ansible-eos Documentation

Release develop

Arista EOS+

June 02, 2016

Contents

The ansible-eos project provides modules for managing resources on Arista EOS nodes. Please see Ansible Galaxy for more details This project is maintained by the Arista Networks EOS+ Consulting Services organization.

Warning: Deprecation Notice
Ansible 2.1 ships with great new networking modules purpose-built for Arista EOS. Due to the easy-to-use nature of these modules, and their great flexibility, it is no longer recommended to use the arista.eos role.
Get started by checking out the Arista solution at Ansible.com

CHAPTER 1

Overview

1.1 Introduction

Ansible is a configuration management framework that provides an automated infrastructure for managing systems devices and applications. Ansible provides this functionality using an agent-less approach that focuses on management of the destination device and/or application over SSH. Ansible achieves its vision through the implementation of playbooks and modules. Playbooks, which are in turn comprised of a series of tasks to be executed on a host or group of hosts, provide the fundamental workflow in Ansible. Modules are host and/or application specific that perform the operations based on the directives of the tasks and playbooks. Complete details about Ansible can be found in their documentation.

1.2 Connection Types

Ansible provides three distinctly different connection types each providing a different method for connecting the Ansible runtime components (playbooks, modules) with the destination device. A summary of the connection types are below.

1.2.1 SSH Connection

When operating in this mode, Ansible will connect to the destination host using an encrypted SSH session. The SSH connection is established using either the hosts native SSH binary or using the Paramiko library. Since it uses SSH as the transport, the Ansible connection needs to be able to authenticate to the remote system and expects to operate in a Linux shell environment.

1.2.2 Local connections

When a host or task is operating with a local connection, tasks are executed from the Ansible host where the job was initiated. Local connections allow Ansible to make use of API transports and remove the need for establishing an SSH connection to the target device.

1.2.3 Accelerated mode

Ansible supported (since v0.8) a mode of operation known as Fireball mode. Fireball mode has since been depreciated in favor of accelerated mode (as of v1.3). Accelerated mode connects to the destination node and starts a daemon that is used for the remainder of the transaction.

Tip: More details about Accelerated Mode from Ansible's documentation.

In addition to the connection types discussed above, Ansible also supports a pull model. The pull model works in conjunction with SCM systems to perform its duties locally on the node. The pull model executes a local utility that retrieves the configuration data and proceeds to execute all of the activity locally on the node.

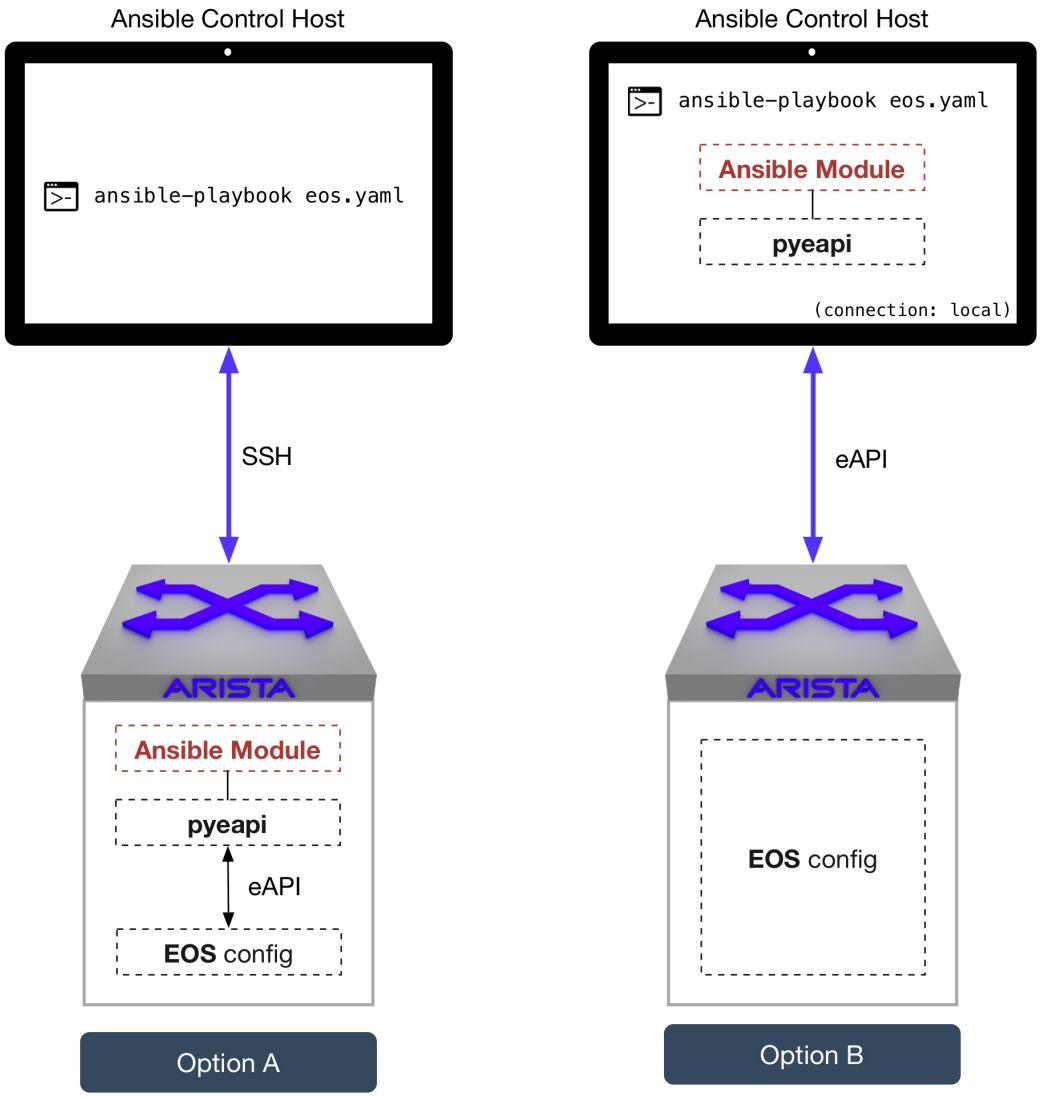
1.3 The Ansible EOS Role

1.3.1 Integration with the Python Client for eAPI

The Ansible Role for EOS builds on the Python Client for eAPI to provide automation of the management plane. Using eAPI as the underlying tranport, Ansible can be configured to interface with Arista EOS using either SSH based connections or HTTP based connections.

1.3.2 Topologies

Above, we discussed how Ansible is typically used to control a node. These principles remain true for Arista EOS nodes, however, there are some nuances that are important to understand when using the Ansible EOS role. Next, we will discuss the two main methods used to control an Arista EOS node using Ansible.



The illustration above demonstrates a typical scenario. You, as the user, want to execute an Ansible Playbook on one (or many) of your Arista nodes. From the user's perspective the interaction with the Ansible Control Host is the same, from your shell you would type:

```
ansible-playbook eos.yaml
```

Notice in the diagram a few things remain constant:

- pyeapi is always required (whether on the control host or EOS node), and
- pyeapi is ultimately responsible for making the eAPI calls to modify the node's configuration

While the overall flow is similar, the way in which the playbook is executed will differ between Option A and Option B. Let's discuss those differences below.

1.3.3 Option A

This method follows the traditional Ansible control procedure, namely:

1. Execute `ansible-playbook eos.yaml` from the Ansible Control Host

2. Collect Fact information from the node
3. Download the module to the node
4. Execute the module on the node
5. pyeapi commands run locally to modify configuration
6. Read stdout and parse it into JSON
7. Return the result to the Ansible Control Host

Assumption 1 You'll notice that this method uses SSH to communicate with the node. This implies that you have already included the Ansible Control Host's public SSH key in the nodes `authorized_keys` file, or you are providing a password when the playbook executes.

Assumption 2 Pyeapi is used by the Ansible EOS role modules to make configuration changes on the node. This implies that `pyeapi` is already installed on the node. The pyeapi module is NOT installed on Arista EOS nodes by default, so installation would be required by the user.

Understanding the Security Model

The Ansible EOS role provides a two stage authentication model to maximize the security and flexibility available for providing programatic access to EOS nodes. The steps above walk through how to enable both eAPI and create a shell account for use with Ansible. This section provides some additional details about how the two stage authentication model works.

Note: The two stage authentication model only applies to Option A.

Implementing a two stage security model allows operators to secure the Ansible shell account and prevent it from configuring EOS. Conversely, having a separate eAPI authentication mechanism allows operators to separately control the users that can run EOS modules without giving them root access to EOS.

When Ansible connects to an EOS node, it must first authenticate to Linux as it would for any other Linux platform. In order to create the shell account, the steps in *2. Preparing EOS for Ansible* should be followed. The steps above will create a user called 'ansible'. You are free to choose any username you like with the following exception: you cannot create a username the same as a local account in EOS (more on that in a moment).

By default, the EOS role assumes the user account is called 'ansible'. If the shell account is different, then the eos_username variable must be set in your playbook to the name of the shell account you intend to use. This ensures that the EOS node is bootstrapped properly for use with Ansible.

The second stage authentication model uses eAPI. eAPI provides its own authentication mechanism for securing what users can perform which actions in EOS. The eAPI user can be one that is authenticated by AAA; however, that is outside the scope of this discussion. The section *1. Enabling EOS Command API* provides an example of how to create a local user to use when authenticating with eAPI.

