Solution Guide

Open Compute Network Operating System Version 1.2.4

OcNOS™ Validated Solution Guide
Network Automation
OcNOS™ ZTP and Ansible

ipinfusion™

Contents

Contents cont.

Glossary

ZTP	Zero Touch Provisioning
DHCP	Dynamic Host Configuration Protocol
YAML	YAML Ain’t Markup Language
SSH	Secure Shell
EBGP	External BGP
ECMP	Equal Cost Multipath
OPEX	Operational Expenditures
OcNOS	Open Compute Network Operating System

Network Automation

Network Automation provides IT Administrators and Network Operators significant benefits. By deploying network automation, Network Operators are able to:

- Decrease their operating costs while improving the quality and consistency of the network.
- Maximize the value of their IT organization, freeing highly skilled engineers from manual tasks and focus on key business initiatives and quality services.
- Deliver a holistic network capable of intelligent change monitoring and management, integrated fault management, compliance enforcement, vulnerability detection, and software deployment.

To address these issues, OcNOS offers two solutions that provide robust network automation:

- Zero Touch Provisioning (ZTP) – Zero touch provisioning is designed to get a switch with all necessary configurations and into service without manual intervention. It involves from getting the correct software image file, IP addresses, and running configuration.
- Ansible – Rather than treating the network as a group of separate devices, Ansible models the data center by describing how all of the systems interrelate. Using a simple IT engine, Ansible easily automates provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.

From new device deployments to configuration updates on large-scale IT infrastructures, Ansible together with ZTP can expedite these tasks by automating their operations.

The remainder of this solution guide takes a closer look at the structure of both ZTP and Ansible, operations. Additionally, an actual real-world example of provisioning a CLOS network is provided for those who want to learn more about the deployment of these automation tools.

Zero Touch Provisioning (ZTP)

In any Data Center deployment, there are typically around 50-100 new switches to configure. Most switches have an identical configuration (except for some unique parameters, like the management IP address), and the new switches will almost always need their software updated to a standard version. Typically, deploying a new network would mean logging into every switch via the console port, repeating a configuration from a template, and then upgrading the software. With multiple switches to build, it is time consuming and error prone. OcNOS provides complete Zero Touch Provisioning through DHCP with a set of vendor options, which can be easily configured on a DHCP Server.

DHCP Option Codes

OcNOS ZTP works using DHCP server options. These options can be configured on any standard DHCP server. As per the DHCP option several actions can be taken, a few of them relevant to ZTP are mentioned below.

Table 1. DHCP Option codes used for Provisioning OcNOS through ZTP

Option String	Option Code	Option Type	Vendor	Meaning
default-url	114	URL	ONIE	Location of the image file
ocnos-license-url	251	Text	OcNOS	Location of the license file
ocnos-provision-url	250	Text	OcNOS	Location of the configuration file

By default, when the switch boots up, if it does not have an OcNOS image, the ONIE installer sends a DHCP discover to get the IP address for the management Ethernet interface. If the DHCP Server is configured with a “default-url “option pointing to the URL of the OcNOS image, it responds with option 114. ONIE installer will fetch the image and starts the installation process.

Once the installation of the image is complete, the switch boots up with the OcNOS image. This time, OcNOS sends a DHCP Discover. If the DHCP Server responds with with the “ocnos-license-url” and the “ocnos-provision-url” options, OcNOS gets the license file and the configuration files and applies them automatically.

Ansible

Ansible can configure OcNOS devices using Ansible’s automation framework. The OcNOS Ansible module can be used to send any arbitrary configuration command to the devices running OcNOS. The OcNOS Ansible module can also be used to create Ansible Playbooks that automate provisioning network services.

Ansible works by connecting to nodes and pushing out small programs to them called “Ansible Modules.” These programs are written to be resource models of the desired state of the system. Ansible then executes these modules over SSH (by default), and removes them when finished.

The library of modules can reside on any machine, with no servers, daemons, or databases required. Typically, Ansible works with any standard terminal program, a text editor, and (ideally) a version control system to keep track of changes to the contents.

