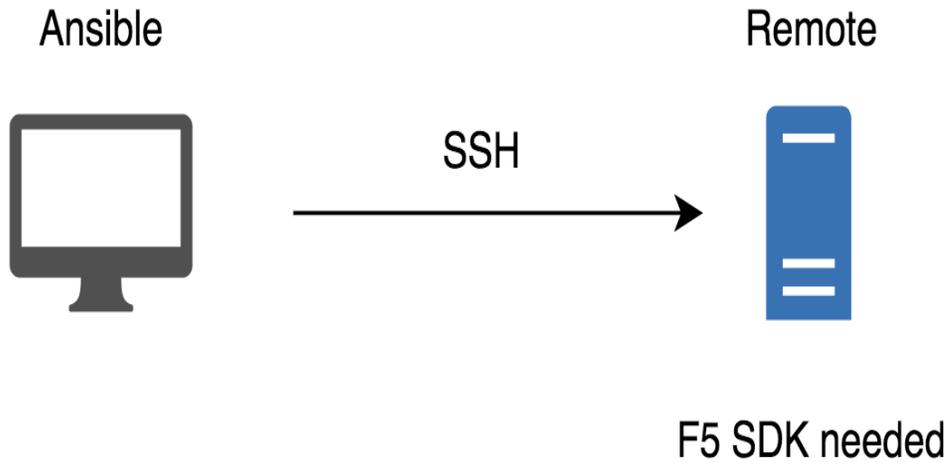
- `ansible_python_interpreter=/.virtualenvs/lab4.6/bin/python`

Let’s re-run the playbook now

```
$ ansible-playbook -i inventory/hosts playbooks/site.yaml
```

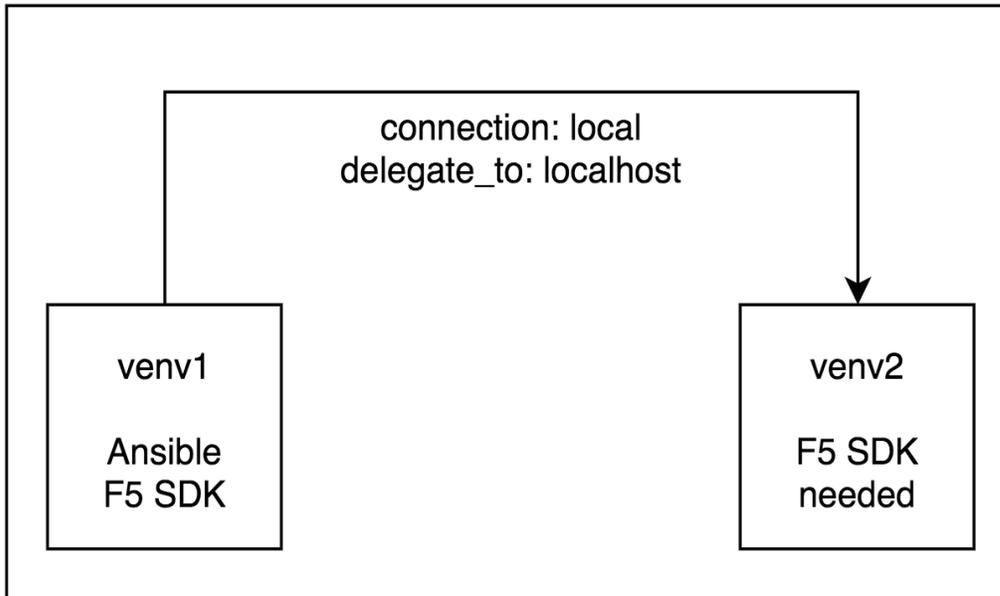
Take a moment to review the images below for a better understanding of what Ansible is doing.