Note: The shell account and eAPI user must be different.

Ansible Host file and eapi.conf for Option A

It is important to understand that pyeapi is ultimately responsible for sending the configuration commands to your node. This means that at some point your adhoc command or playbook needs to indicate the credentials to create an eAPI connection. There are a few different ways to do this as explained below.

Method 1: Using Meta Arguments

Meta arguments are used to pass the exact eAPI connection parameters during adhoc command or play. If you provide all of the required eAPI connection information you will not even need to use eapi.conf. This is the preferred method since you do not need to create/maintain an eapi.conf file on the EOS node.

Tip: Read all about *Meta Arguments*

Example: In a playbook

eos-playbook.yml on Control Host

```
- name: eos nodes
  hosts: eos_switches

  tasks:
  - name: Configure EOS VLAN resources
    eos_vlan: vlanid=100
              name=mynewamazingvlan100
              username={{ username }}
              password={{ password }}
              transport={{ transport }}
```

/etc/ansible/hosts on Control Host

```
[eos_switches]
192.168.0.50
192.168.0.51
192.168.0.52
192.168.0.53

[eos_switches:vars]
ansible_ssh_user=ansible
# Used for eapi connection once SSH'd in
username=eapi
password=password
transport=https
```

/mnt/flash/eapi.conf on EOS node

```
# empty file. This file is not needed on the EOS device.
```

Explanation

This method utilizes the Ansible hosts file to feed information into the playbook. We group our nodes under the `eos_switches` group name to avoid duplication of variables, and use `[eos_switches:vars]` to create a set of variables that apply to all switches in the group. These variables are available in the playbook. We indicate in the play to execute our task against all nodes in this group. Then We use `{{ username }}`, `{{ password }}` etc. to substitute the eapi parameters into the play. Since all of the necessary eAPI information is present, the module does not need to consult an eapi.conf on the EOS node for connection parameters. In effect, the simplified connection flow looks like:

1. SSH to node in `[eos_switches]` (IPs 192.168.0.50-53)
2. Copy module(s) to EOS node.
3. Create eapi connection to `http://localhost:80/command-api`
4. Modify node configuration.

Method 2: Using eapi.conf

In this method we will put all of the eAPI connection info into `/mnt/flash/eapi.conf` on each EOS node that is being controlled. When we execute a play or adhoc command, pyeapi will not be passed connection information from Ansible, therefore it will consult eapi.conf on the EOS node to learn eapi connection information. As you can imagine this causes creates additional administrative overhead and is not the most efficient method (ie try to use Method 1).

Example

eos-playbook.yml on Control Host

```
- name: eos nodes
  hosts: eos_switches

  tasks:
  - name: Configure EOS VLAN resources
    eos_vlan: vlanid=100
              name=mynewamazingvlan100
              connection={{ connection }}
```

/etc/ansible/hosts on Control Host

```
[eos_switches]
spine-1
spine-2
tor-1
tor-2

[eos_switches:vars]
connection=localhost
ansible_ssh_user=ansible
```

/mnt/flash/eapi.conf on EOS node

```
[connection:localhost]
username: admin
password: password
transport: https
```

Explanation

Here we use the `connection` meta argument. This directly relates the connection name in eapi.conf. As you can see there is no eAPI connection information in /etc/ansible/hosts, rather we just have names of nodes. This changes the connection flow in the following way:

1. Control Host SSH into node listed in hosts file. EG ssh into spine-1 with user ansible
2. Copy modules to EOS node.
3. Execute module. The module is told to use `connection=localhost`.
4. Module looks for `localhost` in `/mnt/flash/eapi.conf`.
5. Learns which transport, username and password to use. Sets up eapi connection.
6. Executes commands to modify node configuration.

1.3.4 Option B

This method uses the `connection: local` feature within the `eos.yaml` playbook. This causes the transport method to be an eAPI connection (HTTP[S]) versus SSH. This changes how the playbook gets executed in the follow-

ing way:

1. Include `connection: local` in `eos.yaml`
2. Execute `ansible-playbook eos.yaml` from the Ansible Control Host
3. pyeapi consults the local ~/.eapi.conf file which provides node connection information
4. Collect Fact information from the node
5. Execute the module on the Ansible Control Host
6. pyeapi commands run over the network to modify configuration
7. Read stdout and parse it into JSON
8. Present the result on the Ansible Control Host

Assumption 1 Here, the connection between the Ansible Control Host and the Arista node is an eAPI connection. This implies that you have an `eapi.conf` file on your Ansible Control Host that contains the connection parameters for this node, or you pass the connection parameters as meta arguments. The caveat when using `eapi.conf` is that the password for the eAPI connection is stored as plaintext. See *Is there any intention to encrypt passwords we put into eapi.conf?* for more information.

Ansible Host file and eapi.conf for Option B

Regardless of the method you use to communicate with your node, one thing is constant: pyeapi is ultimately responsible for sending the configuration commands to your node. This means that at some point your adhoc command or playbook needs to indicate the credentials to create an eAPI connection. There are a few different ways to do this as explained below.

Method 1: Using Meta Arguments

Meta arguments are used to pass the exact eAPI connection parameters during adhoc command or play. If you provide all of the required eAPI connection information you will not even need to use eapi.conf. This is the most verbose and least flexible.

Tip: Read all about *Meta Arguments*

Example: In a playbook

eos-playbook.yml on Control Host

```
- name: eos nodes
  hosts: eos_switches
  connection: local

  tasks:
  - name: Configure EOS VLAN resources
    eos_vlan: vlanid=100
              name=mynewamazingvlan100
              host={{ inventory_hostname }}
              username={{ username }}
              password={{ password }}
              transport={{ transport }}
```

/etc/ansible/hosts on Control Host

```
[eos_switches]
192.168.0.50
192.168.0.51
192.168.0.52
192.168.0.53

[eos_switches:vars]
username=eapi
password=password
transport=https
```

~/.eapi.conf on Control Host

```
# empty file
```

Explanation

This method utilizes the Ansible hosts file to feed information into the playbook. The key to success here is grouping our nodes under the `eos_switches` group name. We then use `[eos_switches:vars]` to create a set of variables that apply to all switches in the group. These variables are available in the playbook. We indicate in the play to execute our task against all nodes in this group. Then we use `{{ inventory_hostname }}`, `{{ username }}`, etc. to substitute the host name (ip address in this case) and other connection parameters into the play. Since all of the necessary eAPI information is present, the module does not need to consult an eapi.conf file for connection parameters.

Method 2: Using eapi.conf

In this method we will put all of the eAPI connection info into eapi.conf. When we execute a play or adhoc command, pyeapi will not be passed connection information from Ansible, therefore it will consult eapi.conf to learn connection information.

Example

eos-playbook.yml on Control Host

```
- name: eos nodes
  hosts: eos_switches
  connection: local

  tasks:
  - name: Configure EOS VLAN resources
    eos_vlan: vlanid=100
              name=mynewamazingvlan100
              connection={{ inventory_hostname }}
```

/etc/ansible/hosts on Control Host

```
[eos_switches]
spine-1
spine-2
tor-1
tor-2
```

~/.eapi.conf on Control Host

```
[connection:spine-1]
host: 192.168.0.50
username: admin
```

```
password: password
transport: https

[connection:spine-2]
host: 192.168.0.51
username: admin
password: password
transport: https

[connection:tor-1]
host: 192.168.0.52
username: admin
password: password
transport: https

[connection:tor-2]
host: 192.168.0.53
username: admin
password: password
transport: https
```

Explanation

Here we use a new meta argument `connection`. This directly relates the connection name in eapi.conf. As you can see there is no eAPI connection information in /etc/ansible/hosts, rather we just have names of nodes. When the particular ansible-eos module executes it will reference `~/.eapi.conf` to determine how to connect to the EOS node over eAPI.

1.4 Ansible Tower

Ansible provides a product that implements a web based interface and REST API known as Tower. The web interface provides some additional capabilities to the base Ansible framework around role based access and programmatic interface to the Ansible environment.

CHAPTER 2

Quick Start

Contents

2.1 Introduction

This quick-start guide provides the fastest method to get up and running with the Ansible EOS role. It assumes that you already have an Ansible environment running with the Ansible EOS role. If not, see *Host (Control) System* and *Install the Ansible EOS Role* before following this guide. This guide assumes very little experience with Ansible, therefore, if the steps seem to leave you with questions and uncertainties please let us know so that we can improve it.

2.2 Getting Started

Before jumping in head first, it's important to understand how *The Ansible EOS Role* is deployed. At the preceding link, you'll see two deployment options which correlate to two separate quick start paths below. Have a quick read of *The Ansible EOS Role* and then come follow your preferred path. It's also recommended that you take a look at pyeapi documentation since it plays an essential part in the Ansible EOS role.

2.3 Option A: Connect to Arista Node over SSH

Tasklist

2.3.1 1. Enabling EOS Command API

The modules provided in the Arista EOS role require command API (aka eAPI) to be enabled on the switch. The modules use eAPI to communicate with EOS. Since eAPI is not enabled by default, it must be initially enabled before the EOS modules can be used.

The steps below provide the basic steps to enable eAPI. For more advanced configurations, please consult the EOS User Guide.

As you may have learned in *The Ansible EOS Role*, when you connect to your node over SSH, Ansible will copy the Python module code to the switch and then execute it locally using Pyeapi APIs. Therefore, you have a few options when it comes to which protocol is enabled for eAPI.