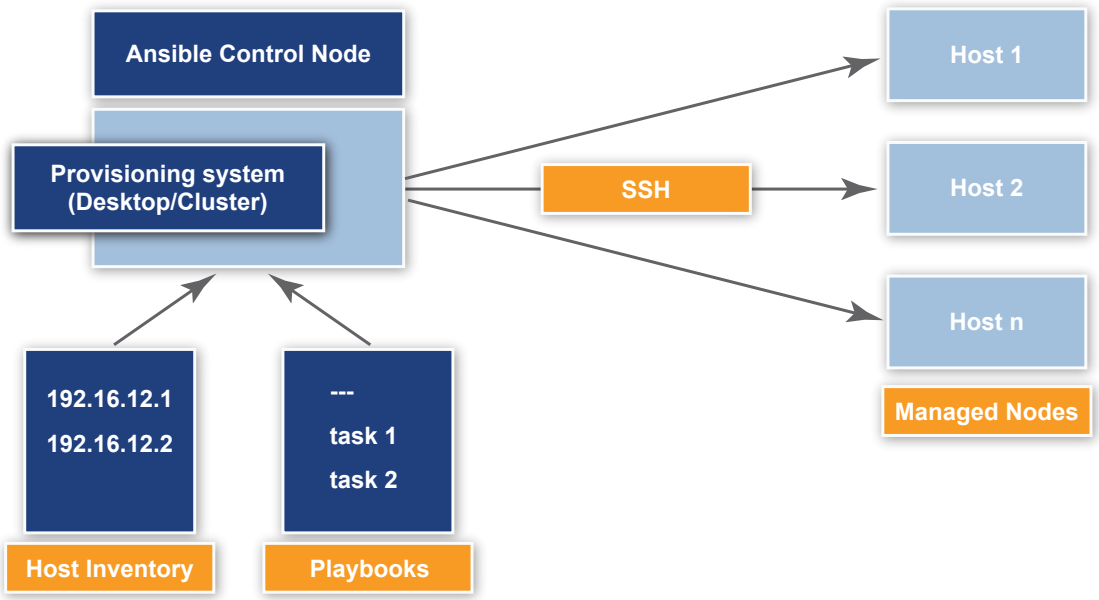
Ansible for Configuration Templating

While ZTP provides provisioning automation using DHCP, generating configuration files for switches can be automated using Ansible.

Ansible uses Jinja2 templating for populating the variables in a template-based configuration file. The variables for the templates are placed in a YAML formatted file. Based on the topology, the switches can be categorized into different roles and the template files for each role can be generated using Ansible as per the directory structure shown in the table below.

Table 2. Ansible directory structure for each role

Directory	Description
tasks	A main.yml file that should include a list of the tasks for this role. Tasks can be split and organized in this folder.
templates	Location where the template module looks automatically for the jinja2 templates included in the roles.
vars	A main.yml file, which contains the variables for this role.



Control Machine Requirements

Ansible can run on any Linux machine with Python 2.7 or later.
Please refer to Ansible documentation for compatibility and running on other operating systems and platforms.

Managed Node Requirements

On the managed nodes, you need a way to communicate, which is normally SSH. By default SSH uses sftp. If sftp is not available, you can switch to scp in /etc/ansible/ansible.cfg:

```
# if True, make ansible use scp if the connection type is ssh
# (default is sftp)
#scp_if_ssh = True
```

Also, the managed nodes need OcNOS 1.2.1 version or later.

Installation on Control Machine

1. Install Ansible:

```
pip install ansible
```

Note: Any installation dependency needs to be resolved.

2. Download the OcNOS Ansible modules from: https://github.com/IPInfusion/OcNOS/tree/1.2/

3. Copy the OcNOS Ansible modules to the default module location:

```
/usr/lib/<python-directory>/site-packages/ansible/modules/core/system/
```

The <python-directory> depends on the Python version used on the control machine. For example, if Python 2.7 is used:

```
/usr/lib/python2.7/site-packages/ansible/modules/core/system
```

Inventory Configuration

The inventory is a description of the nodes that can be accessed by Ansible. By default, the Inventory is described by a configuration file, in INI format, and whose default location is in /etc/ansible/hosts. The configuration file lists either the IP address or hostname of each node that is accessible by Ansible. In addition, nodes can be assigned to groups.

This is an example of a configuration file:

```
192.168.12.10
[example-hosts]
host1 host=10.12.23.28 port=22 username=user password=user123
host2 ansible _ ssh _ host=10.12.23.29 ansible _ ssh _ user=user ansible _ ssh _ pass=user123
host3 ansible _ ssh _ host=10.12.26.29
foo.example.com
bar.example.com

[local]
localhost ansible _ connection=local
```

The same host can be part of multiple groups.

