Getting Started with Ansible

Ansible is a command-line automation tool that simplifies the large scale management of devices. It is one of the simplest tools that you can use to automate a large scale topology. There are only a few basics that you need to learn to use Ansible.

Helpful Links

- Ansible Best Practices
- Ansible Inventories
- Ansible Variables
- Built-in Modules

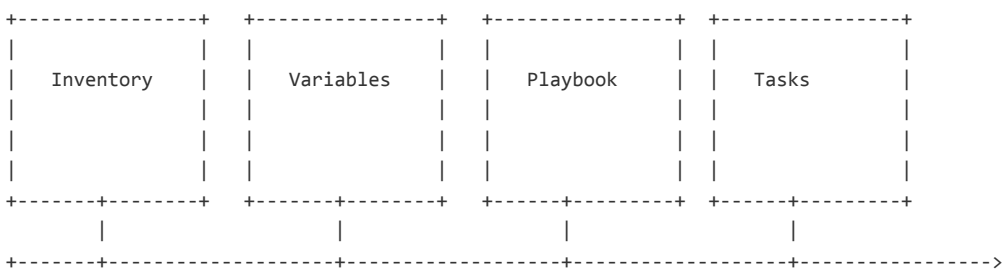
Benefits

- Tasks are run step-by-step easily identifying any issues during a deployment
- Can manage not only Junos devices but configure servers as well
- Extremely flexible ordering of tasks
- Simple to create playbooks with only YAML templates
- Easy to learn
- Easy to extend with custom modules
  - Python is first class language for this
  - But any language can be used to run scrips (Bash, Ruby, Perl)

Drawbacks

- Unable to manage Windows hosts
- Managing a large scale of devices requires a strong structure
  - SSH Keys at scale
  - Large scale variables
- Extremely flexible ordering of tasks
- Difficult to master

Execution Diagram



Ansible Technologies

Ansible at its core uses "Yet Another Markup Language" YAML as the syntax for building playbooks. YAML is a simplified language structure that has become quite popular for use due to its simplicity. In fact it is in use today in the PyEZ libraries for doing tables and views.

A playbook consists of a few required elements.

- Name
  - The name of the running playbook
- Hosts
  - Hosts to apply the tasks to
- Tasks
  - Tasks to apply to the hosts
- (Optionally) Variables
  - Variables allow for the customization of a running task

Playbook Example

```
---
- name: Configure basic firewall policies   #defines playbook
  hosts: mysrx                              #defines hosts to apply
  connection: local                         #defines execution environment, local is needed for
  gather_facts: no                          #gathers facts for the devices
  vars:                                     #variables to be used in the playbook
    junos_user: "root"
    junos_password: "Juniper"
    build_dir: "/tmp/"
    address_entries: [ {'name':'LocalNet','prefix':'172.16.0.0/24'},{'name':'PrivateNet','prefi
    fw_policy_info: [ {'policy_name':'Allow_Policy','src_zone':'trust','dst_zone':'untrust','sr

  tasks:                                    #set of tasks to run
    - name: Build address book entries      #Name of task
      template: src=templates/fw_address_book_global.set.j2 dest={{build_dir}}/fw_address_book_
      with_items: address_entries           #Add in additional variables to itterate over

    - name: Apply address book entries
      junos_install_config: host={{ inventory_hostname }} user={{ junos_user }} passwd={{ junos


    - name: Build firewall policies config template
      template: src=templates/fw_policy.set.j2 dest={{build_dir}}/fw_policy.set
      with_items: fw_policy_info

    - name: Apply firewall policies
      junos_install_config: host={{ inventory_hostname }} user={{ junos_user }} passwd={{ junos
```

Inventory

The inventory defines which hosts you can run Ansible against. This can consist of a simple text file or also utilize an API to gather this information. The format of file is in the traditional INI style format. The listing consists of a single host per line. You can also have groups of hosts that may have a common role. An example is if you had multiple web servers or database servers and you want to apply the same tasks to that group. You can also include ranges of alphanumeric characters as well.

Ansible Inventories

```
mail.example.com        #A single host
host[a:z].example.com   #26 different hosts defined by a range
```

```
172.16.0.1              #A host defined by an IP
172.16.0.[1:254]        #Hosts defined by an IP range

[webservers]        #A group of hosts
foo.example.com
bar.example.com

[dbservers]             #A second group of hosts
one.example.com
two.example.com
three.example.com
```

It is also possible to query the inventory from a script or API. There are existing tools that allow you to plug into things like AWS. With a simple API call to AWS it pulls in your entire inventory from the list of existing VMs.

Variables

Variables are the special sauce that makes Ansible so tasty to use. This allows you to take a playbook and customize it for your specific set of hosts you want to run against. So imagine you have two data centers. Each data center has a set of DNS, NTP, and syslog servers that are specific to the data center. In this case you can use the same playbook for both data centers, but specify different DNS, NTP, and syslog servers for each datacenter.

Variable Example

```
---
- name: Configure basic firewall policies
  hosts: mysrx
  connection: local
  gather_facts: no
  vars:                                 #variables to be used in the playbook
    junos_user: "root"                  #username for our Junos devices
    junos_password: "Juniper"           #password for our Junos devices
    build_dir: "/tmp/"                  #directory for us build templates in
                                        #a complex variable, this is just a python dictiona
    address_entries: [ {'name':'LocalNet','prefix':'172.16.0.0/24'},{'name':'PrivateNet','prefi
```

Variable scope

A variable can be applied to several locations within your Ansible environment. The most specific application of a variable becomes the value that is used when applied. Using the ordering capabilities of variables allows you to further customize how your tasks are run.

Variable Order

1. Host
2. Group
3. Role
4. Variable File
5. Playbook

Ansible Galaxy

Ansible not only includes a host of included modules, but we also have a repository that users can

contribute to for Ansible. This is called Ansible galaxy and it allows you to easily install 3rd party modules for use in your Ansible environment.

Example of installing Junos Ansible modules

```
[root@ansible-cm]# ansible-galaxy install Juniper.junos
downloading role 'junos', owned by Juniper
no version specified, installing 1.0.0
- downloading role from
https://github.com/Juniper/ansible-junos-stdlib/archive/1.0.0.tar.gz
- extracting Juniper.junos to /etc/ansible/roles/Juniper.junos
Juniper.junos was installed successfully
```

Ansible Tower

While all of this may seem great to use, how do you scale these scripts to a larger environment. For this Ansile has the tool Ansible Tower. It gives you a GUI that is wrapped around the management of Ansible tasks. This has a free trial version but it is not free to use. This can assist you in the management of a large scale environment.

Ansible Tower