Transport	eapi.conf Required	Pyeapi run from	Authentication Required
http	Yes	On/Off-switch	Yes
https	Yes	On/Off-switch	Yes
http_local	Yes	On-switch only	No
socket	No	On-switch only	No

Note: http_local and socket are EOS transports only supported in EOS version 4.14.5+

Therefore, it is recommended to use `socket` if you are running a recent version of EOS. Otherwise, use HTTP or HTTPS depending upon your security model.

Step 1.1. Login to the destination node and enter configuration mode

```
switch> enable
switch# configure
switch(config)#
```

Step 1.2a. Enable eAPI for Unix Sockets and Disable HTTP/s

```
switch(config)# management api http-commands
switch(config-mgmt-api-http-cmds)# no shutdown
switch(config-mgmt-api-http-cmds)# protocol unix-socket
switch(config-mgmt-api-http-cmds)# no protocol https
```

Step 1.2a. Enable eAPI for HTTP Local

This will only expose port 8080 at the loopback (localhost)

```
switch(config)# management api http-commands
switch(config-mgmt-api-http-cmds)# no shutdown
switch(config-mgmt-api-http-cmds)# no protocol https
switch(config-mgmt-api-http-cmds)# protocol http localhost
```

Step 1.2a. Enable eAPI for Standard HTTP/S

```
switch(config)# management api http-commands
switch(config-mgmt-api-http-cmds)# no shutdown
```

The configuration above enables eAPI with the default settings. This enables eAPI to listen for connections on HTTPS port 443 by default.

Step 1.3. Create a local user The user created in this step is different than the shell account to be created in the Preparing EOS for Ansible section. Please see the section *Understanding the Security Model* for more details.

```
switch(config)# username eapi secret icanttellyou
```

The username (eapi) and password (icanttellyou) can be any valid string value.

2.3.2 2. Preparing EOS for Ansible

In order to successfully execute playbook tasks the EOS node must be configured to allow the Ansible control node to directly attach to the Linux shell. The following steps provide a walk through for setting up password-less access to EOS nodes for use with Ansible.

Note: These steps will create a user that has root privileges to your EOS node, so please handle credentials accordingly

---

Step 2.1. Login to the destination node and enter the Linux shell

```
veos> enable
veos# bash

Arista Networks EOS shell
```

Step 2.2. Create the user to use with Ansible, create the home directory and prepare for uploading your SSH key. In the below example we will create a user called ansible. The second command will create a temporary password for the user but we will be switching to using SSH keys and the password will be removed

```
# create the user 'ansible' with temporary password 'password'
[admin@veos ~]$ sudo useradd -d /persist/local/ansible -G eosadmin ansible
[admin@veos ~]$ echo password | sudo passwd --stdin ansible
Changing password for user ansible.
passwd: all authentication tokens updated successfully.

# prepare the home directory so we can upload an ssh key
[admin@veos ~]$ sudo mkdir /persist/local/ansible/.ssh
[admin@veos ~]$ sudo chmod 700 /persist/local/ansible/.ssh
[admin@veos ~]$ sudo chown ansible:eosadmin /persist/local/ansible/.ssh
[admin@veos ~]$ sudo ls -lah /persist/local/ansible

# exit the Linux shell and disconnect
[admin@veos01 ~]$ logout
veos#logout
Connection to veos01 closed.
```

Step 2.3. Upload the SSH key to use from your Ansible control host and verify access from remote host

```
ansible@hub:~$ scp ~/.ssh/id_rsa.pub ansible@veos01:.ssh/authorized_keys
Password:

ansible@hub:~$ ssh ansible@veos01

Arista Networks EOS shell

[ansible@veos ~]$
```

Step 2.4. Configure EOS to create user on reboot with no password assigned. This will only allow the Ansible user to login with keys.

```
[ansible@veos ~]$ vi /mnt/flash/rc.eos

#!/bin/sh
useradd -d /persist/local/ansible -G eosadmin ansible
```

Step 2.5. Reboot the EOS node and start automating with Ansible

```
[ansible@veos ~]$ sudo reboot
```

2.3.3 3. Install pyeapi

As mentioned earlier, the Ansible EOS role uses pyeapi on the Arista node that will be configured. Follow the pyeapi installation guide to install the package.

2.3.4 4. A Simple Playbook

If you are new to Ansible it might seem like a lot is going on, but this step will show you how easy it is to manage your Arista device. The power of Ansible lies in the Playbook. We will just skim the surface of what's possible in a playbook, but this should serve as a good launching point.

Step 4.1. Create an Ansible Inventory File

Each Ansible play references one or more nodes. You define these nodes in an Ansible `hosts` file.

Hint: Learn more about Ansible Inventory.

```
ansible@hub:~$ mkdir ~/myfirstplaybook
ansible@hub:~$ cd ~/myfirstplaybook
ansible@hub:~$ vi hosts
```

and add the connection info for your node substituting the IP or FQDN of your node as well as the name of the user created in Step 2.2 above:

```
[eos_switches]
<node>
# Add more entries here for additional devices you want to
# keep in the eos_switches group

[eos_switches:vars]
ansible_ssh_user=<user>
# Information from step 1.2. Used for eapi connection once Ansible SSHes in.
transport=https
username=eapi
password=icanttellyou
port=<port-if-non-default>
```

Note: If `socket` is enabled for eAPI, there is no need to add the variables: `transport`, `username`, `password`, `port`. If `http_local` is being used, simply use `transport=http_local`.

Example

```
[eos_switches]
veos01
veos02
veos03
veos04

[eos_switches:vars]
ansible_ssh_user=ansible
transport=https
username=eapi
password=icanttellyou
```

Step 4.2. Create playbook

Let's create Vlan150 using the *eos_vlan* module:

```
ansible@hub:~$ vi my-test-eos-playbook.yml
```

Then paste in the following

```
---
- hosts: eos_switches
  gather_facts: no

  roles:
    - arista.eos

  tasks:
    - name: configures vlan 150
      eos_vlan:
        vlanid=150
        name=newVlan150
        transport={{ transport }}
        username={{ username }}
        password={{ password }}
        debug=yes
      register: vlan_cfg_output

    - debug: var=vlan_cfg_output
```

Hint: Don't be confused by the presence of `transport`, `username` and `password`. They aren't used until after Ansible SSHes into the EOS node. By including these parameters here we remove the need to have an `eapi.conf` file on each EOS node.

Note: If your eAPI is configured to use Unix Socket there is no need to pass the `transport`, `username`, or `password` attributes since the default is to try and use `transport=socket`.

Step 4.3. Run playbook

Simply execute from your Ansible Host and review output:

```
ansible@hub:~$ ansible-playbook -i hosts my-test-eos-playbook.yml
```

Result:

You should see JSON output containing any changes, along with the current and desired state. So what really happened?

1. We execute the command and Ansible goes to our inventory to find the specified nodes in group `eos_switches`.
2. Ansible is told to connect via SSH with user `ansible` from `ansible_ssh_user=ansible`.
3. Ansible creates a temp directory in the `ansible` user's home directory
4. Ansible copies eos_vlan.py to the temp directory created above.
5. Ansible executes eos_vlan.py with the specified arguments
6. eos_vlan.py uses pyeapi to configure the new Vlan.
7. Ansible cleans up the temp folder and returns output to the control host.

You should notice that Ansible reports configuration has `changed`. If you ran this command again it should report no changes due to idempotency.

2.4 Option B: Connect to Arista Node over eAPI

Tasklist

- *1. Enabling EOS Command API*
- *2. Install pyeapi*
- *3. A Simple Playbook*

2.4.1 1. Enabling EOS Command API

The modules provided in the Arista EOS role require command API (aka eAPI) to be enabled on the switch. The modules use eAPI to communicate with EOS. Since eAPI is not enabled by default, it must be initially enabled before the EOS modules can be used.

The steps below provide the basic steps to enable eAPI. For more advanced configurations, please consult the EOS User Guide.

Step 1.1. Login to the destination node and enter configuration mode

```
switch> enable
switch# configure
switch(config)#
```

Step 1.2. Enable eAPI

```
switch(config)# management api http-commands
switch(config-mgmt-api-http-cmds)# no shutdown
```

The configuration above enables eAPI with the default settings. This enables eAPI to listen for connections on HTTPS port 443 by default.

Step 1.3. Create a local user The user created in this step is used by pyeapi to run configuration commands.

```
switch(config)# username eapi secret icanttellyou
```

The username (eapi) and password (icanttellyou) can be any string value. The values are then used in either eapi.conf or passed in through the module meta arguments to authenticate to eAPI.

2.4.2 2. Install pyeapi

As mentioned earlier, the Ansible EOS role uses pyeapi on the Arista node that will be configured. Follow the **'pyeapi http://pyeapi.readthedocs.org/en/latest/install.html>'_** installation guide to install the package.

Create local pyeapi.conf file

```
[ansible@hub ~]$ vi ~/.eapi.conf
```

with credentials you created in Step 1.3. The `connection:<NAME>` should match the entry in `hosts`, Step 3.1 below:

```
[connection:veos01]
host: <ip-or-fqdn>
transport: https
username: eapi
password: icanttellyou
port: <port-if-non-default>
```

2.4.3 3. A Simple Playbook

If you are new to Ansible it might seem like a lot is going on, but this step will show you how easy it is to manage your Arista device. The power of Ansible lies in the Playbook. We will just skim the surface of what's possible in a playbook, but this should serve as a good launching point.