Defining Variables

An inventory file can also define variables used in playbooks.

Define Variables per Host

```
[OCNOS]
OcNOS1 ansible _ ssh _ host=10.12.44.34 ansible _ ssh _ user=user ansible _ ssh _ pass=user procId=350
nw=1.1.1.1/24 areaId=1
OcNOS2 ansible _ ssh _ host=10.12.44.33 ansible _ ssh _ user=user ansible _ ssh _ pass=user procId=150
nw=2.2.2.2/24 areaId=2
```

Define Common Variables for a Whole Group

```
[OCNOS]
host1 ansible _ ssh _ host=10.12.45.251 ansible _ ssh _ user=user ansible _ ssh _ pass=user
host2 ansible _ ssh _ host=10.12.45.252 ansible _ ssh _ user=user ansible _ ssh _ pass=user
host3 ansible _ ssh _ host=10.12.45.253 ansible _ ssh _ user=user ansible _ ssh _ pass=user

[OCNOS:vars]
hostnameid=OcNOS
procId=100
```

Defining Variables in Separate Files

The preferred practice in Ansible is actually not to store variables in the main inventory file. In addition to storing variables directly in the INI file, host and group variables can be stored in individual files relative to the inventory file:

- Group variables can be defined in this path /etc/ansible/group_vars/
- Host variables can be defined in this path /etc/ansible/host_vars/

The file name should be the same as the group name/host name:

```
cat /etc/ansible/group _ vars/OCNOS
new _ var: 5
```

For more about inventory file and variable definitions, refer to the following links:

- http://docs.ansible.com/ansible/intro_inventory.html
- http://docs.ansible.com/ansible/playbooks_variables.html

Modules

Modules are the units of work in Ansible. Each module is for the most part standalone, and can be written in a standard scripting language such as Python, Perl, Ruby, or bash.

OcNOS-Ansible provides the ocnos_configs module for configuring OcNOS devices

ocnos_configs

The `ocnos _ configs` module prepares and executes OcNOS configuration commands for given Ansible tasks.

Place the commands to be executed with their input values:

```
bridge 1 protocol ieee vlan-aware
```

Placeholders (“?”) are useful when the same command needs to be executed multiple times with different values. Use placeholders instead of actual input values in commands:

```
bridge ? protocol ? ? | 1,ieee,vlan-aware; 2,mstp,ring; 3,rstp
```

The example above executes the bridge protocol command 3 times with parameters as follows:

```
bridge 1 protocol ieee vlan-aware
bridge 2 protocol mstp ring
bridge 3 protocol rstp
```

The characters below can be used with placeholders:

|	Separates a command from values
,	Separates values for placeholders
;	Separates a set of inputs for one command iteration
$	Separates commands given in the same line

The ocnos_configs file supports the following types of sub tasks:

- `config _ cmds:` Input is executed in ZebOS-XP configuration mode
- `exec _ cmd:` Input is executed in ZebOS-XP exec mode
- `module _ config _ cmds:` Input is executed as a group in ZebOS-XP configuration mode

Basic Ansible Commands

Note: Add an SSH Host Authentication Key (RSA) to the list of known hosts on a control machine before executing any Ansible command on a host, as Ansible uses ssh to communicate with host nodes.

Ping Host Nodes

```
[OCNOS is a group of two hosts defined in host file]

[ping is Ansible’s core module]

[root@localhost]# ansible OCNOS -m ping
```

```
ocnos1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
ocnos2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Run a live command on all hosts:

```
[root@localhost]# ansible OCNOS -a '/bin/echo Hello'
ocnos1 | SUCCESS | rc=0 >>
Hello
ocnos2 | SUCCESS | rc=0 >>
Hello
```

Gathering Facts

```
[root@localhost]# ansible OCNOS -m setup
ocnos2 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.12.23.29"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::207:e9ff:fea5:1a89",
            "fe80::207:e9ff:fea5:1a88",
            "fe80::202:a5ff:fe4e:ee76"
        ],
        "ansible_architecture": "i386",
        "ansible_bios_date": "04/06/2011",
        "ansible_bios_version": "J01 v02.01",
        "ansible_cmdline": {
```

Get the Running Configuration