This is how Ansible normally works



In the virtualenv situation above, what we had instead, is that the Ansible client had the F5 SDK installed in one virtualenv, but the remote host used a different virtualenv. Therefore, we had a similar situation as the picture above, but using virtualenv instead

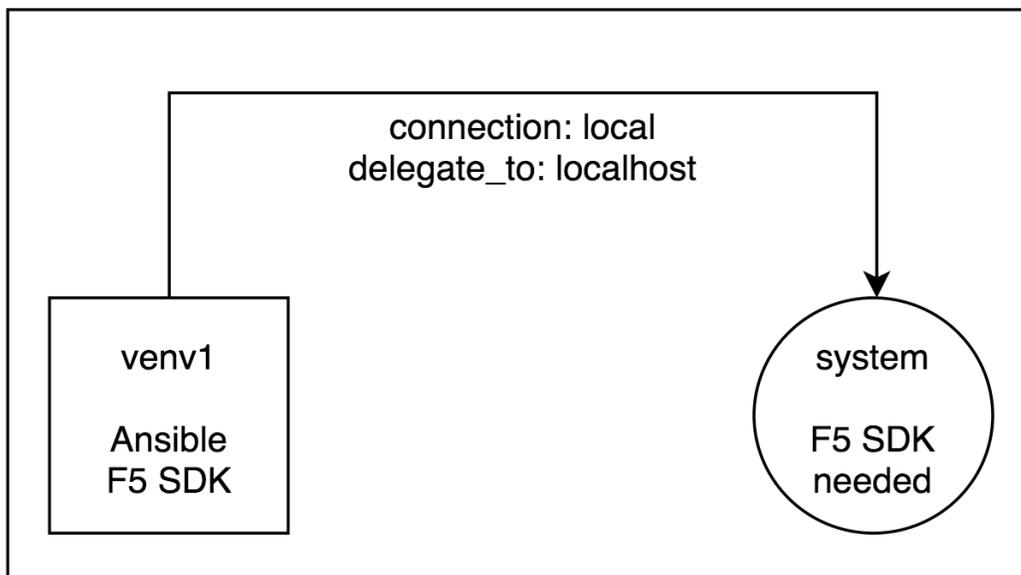
## Ansible Controller



As you can see, we have the F5 SDK installed in the venv we were using, but **not** in the venv that the remote host was configured for.

The same is implied when you are only using a single venv and the remote host specifies nothing. In that case, you will need the dependencies installed in the **system** python.

## Ansible Controller



### 3.4.7 Dealing with “authorization failed” errors

#### Problem

Your playbook is failing with an error about “F5 Authorization failed”

#### Solution

Either your password is wrong. But it isn't! Yes... for the last time... it is.

Or,

Your remote authentication is configured (on BIG-IP) incorrectly.

Or,

The role of the user that you are using with the Ansible module is not Administrator or an equivalent.

Ensure that your password is correct. There is a specific command that you should ensure runs without error, it is

```
$ curl -k -u admin:admin https://10.1.1.4/mgmt/tm/sys | jq .
```

Replace `admin:admin` with your `user`, and `password` combination. If the above command does not succeed, then it will not be possible for the F5 Ansible module to succeed. On some other versions of BIG-IP it was not possible for this to succeed.

A successful output will look like the following

```
(f5ansible) SEA-ML-00028116:labs trupp$ curl -k -u admin:admin https://localhost:10443/mgmt/tm/sys | jq .
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
100 3409  100 3409    0     0  56110      0  --:--:-- --:--:-- --:--:-- 56816
{
  "kind": "tm:sys:syscollectionstate",
  "selfLink": "https://localhost/mgmt/tm/sys?ver=12.1.0",
  "items": [
    {
      "reference": {
        "link": "https://localhost/mgmt/tm/sys/application?ver=12.1.0"
      }
    },
    {
      "reference": {
        "link": "https://localhost/mgmt/tm/sys/crypto?ver=12.1.0"
      }
    }
  ],
}
```

In addition to incorrect passwords, ensure that your remote authentication is correct. You should see entries in `/var/log/secure`.