Step 3.1. Create an Ansible Inventory File

Let's add the details of our test node to an Ansible Inventory file.

Hint: Learn more about Ansible Inventory.

```
ansible@hub:~$ mkdir ~/myfirstplaybook
ansible@hub:~$ cd ~/myfirstplaybook
ansible@hub:~$ vi hosts
```

and add the connection info for your node substituting the IP or FQDN of your node under our `eos_switches` group. This should match the `connection` parameter in your `~/.eapi.conf`:

```
[eos_switches]
<node>
```

Example

```
[eos_switches]
veos01
```

Step 4.2. Create playbook

Let's create Vlan150 using the *eos_vlan* module:

```
ansible@hub:~$ vi my-test-eos-playbook.yml
```

Then paste in the following

```
---
- hosts: eos_switches
  gather_facts: no
  connection: local
```

```
roles:
  - arista.eos

tasks:
  - name: Add Vlan 150 to my switches
    eos_vlan:
      vlanid=150
      name=newVlan150
      connection={{ inventory_hostname }}
      debug=yes
    register: vlan_cfg_output

  - debug: var=vlan_cfg_output
```

Step 4.3. Run playbook

Simply execute from your Ansible Host:

```
ansible@hub:~$ ansible-playbook -i hosts my-test-eos-playbook.yml
```

Result:

You should see JSON output containing any changes, along with the current and desired state. So what really happened?

1. We execute the command and Ansible goes to our inventory to find the specified nodes that match group `eos_switches`.
2. Ansible is told to use `connection:local` so no SSH connection will be established to the node.
3. Ansible substitutes the host name from `hosts` into the `{{ inventory_hostname }}` parameter. This creates the link to the `[connection:veos01]` in `~/.eapi.conf`.
4. Ansible creates a temp directory in the user's home directory, eg `$HOME/.ansible/tmp/`.
5. Ansible copies eos_vlan.py to the temp directory created above.
6. Ansible executes eos_vlan.py with the specified arguments
7. eos_vlan.py uses pyeapi to configure the Vlan.
8. pyeapi consults `~/.eapi.conf` to find connection named `veos01`
9. Ansible cleans up the temp folder and returns output to the control host.

You should notice that Ansible reports configuration has `changed`. If you ran this command again it should report no changes due to idempotency.

2.5 Now what?

This guide should have helped you install and configure all necessary dependencies and given you a basic idea of how to use the Ansible EOS role. Next, you can add to your Ansible playbooks using a combination of modules. You can also check out the list of modules provided within the Ansible EOS Role to see all of the ways to make configuration changes. There's also an examples directory which has a full-featured set of tasks and roles to build an entire leaf/spine network with MLAG and BGP.

Tip: Please send us some feedback on ways to improve this guide.

CHAPTER 3

Installation

The installation of Ansible is straightforward and simple. This section provides an overview of the installation of Ansible on a host system as well as how to configure an Arista EOS node to work with the Ansible framework.

Important: Ansible 1.9 or later is required.

3.1 Host (Control) System

Installing Ansible on a host (or control) system is a relatively simple process. Ansible supports all major Linux distributions running Python 2.6 or later as a control system. Ansible is integrated with package managers for each system type to ease the installation. Ansible can also be run directly from a Git checkout.

A quick reference summary of the various installation method is found below. For authoritative details regarding the installation of Ansible on a control system, see Ansible's installation documentation.

3.1.1 Installing via YUM

Ansible is provided via standard RPM installations from EPEL 6 and Fedora repositories. Simply run Yum with appropriate permissions to install the latest version of Ansible.

```
$ sudo yum install ansible
```

3.1.2 Installing via Apt (Ubuntu)

In order to install directly from Apt, the Ansible PPA will need to be added to Apt's sources. Ansible binaries are installed from this PPA. Once the PPA has been added to the Apt sources list execute the following commands to install Ansible.

```
sudo apt-get install software-properties-common
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible
```

3.1.3 Installing via PIP

Ansible can be installed using Python PIP. To install Ansible with PIP, simply enter the following command from a shell prompt.

```
sudo pip install ansible
```

3.2 Install the Ansible EOS Role

There are two methods that can be used to install the ansible-eos modules on your system; (1) Ansible Galaxy, (2) Github - from source. The first method is the easiest and makes using the modules a little easier, but the drawback is that you are dependent upon releases being posted to Ansible Galaxy. The second method is good if you plan on working with the actual module code from source or wish to closely follow all changes in development.

3.2.1 Install Using Ansible Galaxy

From your Ansible Control Host, type:

```
sudo ansible-galaxy install arista.eos
```

Tip: To upgrade the role via Galaxy use `sudo ansible-galaxy install --force arista.eos`

Then you can use the role in your play as:

```
#my-playbook.yml
---
- hosts: eos_switches
  gather_facts: no

  roles:
    - arista.eos

  tasks:
    - name: configure Vlan150
      eos_vlan:
        vlanid=150
```

3.2.2 Installing from GitHub (for active development)

To get started, download the latest Arista EOS modules from Github using the clone command. From a terminal on the Ansible control system issue the following command:

```
git clone https://github.com/arista-eosplus/ansible-eos.git
```

The command above will create a new directory called 'ansible-eos' and clone the entire repository. Currently, the ansible-eos folder contains the "develop" branch which provides the latest code. Since the "develop" branch is still a work in progress, it might be necessary to switch to a released version of the EOS modules. In order to switch to a specific release version, change directories to the ansible-eos directory and enter the following command.

```
git tag
git checkout tags/<tag name>
```

The first command above "git tag" provides a list of all available tags. Each release has a corresponding tag that denotes the released code. To switch to a specific release simply use the name of the tag in the second command as the <tag name>.

For instance, to use the v1.0.0 release, enter the command

```
git checkout tags/v1.0.0
```

At any point in time switching to a different release is as easy as changing to the ansible-eos directory and re-issuing the "git checkout" command.

You will need to make Ansible aware of this new role if you want to use the included modules in your plays. You have a few options:

Option 1: Create Symlink (preferred)

We will create a symlink in `/etc/ansible/roles/` to make Ansible aware of the `ansible-eos` role. Notice that the symlink name is `arista.eos`. This is because the Ansible Galaxy role is named `arista.eos`:

```
# create soft symlink
cd /etc/ansible/roles
sudo ln -s /path/to/where/your/git/clone/is/ansible-eos arista.eos
```

Then you can use the role in your play as:

```
#my-playbook.yml
---
- hosts: eos_switches
  gather_facts: no

  roles:
    - arista.eos

  tasks:
    - name: configure Vlan150
      eos_vlan:
        vlanid=150
```

Option 2: Edit ansible.cfg roles_path

Here, you can edit `/etc/ansible/ansible.cfg` to make Ansible look for the `ansible-eos` directory:

```
# open the config file in an editor
sudo vi /etc/ansible/ansible.cfg

# if roles_path exists add a colon and the new path
# if the variable doesn't exist, create it under [defaults] section
[defaults]
roles_path=/path/to/where/your/git/clone/is/ansible-eos
```

Then you can use the role in your play as:

```
#my-playbook.yml
---
- hosts: eos_switches
  gather_facts: no

  roles:
    - ansible-eos

  tasks:
    - name: configures the hostname on tor1
```

```
    eos_vlan:
      vlanid=150
```

CHAPTER 4

Modules

4.1 All Modules

4.1.1 eos_acl_entry

Synopsis

Added in version 1.1.0

This module will manage standard ACL entries on EOS nodes

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.2 or later

Examples

```
- eos_acl_entry: seqno=10 name=foo action=permit srcaddr=0.0.0.0
  srcprefixlen=32

- eos_acl_entry: seqno=20 name=foo action=deny srcaddr=172.16.10.0
  srcprefixlen=16
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.2 eos_bgp_config

Synopsis

Added in version 1.1.0

The eos_bgp_config module provides resource management of the global BGP routing process for Arista EOS nodes

Options

Important: Requires Arista EOS 4.13.7M or later with command API enable

Important: Requires Python Client for eAPI 0.4.0 or later

Examples

```
- name: enable BGP routing with AS 65535
  eos_bgp_config: bgp_as=65535 state=present enable=yes

- name: disable the BGP routing process
  eos_bgp_config: bgp_as=65535 enable=no

- name: configure the BGP router-id
  eos_bgp_config: bgp_as=65535 router_id=1.1.1.1

- name: configure the BGP with just max paths
  eos_bgp_config: bgp_as=65535 router_id=1.1.1.1 maximum_paths=20

- name: configure the BGP with maximum_paths and maximum_ecmp_paths
  eos_bgp_config: bgp_as=65535 router_id=1.1.1.1 maximum_paths=20
  maximum_ecmp_paths=20
```

Note: All configuraiton is idempontent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports tateful resource configuration

4.1.3 eos_bgp_neighbor

Synopsis

Added in version 1.1.0

This eos_bgp_neighbor module provides stateful management of the neighbor statements for the BGP routing process for Arista EOS nodes

Options

Important: Requires Arista EOS 4.13.7M or later with command API enable

Important: Requires Python Client for eAPI 0.3.1 or later

Examples

```
- name: add neighbor 172.16.10.1 to BGP
  eos_bgp_neighbor: name=172.16.10.1 enable=yes remote_as=65000

- name: remove neighbor 172.16.10.1 to BGP
  eos_bgp_neighbor name=172.16.10.1 enable=yes remote_as=65000 state=absent
```

Note: All configuraiton is idempontent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports tateful resource configuration

4.1.4 eos_bgp_network

Synopsis

Added in version 1.1.0

This eos_bgp_network module provides stateful management of the network statements for the BGP routing process for Arista EOS nodes

Options

Important: Requires Arista EOS 4.13.7M or later with command API enable

Important: Requires Python Client for eAPI 0.3.1 or later

Examples

```
- name: add network 172.16.10.0/26 with route-map test
  eos_bgp_network: prefix=172.16.10.0 masklen=26 route_map=test

- name: remove network 172.16.0.0/8
  eos_bgp_network: prefix=172.16.0.0 masklen=8 state=absent
```

Note: All configuraiton is idempontent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports tateful resource configuration

4.1.5 eos_command

Synopsis

Added in version 1.0.0

The eos_command module provides a module for sending arbitray commands to the EOS node and returns the ouput. Only priviledged mode (enable) commands can be sent.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: execute show version and show hostname
  eos_command: commands='show version, show hostname'
```

Note: This module does not support idempotent operations.