```
[root@localhost]# ansible OCNOS -a 'ocsh -e "show running-config"'
ocnos1 | SUCCESS | rc=0 >>
!
no service password-encryption
!
logging monitor 7
!
ip vrf management
!
forwarding profile l2-profile-three
ip domain-lookup
bridge 1 protocol mstp
tfo Disable
data-center-bridging enable bridge 1
feature telnet
feature ssh
no feature tacacs+
no feature ldap
```

```
snmp-server view all .1 included
snmp-server enable snmp
ntp enable
username ocnos role network-admin password encrypted $1$Mqm5VOt.$YUVowR2V4a9aiciX0YDwu.
sFlow disable
!
vlan database
 vlan 10-20 bridge 1 state enable
!
spanning-tree mst configuration
!
ip pim register-rp-reachability
!
interface eth0
 ip address 10.12.45.253/8
!
interface lo
 ip address 127.0.0.1/8
 ipv6 address ::1/128
!
interface xe1
!
interface xe2
 switchport
 bridge-group 1
 switchport mode hybrid
 switchport hybrid allowed vlan add 10 egress-tagged disable
!
interface xe3
!
interface xe4
!
line con 0
 login
line vty 0 39
 login
!
end
```

Playbooks

Ansible uses a simple language called YAML to create what are called “Ansible Playbooks.” Playbooks express configurations, deployments, and orchestrations in Ansible. Each playbook maps a group of hosts to a set of roles. Each role is represented by calls to Ansible tasks.

A sample playbook for the OcNOS Ansible module is shown below:

```
- hosts: OCNOS
  tasks:
    - ocnos _ configs:
         config _ cmds:
               - ‘bridge ? protocol ? ? | 1, ieee; 2,mstp,ring; 3,rstp $ interface eth3’
    - ocnos _ configs:
          config _ cmds:
                - ‘vlan ? protocol ? ? | 1,ieee; 2,mstp,ring; 3,rstp’
                - ‘hostname {{ hostnameid }}’
          module _ config _ cmds:
                - ‘interface ? $ bandwidth ? | eth1; 1g; $ eth2; 10g;’
    - ocnos _ configs:
             exec _ cmds:
                    - ‘wr’
```

The OcNOS Ansible module takes ocnos_config tasks as input and prepares an OcNOS configuration command, that is executed on hosts:

```
- ocnos _ configs:
      config _ cmds:
           - 'bridge ? protocol ? ? | 1, ieee; 2,mstp,ring; 3,rstp $ interface eth3'
```

For the above task, the OcNOS command will look as follows:

```
ocsh -k -e 'conf t' -e 'bridge 1 protocol ieee' -e 'bridge 2 protocol mstp ring' -e 'bridge 3
protocol rstp' -e 'interface eth3
'
```

Note: Commands run in non-strict mode and errors from ZebOS modules are ignored.

Looping in a Playbook

Ansible supports looping in playbooks, which is useful while configuring the same set of commands on multiple object instances:

```
- hosts: LEAF1
  ignore _ errors: yes
  tasks:
   - ocnos _ configs:
       config _ cmds:
         - interface {{ item }}
         - 'switchport'
         - 'bridge-group 1'
         - 'switchport mode trunk'
         - 'switchport trunk allowed vlan add 10'
         - 'switchport trunk allowed vlan add 20'
         - 'switchport trunk allowed vlan add 30'
         - 'static-channel-group 1'
     with _ items:
         - xe5
         - xe6
```

Executing a Playbook

1. Create a playbook file with a .yml extension
2. Execute as:

```
ansible-playbook <name>.yml
```

Enable Logging

Enable logging in `/etc/ansible/ansible.cfg`:

```
# logging is off by default unless this path is defined
# if so defined, consider logrotate
#log _ path = /var/log/ansible.log
```

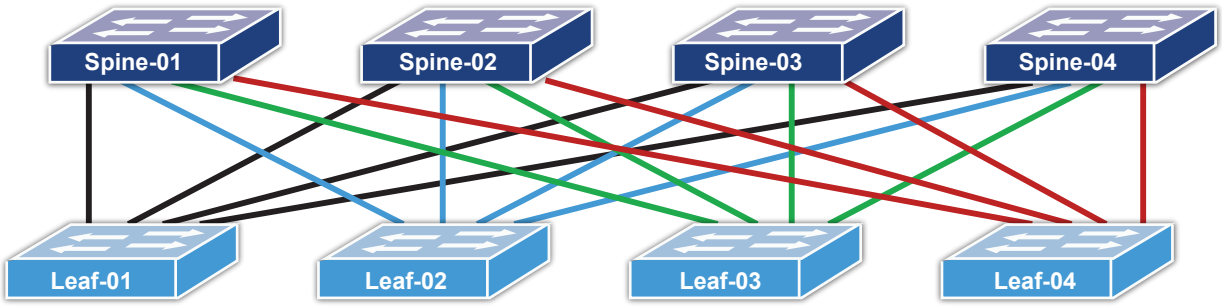
Reference

Find more information on Ansible at http://docs.ansible.com/index.html

APPENDIX: Using Ansible and ZTP to Perform EBGP IP CLOS Fabric Provisioning

A CLOS network is a multi-stage switching arrangement that provides non-blocking performance. In the provisioning example, we have 4 leafs connecting to 4 spines. See the topology shown in Figure 1.

Figure 1. Spine Leaf Topology



The uplinks from each leaf are color coded to mark the identity of the leaf spine connectivity.

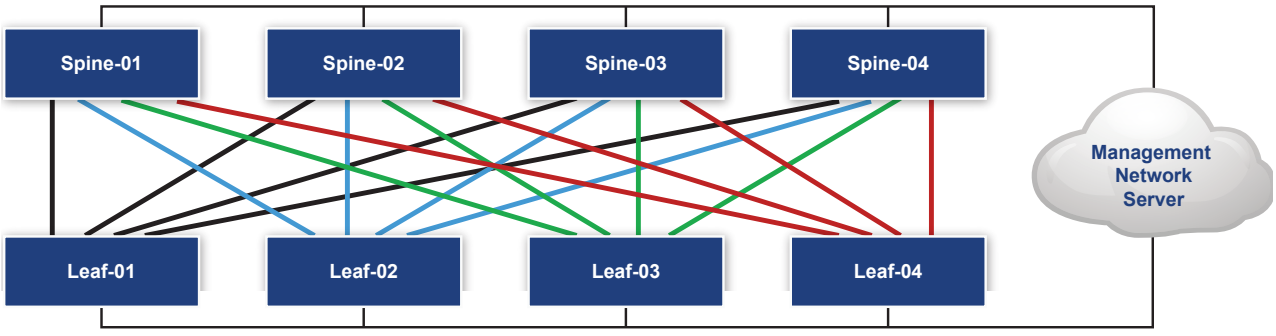
Table 1. Spine and Leaf Color Coding

Leaf Switch	Spine Switch	Color Code
Leaf-01	Spine-01	Black
Leaf-02	Spine-02	Blue
Leaf-03	Spine-03	Green
Leaf-04	Spine-04	Red

Note: For simplicity, we consider the Management Ethernet interfaces of all the switches to be connected to an Out of Band (OOB) management network.

The DHCP Server, HTTP Server for the OcNOS Images, Licenses, and Configuration files all reside on the management network. In this example, for simplicity, all servers are located in the same host.

Figure 2. Topology Connected to an Management Network



The recommended OS for the OOB host is Ubuntu, which can be downloaded here:

http://mirror.pnl.gov/releases/wily/ubuntu-15.10-desktop-i386.iso

The OS should contain Python 2.7.10, and Ansible can be installed via

In the example shown below, the network address of the interface of the host is assigned as 10.10.100.250/24. The DHCP Server responds with address in the range of 10.10.100.0/24.

Table 2. Spine Switch Common Parameters

Leaf Common Parameters	Spine-01	Spine-02	Spine-03	Spine-04
MAC Address (Management Ethernet)	ec:f4:bb:fd:5b:81	ec:f4:bb:fd:5b:82	ec:f4:bb:fd:5b:83	ec:f4:bb:fd:5b:84
Management IP address	10.10.100.1	10.10.100.2	10.10.100.3	10.10.100.4
License URL	http://10.10.100.250/licenses/leaf01_license.bin	http://10.10.100.250/licenses/leaf02_license.bin	http://10.10.100.250/licenses/leaf03_license.bin	http://10.10.100.250/licenses/leaf04_license.bin
Configuration File URL	http://10.10.100.250/licenses/leaf01_conf	http://10.10.100.250/licenses/leaf02_conf	http://10.10.100.250/licenses/leaf03_conf	http://10.10.100.250/licenses/leaf04_conf
Router ID	1.1.1.1/32	2.2.2.2/32	3.3.3.3/32	4.4.4.4/32

Table 3. Leaf Switch Common Parameters

Leaf Common Parameters	Leaf-01	Leaf-02	Leaf-03	Leaf-04
MAC Address (Management Ethernet)	ec:f4:bb:fd:5b:85	ec:f4:bb:fd:5b:86	ec:f4:bb:fd:5b:87	ec:f4:bb:fd:5b:88
Management IP address	10.10.100.5	10.10.100.6	10.10.100.7	10.10.100.8
License URL	http://10.10.100.250/licenses/leaf01_license.bin	http://10.10.100.250/licenses/leaf02_license.bin	http://10.10.100.250/licenses/leaf03_license.bin	http://10.10.100.250/licenses/leaf04_license.bin
Configuration File URL	http://10.10.100.250/licenses/leaf01_conf	http://10.10.100.250/licenses/leaf02_conf	http://10.10.100.250/licenses/leaf03_conf	http://10.10.100.250/licenses/leaf04_conf
Router ID	5.5.5.5/32	6.6.6.6/32	7.7.7.7/32	8.8.8.8/32

Table 4. Network Parameters for EBGP IP CLOS Topology

Spine Switch	Spine Interface	Spine IP Address	Leaf Switch	Leaf Interface	Leaf IP Address	Spine BGP ASN	Leaf BGP ASN
Spine-01	xe1/1	192.168.15.0/31	Leaf-01	xe1/1	192.168.15.1/31	65501	65412
Spine-01	xe1/2	192.168.16.0/31	Leaf-02	xe1/1	192.168.16.1/31	65501	65413
Spine-01	lxe1/3	192.168.17.0/31	Leaf-03	xe1/1	192.168.17.1/31	65501	65414
Spine-01	xe1/4	192.168.18.0/31	Leaf-04	xe1/1	192.168.18.1/31	65501	65415
Spine-02	xe1/1	192.168.25.0/31	Leaf-01	xe1/2	192.168.25.1/31	65502	65412