Finally, the only supported role for the F5 Ansible modules is `Administrator`. No module is expected to work without this role being assigned to them.

## Discussion

Authentication can be a gnarly beast to debug because there are *so* many possible reasons it could not be working.

By far, the two most common reasons are

- People are not providing the right password
- Remote authentication is misconfigured on BIG-IP

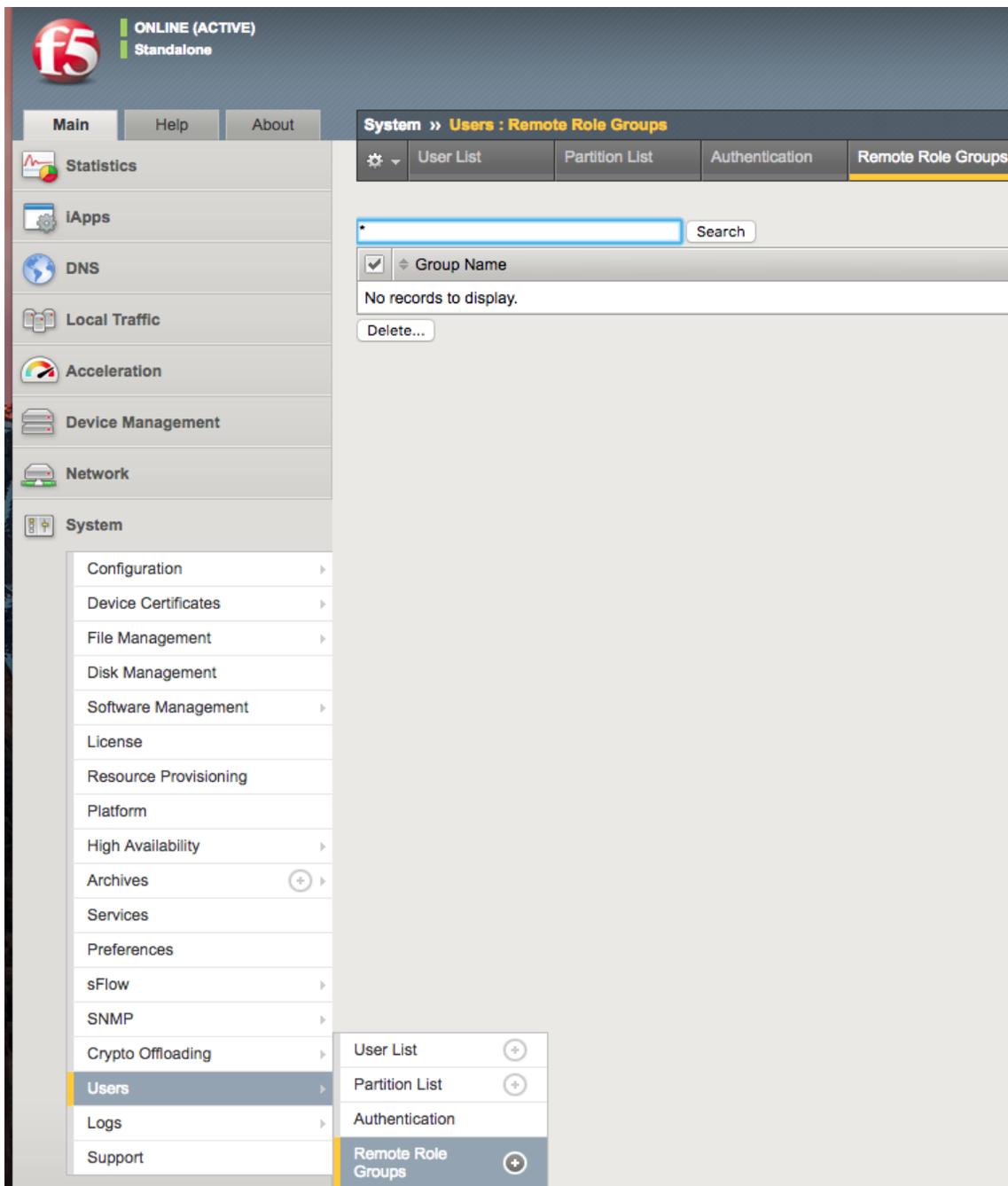
Understandably, people will often tell you that “of course my password is correct”. They will be wrong. 9 times out of 10, they will have either typed in the wrong password, or targeted the wrong BIG-IP (the BIG-IPs having different passwords).