Note: Supports eos metaparameters for using the eAPI transport

Note: This module does not support stateful configuration

4.1.6 eos_config

Synopsis

Added in version 1.0.0

The eos_config module evalues the current configuration for specific commands. If the commands are either present or absent (depending on the function argument, the eos_config module will configure the node using the command argument.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: idempotent operation for removing a SVI
  eos_config:
    command='no interface Vlan100'
    regexp='interface Vlan100'
    state=absent

- name: non-idempotent operation for removing a SVI
  eos_config:
    command='no interface Vlan100'

- name: ensure default route is present
  eos_config:
    command='ip route 0.0.0.0/0 192.168.1.254'

- name: configure interface range to be shutdown if it isn't already
  eos_config:
    command='shutdown'
    regexp='(?<=[^no ] )shutdown'
    section='interface {{ item }}'
  with_items:
    - Ethernet1
    - Ethernet2
    - Ethernet3
```

Note: This module does not support idempotent operations.

Note: Supports eos metaparameters for using the eAPI transport

Note: This module does not support stateful configuration

4.1.7 eos_ethernet

Synopsis

Added in version 1.0.0

The eos_ethernet module manages the interface configuration for physical Ethernet interfaces on EOS nodes.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: Ensure that Ethernet1/1 is administratively enabled
  eos_ethernet: name=Ethernet1/1 enable=yes

- name: Enable flowcontrol send and receive on Ethernet10
  eos_ethernet: name=Ethernet10 flowcontrol_send=yes flowcontrol_receive=yes
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Does not support stateful resource configuration.

4.1.8 eos_facts

Synopsis

Added in version 1.0.0

The eos_facts module collects facts from the EOS for use in Ansible playbooks. It can be used independently as well to discover what facts are availble from the node. This facts module does not cache any facts. If no configuration options are specified, then all facts are returned.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: collect all facts from node
  eos_facts:

- name: include only a filtered set of facts returned
  eos_facts: include=interfaces

- name: exclude a specific set of facts
  eos_facts: exclude=vlans
```

Note: Supports eos metaparameters for using the eAPI transport

Note: The include and exclude options are mutually exclusive

4.1.9 eos_interface

Synopsis

Added in version 1.0.0

The eos_interface module manages the interface configuration for any valid interface on EOS nodes.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: ensures the interface is configured
  eos_interface: name=Loopback0 state=present enable=yes

- name: ensures the interface is not configured
  eos_interface: name=Loopback1 state=absent
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration. This method also supports the 'default' state. This will default the specified interface. Note however that the default state operation is NOT idempotent.

4.1.10 eos_ipinterface

Synopsis

Added in version 1.0.0

The eos_ipinterface module manages logical layer 3 interface configurations.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: Ensure a logical IP interface is configured on Vlan100
  eos_ipinterface: name=Vlan100 state=present address=172.16.10.1/24

- name: Ensure a logical IP interface is not configured on Ethernet1
  eos_ipinterface: name=Ethernet1 state=absent
```

```
- name: Configure the MTU value on Port-Channel10
  eos_ipinterface: name=Port-Channel10 mtu=9000
```

Note: Currently this module only supports IPv4

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.11 eos_mlag_config

Synopsis

Added in version 1.0.0

The eos_mlag_interface module manages the MLAG interfaces on Arista EOS nodes. This module is fully stateful and all configuration of resources is idempotent unless otherwise specified.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: Ensure the MLAG domain-id is mlagPeer
  eos_mlag_config: domain_id=mlagPeer

- name: Configure the peer address and local interface
  eos_mlag_config: peer_address=2.2.2.2 local_interface=Vlan4094
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Does not support stateful resource configuration.

4.1.12 eos_mlag_interface

Synopsis

Added in version 1.0.0

The eos_mlag_interface module manages the MLAG interfaces on Arista EOS nodes. This module is fully stateful and all configuration of resources is idempotent unless otherwise specified.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: Ensure Ethernet1 is configured with mlag id 10
  eos_mlag_interface: name=Ethernet1 state=present mlag_id=10

- name: Ensure Ethernet10 is not configured as mlag
  eos_mlag_interface: name=Ethernet10 state=absent
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.13 eos_ping

Synopsis

Added in version 1.0.0

The eos_ping module will execute a network ping from the node and return the results. If the destination can be successfully pinged, then the module returns successfully. If any of the sent pings are not returned the module fails. By default, the error threshold is set to the same value as the number of pings sent

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.4.0 or later

Examples

```
- eos_ping: dst=192.168.1.254 count=10

# Set the error_threshold to 50% packet loss
- eos_ping: dst=192.168.1.254 count=10 error_threshold=50
```

Note: Important fixes to this module were made in pyeapi 0.4.0. Be sure to update to at least that version.

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Does not support stateful resource configuration.

4.1.14 eos_portchannel

Synopsis

Added in version 1.0.0

The eos_portchannel module manages the interface configuration for logical Port-Channel interfaces on EOS nodes.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: Ensure Port-Channel1 has members Ethernet1 and 2
  eos_portchannel: name=Port-Channel1 members=Ethernet1,Ethernet2

- name: Ensure Port-Channel10 uses lacp mode active
  eos_portchannel: name=Port-Channel10 members=Ethernet1,Ethernet3
                   lacp_mode=active
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.15 eos_purge

Synopsis

Added in version 1.0.0

The eos_purge module will scan the current nodes running-configuration and purge resources of a specified type if the resource is not explicitly configured in the playbook. This module will allow a playbook task to dynamically determine which resources should be removed from the nodes running-configuration based on the playbook. Note Purge is not supported for all EOS modules

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
# configure the set of vlans for the node

- name: configure vlans
  eos_vlan: vlanid={{ item }}
  with_items: ['1', '10', '11', '12', '13', '14', '15']
  register: required_vlans


# note the value for results is the registered vlan variable.  Also of
# importance is the to_nice_json filter which is required

- name: purge vlans not on the list
  eos_purge: resource=eos_vlan results='{{ required_vlans|to_nice_json }}'
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Does not support stateful resource configuration.

4.1.16 eos_routemap

Synopsis

Added in version 1.2.0

This module will manage routemap entries on EOS nodes

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.4.0 or later

Examples

```
- eos_routemap: name=rm1 action=permit seqno=10
                description='this is a great routemap'
                match='as 50,interface Ethernet2'
                set='tag 100,weight 1000'
                continue=20
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.17 eos_staticroute

Synopsis

Added in version 1.2.0

The eos_staticroute module manages static route configuration options on Arista EOS nodes.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.4.0 or later

Examples

```
- eos_staticroute: ip_dest=1.1.1.0/24 next_hop=Ethernet1
                   next_hop_ip=1.1.1.1 distance=1
                   tag=15 name=route1
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.18 eos_stp_interface

Synopsis

Added in version 1.0.0

Provides active state management of STP interface configuration on Arista EOS nodes.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: Ensure portfast is enabled on Ethernet3
  eos_stp_interface: name=Ethernet3 portfast=yes

- name: Ensure bpduguard is enabled on Ethernet49
  eos_stp_interface: name=Ethernet49 bpduguard=yes
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Does not support stateful resource configuration.

4.1.19 eos_switchport

Synopsis

Added in version 1.0.0

Provides active state management of switchport (layer 2) interface configuration in Arista EOS. Logical switchports are mutually exclusive with `eos_ipinterface`.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: Ensure Ethernet1 is an access port
  eos_switchport: name=Ethernet1 mode=access access_vlan=10

- name: Ensure Ethernet12 is a trunk port
  eos_switchport: name=Ethernet12 mode=trunk trunk_native_vlan=100

- name: Add the set of allowed vlans to Ethernet2/1
```

```
  eos_switchport: name=Ethernet2/1 mode=trunk trunk_allowed_vlans=1,10,100

- name: Add trunk group values to an interface
  eos_switchport: name=Ethernet5 trunk_groups=foo,bar,baz
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.20 eos_system

Synopsis

Added in version 1.0.0

The eos_system module manages global system configuration options on Arista EOS nodes.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: configures the hostname to spine01
  eos_system: hostname=spine01
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.21 eos_user

Synopsis

Added in version 1.2.0

The eos_user module helps manage CLI users on your Arista nodes. You can create, delete and modify users along with their passwords.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.4.0 or later

Important: Requires Cleartext passwords are not accepted in playbooks

Examples