Spine-02	xe1/2	192.168.26.0/31	Leaf-02	xe1/2	192.168.26.1/31	65502	65413
Spine-02	lxe1/3	192.168.27.0/31	Leaf-03	xe1/2	192.168.27.1/31	65502	65414
Spine-02	xe1/4	192.168.28.0/31	Leaf-04	xe1/2	192.168.28.1/31	65502	65415
Spine-03	xe1/1	192.168.35.0/31	Leaf-01	xe1/3	192.168.35.1/31	65503	65412
Spine-03	xe1/2	192.168.36.0/31	Leaf-02	xe1/3	192.168.36.1/31	65503	65413
Spine-03	lxe1/3	192.168.37.0/31	Leaf-03	xe1/3	192.168.37.1/31	65503	65414
Spine-03	xe1/4	192.168.38.0/31	Leaf-04	xe1/3	192.168.38.1/31	65503	65415
Spine-04	xe1/1	192.168.45.0/31	Leaf-01	xe1/4	192.168.45.1/31	65504	65412
Spine-04	xe1/2	192.168.46.0/31	Leaf-02	xe1/4	192.168.46.1/31	65504	65413
Spine-04	lxe1/3	192.168.47.0/31	Leaf-03	xe1/4	192.168.47.1/31	65504	65414
Spine-04	xe1/4	192.168.48.0/31	Leaf-04	xe1/4	192.168.48.1/31	65504	65415

The common parameters are used to build the “dhcpd.conf” file that is used by the DHCP Server. The Network parameters are used to build the configuration files for Spine and Leaf switches. Ansible playbooks are used to populate these parameters to the template files which are detailed in the next section.

Ansible Playbook Creation

Ansible uses a concept called a “Playbook” to perform more than one operation at a time. Playbooks are expressed in YAML, which uses a standard YAML Parser.

Ansible Playbooks are made up of one or more plays, and a play consists of three sections:

- The **target section** defines the hosts on which the play will be run and how it will be run. This is where we set the SSH username and other SSH-related settings.
- The **variable section** defines variables, that are made available to the play while running.
- The **task section** lists all the modules in the order that we want them to be run by Ansible.

In our example, we will be creating a playbook named “ebgp-clos.yml”.

For a larger project, such as data center provisioning automation, Ansible provides “roles” that allow us to group files together in a defined structure. Roles allow us to place variables, files, tasks, templates, and handlers in a folder, and then include them when needed (refer back to Table 2).

For this example, three roles are created.

- DHCP Role – takes the input from the common parameters and generates a configuration file from a DHCP template configuration.
- Spine Role – takes the input from the network parameters and generates a configuration file from an OcNOS template configuration.
- Leaf Role – takes the input from the network parameters and generates a configuration file from an OcNOS template configuration.
- Once the playbook is created, it can be run using the command ansible-playbook ebgp-clos.yml. This generates configuration files in appropriate folders so that ZTP can handle the rest.
- For each role, we need to follow a three step process:

1. Create a “main.yml” inside of the vars folder by specifying the parameters.
2. Generate the configuration template file in the template folder.
3. Create a “main.yml” inside the tasks folder which uses the built-in template module to identify the template placed in the template folder.

The next sections are a step-by-step guide for creating and running the playbook:

- Initial configuration generation using the OcNOS Ansible module
- Configuration during run time using the OcNOS Ansible module

Initial Configuration Generation using OcNOS Ansible Module

Step 1: Create Roles Folder in the Ansible Directory

We have identified three roles (dhcp, spine, and leaf roles) and the folders with the same names are created in the roles folder in the default Ansible directory. The default Ansible directory is “/etc/ansible”. In each folder we create tasks, templates, and vars folders.