If this is not the case, then confirm that their remote authentication is configured correctly. BIG-IP will log authentication successes to `/var/log/secure`. Therefore, if there are no entries in that file when a user runs an Ansible playbook with a remote auth user, that could be a problem.

The final thing that is often missed (with remote authentication in particular) is the assignment of BIG-IP roles to the remote-auth role. TACACS is notorious for this. Just because you have remote auth configured does not necessarily mean that all is well. You must also ensure that the remote users are associated with the local Administrator role.

Too often this is overlooked.

On later versions of BIG-IP, you can find the menu that needs to be configured in **System > Users > Remote Role Groups**. See the image below.



If these are not configured properly, then you'll be dead in the water. Figure your remote authentication out. This is almost never an Ansible problem. 99.999% of the time it is a user problem.

### 3.4.8 Dealing with unsupported versions

#### Problem

You're not sure what version of BIG-IP is supported

## Solution

The list of supported versions, at the time of this writing, is

- BIG-IP 12.0.0 (BIGIP-12.0.0.0.0.606)
- BIG-IP 12.1.0 (BIGIP-12.1.0.0.0.1434)
- BIG-IP 12.1.0-hf1 (BIGIP-12.1.0.1.0.1447-HF1)
- BIG-IP 12.1.0-hf2 (BIGIP-12.1.0.2.0.1468-HF2)
- BIG-IP 12.1.1 (BIGIP-12.1.1.0.0.184)
- BIG-IP 12.1.1-hf1 (BIGIP-12.1.1.1.0.196-HF1)
- BIG-IP 12.1.1-hf2 (BIGIP-12.1.1.2.0.204-HF2)
- BIG-IP 12.1.2 (BIGIP-12.1.2.0.0.249)
- BIG-IP 12.1.2-hf1 (BIGIP-12.1.2.1.0.264-HF1)
- BIG-IP 13.0.0 (BIGIP-13.0.0.0.0.1645)
- BIG-IP 13.0.0-hf1 (BIGIP-13.0.0.1.0.1668-HF1)
- BIG-IP 13.0.0-hf2 (BIGIP-13.0.0.2.0.1671-HF2)

If you are using an unsupported version, no F5 Ansible modules are expected to work except `bigip_command`. You must also use SSH to connect to the device (as REST will be unavailable on older platforms).

To use this, set the parameter `transport: cli` and authenticate as `root` for it to work.

We have not written a deprecation policy for EOL'ing supported versions (in Ansible) of F5 products.

## Discussion

At this time we have a large, and growing, list of F5 products that we have tested to work with Ansible. Eventually, this list will be pruned.

In **all** cases, our recommendation is to **plan your upgrade path**. No exceptions.

On legacy product (versions less than 12) we do not expect any of our modules to work, so do not even try. The **only** module that may work is the `bigip_command` module. However, on legacy versions, for it to work correctly,

- you **must** use the `transport: cli`
- you **must** set the `user` argument to a name that has the *Administrator* role.

If you do not do the above things, do not expect that any of your `tmsh` commands will work. An example usage would be

```
- name: Run multiple commands as root over CLI
  bigip_command:
    commands:
      - tmsh create ltm virtual foo
      - tmsh create ltm pool bar
    server: lb.mydomain.com
    password: secret
    user: root
    validate_certs: no
```

```
transport: cli  
delegate_to: localhost
```

WE MAKE APPS  FASTER.  
SMARTER.  
SAFER.

F5 Networks, Inc. | [f5.com](http://f5.com)