```
- name: Create simple user with no assigned password
  eos_user: name=simpletom nopassword=true

- name: Create user with MD5 password
  eos_user: name=securetom encryption=md5
            secret=$1$J0auuPhz$Pkr5NnHssW.Jqlk17Ylpk0

- name: Create user with SHA512 password (passwd truncated in eg)
  eos_user: name=securetom encryption=sha512
            secret=$6$somesalt$rkDq7Az4Efjo

- name: Remove user
  eos_user: name=securetom state=absent

- name: Create user with privilege level 10
  eos_user: name=securetom encryption=sha512
            secret=$6$somesalt$rkDq7Az4Efjo
            privilege=10
```

```
- name: Create user with role network-admin
  eos_user: name=securetom encryption=sha512
            secret=$6$somesalt$rkDq7Az4Efjo
            privilege=10 role=network-admin

- name: Add an SSH key with a user no password
  eos_user: name=sshkeytom nopassword=true
            sshkey='ssh-rsa somesshkey'

- name: Remove SSH key with a user no password
  eos_user: name=sshkeytom nopassword=true
            sshkey=''
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.22 eos_varp

Synopsis

Added in version 1.2.0

This module will manage global Varp configuration on EOS nodes

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.4.0 or later

Examples

```
- eos_varp: mac_address='00:11:22:33:44:55'
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Does not support stateful resource configuration.

4.1.23 eos_varp_interface

Synopsis

Added in version 1.2.0

This module will manage interface Varp configuration on EOS nodes. Typically this includes Vlan interfaces only by using the ip virtual-router address command.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.4.0 or later

Examples

```
- eos_varp_interface: name=Vlan1000 shared_ip='1.1.1.2,1.1.1.3,1.1.1.4'
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Does not support stateful resource configuration.

4.1.24 eos_vlan

Synopsis

Added in version 1.0.0

The eos_vlan module manages VLAN configurations on Arista EOS nodes.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: ensures vlan 100 is configured
  eos_vlan: vlanid=100 state=present

- name: ensures vlan 200 is not configured
  eos_vlan: vlanid=200 state=absent

- name: configures the vlan name
  eos_vlan: vlanid=1 name=TEST_VLAN_1

- name: configure trunk groups for vlan 10
  eos_vlan: vlanid=10 trunk_groups=tg1,tg2,tg3
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.25 eos_vrrp

Synopsis

Added in version 1.2.0

This module will manage VRRP configurations on EOS nodes

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.4.0 or later

Examples

```
# Configure the set of tracked objects for the VRRP
# Create a list of dictionaries, where name is the object to be
# tracked, action is shutdown or decrement, and amount is the
# decrement amount. Amount is not specified when action is shutdown.

vars:
  tracks:
      - name: Ethernet1
        action: shutdown
      - name: Ethernet2
        action: decrement
        amount: 5

# Setup the VRRP

  - eos_vrrp:
      interface=Vlan70
      vrid=10
      enable=True
      primary_ip=10.10.10.1
      priority=50
      description='vrrp 10 on Vlan70'
      ip_version=2
      secondary_ip=['10.10.10.70','10.10.10.80']
      timers_advertise=15
      preempt=True
      preempt_delay_min=30
      preempt_delay_reload=30
      delay_reload=30
      track="{{ tracks }}"
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.26 eos_vxlan

Synopsis

Added in version 1.0.0

The eos_vxlan module manages the logical VxLAN interface configuration on Arista EOS nodes.

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: ensures the vxlan interface is configured
  eos_vxlan: name=Vxlan1 state=present enable=yes

- name: ensures the vxlan interface is not configured
  eos_vxlan: name=Vxlan1 state=absent

- name: configures the vxlan source interface
  eos_vxlan: name=Vxlan1 source_interface=Loopback0
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.27 eos_vxlan_vlan

Synopsis

Added in version 1.0.0

The eos_vxlan_vlan module manages the Vxlan VLAN to VNI mappings for an Arista EOS node that is operating as a VTEP

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: create a new vlan to vni mapping
  eos_vxlan_vlan: name=Vxlan1 state=present vlan=100 vni=1000

- name: remove an existing mapping if present in the config
  eos_vxlan_vlan: name=Vxlan1 state=absent vlan=200
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.1.28 eos_vxlan_vtep

Synopsis

Added in version 1.0.0

The eos_vxlan_vtep module manages the Vxlan global VTEP flood list configure on Arista EOS nodes that are operating as VTEPs

Options

Important: Requires Arista EOS 4.13.7M or later with command API enabled

Important: Requires Python Client for eAPI 0.3.0 or later

Examples

```
- name: Ensures that 1.1.1.1 is in the global flood list
  eos_vxlan_vtep: name=Vxlan1 state=present vtep=1.1.1.1

- name: Ensures that 2.2.2.2 is not in the global flood list
  eos_vxlan_vtep: name=Vxlan1 state=absent vtep=2.2.2.2
```

Note: All configuration is idempotent unless otherwise specified

Note: Supports eos metaparameters for using the eAPI transport

Note: Supports stateful resource configuration.

4.2 BGP Modules

4.3 Bridging Modules

4.4 IP Modules

4.5 Interfaces Modules

4.6 MLAG Modules

4.7 Route Policy Modules

4.8 STP Modules

4.9 System Modules

4.10 VARP Modules

4.11 VRRP Modules

4.12 VXLAN Modules

CHAPTER 5

Meta Arguments

Most EOS modules support additional arguments (meta arguments) in addition to the arguments available for configuring the resource. The meta arguments provide additional connection and troubleshooting arguments for executing tasks in Ansible.

Not all modules support all meta arguments. Please review the individual module documentation for applicability.

5.1 Troubleshooting Arguments

This section provides an overview of the arguments available for troubleshooting tasks with EOS modules.

- debug (booleans) - Enables additional output from the module
- logging (booleans) - Enables or disables logging details to syslog

5.2 Connection Arguments

The connection arguments provide a set of arguments that override the values from eapi.conf or eliminate the need for eapi.conf all together.

- config (string) - overrides the default path to the eapi.conf file
- username (string) - specifies the eAPI username used to authenticate
- password (string) - specifies the eAPI password used to authenticate
- host (string) - specifies the host address or FQDN for the connection
- port (string or integer) - specifies the port to use when connecting
- connection (string) - specifies the name of the connection profile to use
- transport (string) - configures the transport to use. Valid transport options include “http”, “https”, “socket”, “http_local”.

5.3 State Arguments

The state arguments provide state configuration for modules that are identified as stateful.

- state (string) - configures the resource state. Valid values include “present”, “absent”. Note that some modules can additional states

CHAPTER 6

Support

6.1 Contact

The Ansible EOS role is developed by Arista EOS+ CS and supported by the Arista EOS+ community. Support for the modules as well as using Ansible with Arista EOS nodes is provided on a best effort basis by the Arista EOS+ CS team and the community. You can contact the team that develops these modules by sending an email to ansible-dev@arista.com.

For customers that are looking for a premium level of support, please contact your local account team or email eosplus@arista.com for help.

6.2 Submitting Issues

The Arista EOS+ CS development team uses Github Issues to track discovered bugs and enhancement request to the Ansible EOS role. The issues tracker can be found at https://github.com/arista-eosplus/ansible-eos/issues.

For defect issues, please provide as much relevant data as possible as to what is causing the issue, if and how it is reproducible, the version of EOS and Ansible running.

For enhancement requests, please provide a brief description of the enhancement request and the version of EOS to be supported.

The issue tracker is monitored by Arista EOS+ CS and issues submitted are categorized and scheduled for inclusion in upcoming Ansible EOS role versions.

6.3 Debugging Module Output

All Ansible EOS role modules provide a consistent output and options for troubleshooting the module operations. Each module provides logging and debug information to help debugging the change the module is introducing. Modules provide two arguments for debugging: logging (default=on) and debug (default=off).

When a module executes, the module output can be registered as a variable and then used to display the output. Below is an example task that configures a logical Vxlan interface:

```
- name: Configure Vxlan logical interface
  eos_vxlan: name={{ vxlan.name }}
         description={{ vxlan.description|default(omit) }}
         source_interface={{ vxlan.source_interface }}
         multicast_group={{ vxlan.multicast_group }}
```

```
        debug=no
        connection={{ inventory_hostname }}
  when: vxlan is defined
  register: eos_vxlan_output
```

Once the variable is registered, for instance eos_vxlan_output in the above example, the Ansible debug module can be used to display the output.:

```
- name: Configure Vxlan logical interface
  eos_vxlan: name={{ vxlan.name }}
        description={{ vxlan.description|default(omit) }}
        source_interface={{ vxlan.source_interface }}
        multicast_group={{ vxlan.multicast_group }}
        debug=no
        connection={{ inventory_hostname }}
  when: vxlan is defined
  register: eos_vxlan_output

- debug: var=eos_vxlan_output
```

When the debug module is added to the playbook, the eos_vxlan module will display the following output.:

```
TASK: [debug var=eos_vxlan_output] ******************************************
ok: [veos02] => {
    "var": {
        "eos_vxlan_output": {
            "changed": false,
            "changes": {},
            "instance": {
                "description": null,
                "enable": true,
                "multicast_group": "239.10.10.10",
                "name": "Vxlan1",
                "source_interface": "Loopback0",
                "state": "present",
                "udp_port": 4789
            },
            "invocation": {
                "module_args": "name=Vxlan1 source_interface=Loopback0 multicast_group=239.10.10.10 c
                "module_name": "eos_vxlan"
            }
        }
    }
}
```

In the module output are the standard responses from Ansible task runs including invocation and changed. Invocation shows the name of the module that was executed and the arguments passed to to module which should match the task in the playbook.

The changed key displays true if any changes are made to the system or false if no changes are required on the end system.

The Ansible EOS role adds the keys for changes and instance. The instance key provides a view of the resource at the conclusion of the task execution. When compared to the nodes running-configuration, the instance should be displaying configuration values that are consistent with the nodes current configuration.

The changes key provides the set of key / value pairs that are changed during a module execution. Since the changed key has a value of false, no changes where made in this instance. The example below shows the output when changes are made to the configuration.:

```
TASK: [debug var=eos_vxlan_output] ******************************************
ok: [veos02] => {
    "var": {
        "eos_vxlan_output": {
            "changed": true,
            "changes": {
                "multicast_group": "239.10.10.10",
                "source_interface": "Loopback0"
            },
            "instance": {
                "description": null,
                "enable": true,
                "multicast_group": "239.10.10.10",
                "name": "Vxlan1",
                "source_interface": "Loopback0",
                "state": "present",
                "udp_port": 4789
            },
            "invocation": {
                "module_args": "name=Vxlan1 source_interface=Loopback0 multicast_group=239.10.10.10 c
                "module_name": "eos_vxlan"
            }
        }
    }
}
```

The above example show the output from the same module; however, this time changes are introduced as indicated by the changed key being set to true. In addition, the changes key shows which arguments where changed and the value the keys were set to. For all other arguments that are not included in the changes key, no configuration updates were executed.

Thus far, the examples have shown the ouput for eos_* modules that is available for every run without any changes. All modules also provide a `debug` argument that, when enabled, provides additional information about the execution of the module.

Below is an example of the same module execution, only this time with debug enabled:

```
TASK: [debug var=eos_vxlan_output] ******************************************
ok: [veos02] => {
    "var": {
        "eos_vxlan_output": {
            "changed": true,
            "changes": {
                "multicast_group": "239.10.10.10",
                "source_interface": "Loopback0"
            },
            "debug": {
                "current_state": {
                    "description": null,
                    "enable": true,
                    "multicast_group": "",
                    "name": "Vxlan1",
                    "source_interface": "",
                    "state": "present",
                    "udp_port": 4789
                },
                "desired_state": {
                    "description": null,
                    "enable": true,
```

```
                    "multicast_group": "239.10.10.10",
                    "name": "Vxlan1",
                    "source_interface": "Loopback0",
                    "state": "present",
                    "udp_port": null
                },
                "node": "Node(connection=EapiConnection(transport=https://192.168.1.17:443//command-a
                "params": {
                    "config": null,
                    "connection": "veos02",
                    "debug": true,
                    "description": null,
                    "enable": true,
                    "logging": true,
                    "multicast_group": "239.10.10.10",
                    "name": "Vxlan1",
                    "password": null,
                    "source_interface": "Loopback0",
                    "state": "present",
                    "udp_port": null,
                    "username": null
                },
                "pyeapi_version": "0.2.2",
                "stateful": true
            },
            "instance": {
                "description": null,
                "enable": true,
                "multicast_group": "239.10.10.10",
                "name": "Vxlan1",
                "source_interface": "Loopback0",
                "state": "present",
                "udp_port": 4789
            },
            "invocation": {
                "module_args": "name=Vxlan1 source_interface=Loopback0 multicast_group=239.10.10.10 c
                "module_name": "eos_vxlan"
            }
        }
    }
}
```


With the `debug` key set to `yes` the the module output provides an additional keyword `debug` that provides additional information. While the keys under `debug` could vary from module to module, the following keys are in common across all module implementations

- current_state - shows the resource instance values at the beginning of the task run before any changes are attempted
- desired_state - shows the desired state of the resource based on the input arguments from the task
- node - shows the eAPI connection information
- params - shows all parameters used to build the module including arguments and metaparameters
- pyeapi_version - shows the current version of pyeapi library used
- stateful - shows whether or not the module is stateful

Using the `debug` argument provides a fair amount of detail about how the module executes on the node. There is also logging information that also provides some details about the changes the module is making to the end system.

Logging is enabled by default and can be disabled by configuring the *logging* keyword argument to `false`.

All logging information is sent to the local syslog on the device executing the module. When using the SSH transport, all logging information will be found in the node's syslog and in the case of using the eAPI transport, the logging information will be found on the Ansible control hosts syslog.

From the same example as above, the `eos_vxlan` module provides logging information in syslog as shown below:

```
Apr 16 00:36:34 veos02 ansible-eos_vxlan: Invoked with username=None enable=True logging=True name=Vx
Apr 16 00:36:34 veos02 ansible-eos: DEBUG flag is True
Apr 16 00:36:34 veos02 ansible-eos: Connected to node Node(connection=EapiConnection(transport=https
Apr 16 00:36:34 veos02 ansible-eos: called instance: {'multicast_group': '', 'state': 'present', 'ena
Apr 16 00:36:34 veos02 ansible-eos: Invoked set_source_interface for eos_vxlan[Vxlan1] with value Loo
Apr 16 00:36:34 veos02 ansible-eos: Invoked set_multicast_group for eos_vxlan[Vxlan1] with value 239.
Apr 16 00:36:35 veos02 ansible-eos: called instance: {'multicast_group': '239.10.10.10', 'state': 'pr
Apr 16 00:36:35 veos02 ansible-eos: Module completed successfully
```

The log output displays the invocation of the module by Ansible and includes information about the execution process.

Using both the `debug` and `logging` keywords provides a window into the execution of the Ansible EOS role and should make troubleshooting undesired results easier.

6.4 Debugging EOS Connectivity Issues

Sometimes it is difficult to quickly deduce what is causing a particular playbook or task not to run without error. While Ansible provides some verbose details during the task execution, sometimes the problem relates to connecting from the Ansible control host to the EOS node.

This section provides some basic tips on troubleshooting connectivity issues with Arista EOS nodes.

When starting to troubleshoot connectivity errors, the first place to start is with some simple `ping` tests to ensure there is connectivity between the Ansible control host and the EOS node.:

```
$ ping -c 5 192.168.1.16
PING 192.168.1.16 (192.168.1.16): 56 data bytes
64 bytes from 192.168.1.16: icmp_seq=0 ttl=64 time=1.202 ms
64 bytes from 192.168.1.16: icmp_seq=1 ttl=64 time=1.082 ms
64 bytes from 192.168.1.16: icmp_seq=2 ttl=64 time=0.829 ms
64 bytes from 192.168.1.16: icmp_seq=3 ttl=64 time=0.936 ms
64 bytes from 192.168.1.16: icmp_seq=4 ttl=64 time=1.021 ms
--- 192.168.1.16 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.829/1.014/1.202/0.127 ms
```

The output above validates that the EOS node is reachable from the Ansible control host.

If the configured playbook or task is not using `connection: local`, then we can use SSH to validate that the SSH keyless login is working properly:

```
$ ssh ansible@192.168.1.16
Last login: Sun May  3 17:49:07 2015 from 192.168.1.130

Arista Networks EOS shell

[ansible@Arista ~]$
```

If the user (ansible in the above example) is unable to login to the node, please review the *Quick Start* guide to ensure you have SSH configured correctly.

Lastly, check to make sure the dependency eAPI has been enabled on the target Arista EOS node. To verify that eAPI is enabled and running, use the `show management api http-commands` command in EOS:

```
Arista#show management api http-commands
Enabled:        Yes
HTTPS server:   shutdown, set to use port 443
HTTP server:    running, set to use port 80
VRF:            default
Hits:           4358
Last hit:       59729 seconds ago
Bytes in:       680505
Bytes out:      64473935
Requests:       4278
Commands:       10918
Duration:       833.907 seconds
User       Hits       Bytes in       Bytes out    Last hit
---------- ---------- -------------- -------------- -----------------
eapi       4278       680505         64473935     59729 seconds ago

URLs
------------------------------------
Management1 : http://192.168.1.16:80
```

In the example command output above, check to be sure that `Enabled:` is `Yes` and either `HTTP server:` or `HTTPS server` is in a running state.

CHAPTER 7

Developer Information

7.1 Introduction

This section provides information for individuals that want to get started developing EOS modules. Whether adding new modules, extending existing modules or developing bug fixes, the details here explain how to get started working with the Ansible EOS role form source.

This section assumes that an Ansible development environment has already been created. For specific details on developing with Ansible please see the Developer Information found in the official Ansible documentation.

7.2 Running from Source

In order to get started running the Ansible EOS role from source, create a clone of the *develop* branch (or any other branch that you are interested in) on your local machine.:

```
$ git clone https://github.com/arista-eosplus/ansible-eos
Cloning into 'ansible-eos'...
remote: Counting objects: 486, done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 486 (delta 6), reused 0 (delta 0), pack-reused 450
Receiving objects: 100% (486/486), 1.66 MiB | 1.51 MiB/s, done.
Resolving deltas: 100% (303/303), done.
Checking connectivity... done.
```

Once the ansible-eos Github repository is installed, using the modules is as easy as passing the path to the ansible executable:

```
$ ansible -M /workspace/ansible-eos/library -m eos_vlan -a "vlanid=100"
```

Simply specify the module to be run (eos_vlan in the above example) and the arguments to pass to the module using the -a option.

7.3 Write Test Cases

The EOS role includes a number of modules for configuring resources on destination EOS nodes. All module test cases are defined in test/testcases. Test cases are defined as a simple YAML file that describes the module to run along with the arguments to be passed to the module. The test suite will then build an ansible command run it against a switch (either a hardware based model or vEOS).

In order to configure the test suite to run against switches in a given environment, modify the test/fixtures/eapi.conf and test/fixtures/hosts file to reflect the nodes to be tested.

Once the eapi.conf file and hosts file have been updated, use the following command to execute the test suite:

```
$ make tests
```

7.4 Contributing

The modules developed as part of the Ansible EOS role are supported by the Arista EOS+ community. We gladly accept and encourage contributions in the form of new modules, updated modules, test cases and documentation updates. Simply develop the changes and submit a pull request through Github.

For changes submitted by pull request, the Arista EOS+ community enforces some basic rules for new contributions.

1. New modules must be fully documented per Ansible module documentation standards
2. New or changed modules must include test cases that test the new module or new arguments made available in the module.

If you have any questions regarding module development or running modules from source, please feel free to contact Arista EOS+ at ansible-dev@arista.com

CHAPTER 8

FAQ

8.1 Introduction

The below list provides some answers to commonly asked questions about the Ansible EOS role.

8.1.1 What are the basic requirements for using the EOS role for Ansible?

This varies a little bit based upon how you communicate with your node. The two options are explained in *Topologies*.

Regardless of connection method you need the following:

- Ansible 1.9 or later
- Arista EOS 4.13.7M or later running on your node
- EOS Command API enabled (see *1. Enabling EOS Command API* for more information)

In addition to the above, there are other connection-specific requirements:

If you connect to your node via SSH:

- You need the Python Client for eAPI 0.3.0 or later **installed on your EOS node** (see *3. Install pyeapi* for more information)
- Linux shell account on your EOS node (see *2. Preparing EOS for Ansible* for more information)

If you connect to your node via eAPI:

- You need the Python Client for eAPI 0.3.0 or later **installed on your Ansible server** (see *2. Install pyeapi* for more information)

8.1.2 Is there any intention to encrypt passwords we put into eapi.conf?

No, we are working on support for certificates but we cannot encrypt the password in eapi.conf. The best alternative is to use Ansible vault or prompt for password at runtime of the playbook.

8.1.3 Is pyeapi required on both the Ansible control host and the EOS node?

It depends on if using (or want to use) connection local. The Python client for eAPI (pyeapi) must be installed where ever the Ansible module is executed from. The pyeapi client is a dependency of the common module code for all of the modules.

8.1.4 Do I have to use the pyeapi eapi.conf file?

No, it is not a absolute requirement. All EOS modules will accept connection parameters for configuring the eAPI transport properties. Using eapi.conf is convenient but not necessary.

8.1.5 Does the EOS role work with Ansible Tower?

Yes, the Ansible EOS role works fine with implementations that utilize Ansible Tower for management.

8.1.6 Does the Ansible EOS role work with all version of Arista EOS?

The Ansible EOS role is tested to work with EOS 4.13.7M or later releases. Any EOS release prior to 4.13.7M is not guaranteed to work with the EOS role.

8.1.7 Is there any requirement to put changes into ansible.cfg?

No, it works with all the Ansible defaults.

8.1.8 Is there something like a rollback function available in ansible?

Yes, it's all in the implementation. When working with a tool like Ansible, the node configuration should be kept under version control. As such, rolling back a nodes configuration is a matter of reverting the config. It's an implementation detail, not necessarily a module or feature. We have successfully demonstrated rollback many times using Ansible.

CHAPTER 9

Release Notes

9.1 v1.0.0

- adds support for pyeapi
- adds system test harness for testing modules against eos nodes
- adds stateful common module

9.1.1 New Modules

- eos_command.py
- eos_config.py
- eos_ethernet.py
- eos_facts.py
- eos_interface.py
- eos_ipinterface.py
- eos_mlag_config.py
- eos_mlag_interface.py
- eos_portchannel.py
- eos_purge.py
- eos_stp_interface.py
- eos_switchport.py
- eos_system.py
- eos_vlan.py
- eos_vxlan.py
- eos_vxlan_vlan.py
- eos_vxlan_vtep.py

9.2 v1.0.1

- adds additional parameters to eos_config
- adds vlan argument to eos_vxlan_vtep
- fixes issue with honoring enablepwd if specified
- fixes #37

9.2.1 New Modules

None

9.3 v1.1.0

- adds trunk_groups argument to eos_switchport
- changes trunk_allowed_vlans to allow a range of vlans
- changes arguments used by eos_config (see documentation)
- fixes an issue with out of order vlans in eos_switchport

9.3.1 New Modules

None

9.4 v1.2.0

2015-11-05

9.4.1 New Modules

- **Add eos_vrrp (78) [grybak]** Add the eos_vrrp module. This module controls interface VRRP configuration. (Requires pyeapi update)
- **Feature staticroute (68) [grybak]** Adds the eos_staticroute module to perform configuration management of static ip routes. (Requires pyeapi update)
- **Add eos_varp and eos_varp_interface modules (67) [phil-arista]** Adds the eos_varp and eos_varp_interface modules. The eos_varp module provides management of the system's virtual mac address. The eos_varp_interface manages virtual-router ip addresses within Vlans. (Requires pyeapi update)
- **Add eos_routemap module (66) [phil-arista]** The eos_routemap module provides configuration management of system route-maps. (Requires pyeapi update)
- **Add eos_user module (51) [phil-arista]** The eos_user module adds the ability to manage CLI users. All user attributes are configurable including SSH Key support. (Requires pyeapi update)

9.4.2 Enhancements

- **Added encoding option to command module. (65) [chepazzo]** The eos_command module now only supports enable commands. This enhancement allows you to pass an encoding option. The choices are json and text. The encoding option determines the format of the returned output.
- **Add support for maximum-paths (64) [phil-arista]** This enhancement adds the ability to define BGP maximum paths and maximum ecmp paths. (Requires pyeapi update)
- **Add ip routing (61) [phil-arista]** This enhancement augments the eos_system module. It now provides the ability to enable `ip routing`. (Requires pyeapi update)

9.4.3 Fixed

- **eos_ping should analyze loss instead of errors (53)** Due to variations in EOS ping output, it became necessary to analyze loss instead of errors.
- **eos_ping fails when network is unreachable (52)** The eos_ping module will now successfully exit even when the ping result is `network unreachable`
- **eos_ping resuses 'host' argument (47)** The eos_ping module used the attribute `host` which caused a conflict with the meta argument `host`. The updated attribute is called `dst`.
- **port-channel set to mode "on" not "active" on initial pass (36)** The eos_portchannel module runs set_lacp_mode before set_members. This means that when set_members is run, you end up with the default lacp mode instead of the mode you defined. Now, the set_members method includes a mode keyword. (Required pyeapi update)

9.5 v1.3.0

2016-02-17

9.5.1 New Modules

- None

9.5.2 Enhancements

- **Enhance autorefresh (88) [phil-arista]** This knob is accessible in the module and is turned off by default. This reduces the number of 'show run all' that are executed during a module run.
- **Workaround for Ansible 2.0 changes in AnsibleModule._log_invocation(). (85) [chepazzo]** Modify module logging to accommodate Ansible 2.0 core changes.
- **Add disable key to existing modules for negation of properties (84) [grybak-arista]** Implements a disable key in modules for negation of properties, when appropriate.

9.5.3 Fixed

- **Enable/Disable logic incorrect in modules with command_builder (86)** The command builder in pyeapi was updated to make logic more uniform across all APIs. This required an update to the Ansible modules. This bug addresses some modules that did not get updated on the first go.

- **[eos_vlan] Set_trunks doesn't pass correct value to API (82)** The eos_vlan module did not properly separate the trunk groups when calling the set_trunks API method. This fixes that issue within the module. No change to pyeapi.
- **eos_interface defaulting an interface (73)** The common/eos.py code was fixed to allow flexible support of state methods within the module. This issue was resolved with that addition to the common code along with an added 'default' method within the module to call the interfaces API default method. Note that state=default is not an idempotent operation. It will run every time since the resulting state will be state=present.
- **eos_bgp_* modules take a long time to complete (59)** This has been improved. It's still not lightning fast since 'show run all' is used to parse the config. In PR #88 we add a knob to control pyeapi's autorefresh, so the running config will only get pulled down 1x (max 2x if router bgp is created) and then all other commands will get run to configure the attributes of the bgp config.
- **Using "params['connection']" in the modules means that the [DEFAULT] section configuration in pyeapi (eapi.conf) will n** This issue has been retested with the latest code and is no longer present. Note: It is unclear at what point this was resolved.

9.5.4 Known Caveats

- domain_id parameter of eos_mlag_config module doesn't support '-' and dot (90)

9.5.5 Notes

- This ansible-eos release should be partnered with min pyeapi version 0.5.0
- Ansible is releasing new networking modules into the core ansible code. These new modules will allow you to easily work with Jinja templates to implement your running-config. They are also releasing eos_facts and eos_commands modules which will make it easier to get up and running. Please contact us at ansible-dev@arista.com for more information.

CHAPTER 10

License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the {organization} nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.