```
root@MadDog:/etc/ansible# mkdir -p roles/{dhcp/{tasks,templates,vars},spine/{tasks,templates, vars},
leaf/{tasks,templates,vars}}
root@MadDog:/etc/ansible# tree
.
├── ansible.cfg
├── hosts
└── roles
    ├── dhcp
    │   ├── tasks
    │   ├── templates
    │   └── vars
    ├── leaf
    │   ├── tasks
    │   ├── templates
    │   └── vars
    └── spine
        ├── tasks
        ├── templates
        └── vars

13 directories, 2 files
root@MadDog:/etc/ansible#
```

Step 2: Create Variables, Templates and Tasks files for the DHCP Role

Step 2.1: Create a Variable File for DHCP Role

The variable file is a YAML file “main.yml” and the inputs are from the common parameters table. All the input parameters for the eight devices in the example topology are populated using YAML Structure. Each device is represented as a hostname and each hostname has a list of parameters. Refer to Figure 4 for the complete YAML file. This YAML file is placed in the vars folder in the dhcp folder.

Step 2.2: Create a DHCP Configuration Template File

The configuration template file for the DHCP server needs to be created in the templates folder of the dhcp folder. This file is a Jinja2 template and is named dhcp.j2. Refer to Figure 5 for the completed configuration template file. The Jinja2 Template engine provides a looping feature which is used in the template.

Step 2.3: Create a Task File

The task file is a YAML file “main.yml” which uses the default Template module of Ansible. This YAML file goes into the tasks folder inside the dhcp folder. Refer to Figure 3 for the contents of the YAML file inside the tasks folder.

Figure 3. YAML File in the Tasks Sub-folder in the dhcp Folder

```
- name: Generate DHCP Configuration file  template: src={{ item.profile }}.j2 dest=/etc/dhcp/dhcpd.conf
  with _ items:
   dhc _ segment
```

Figure 4. YAML File Inside the vars Sub-folder in the dhcp Folder

```
---
dhcp _ segment:
 - topology: ebgp _ ip _ clos _ topology
   profile: dhcp
   devices:
    - hostname: Leaf _ 01 _ device
      mac _ address: ec:f4:bb:fd:5b:85
      fixed _ address: 10.10.100.5
      license _ url: http://10.10.100.250/licenses/leaf01 _ license.bin
      provision _ url: http://10.10.100.250/configs/leaf-01.conf
    - hostname: Leaf _ 02 _ device
      mac _ address: ec:f4:bb:fd:5b:86
      fixed _ address: 10.10.100.6
      license _ url: http://10.10.100.250/licenses/leaf02 _ license.bin
      provision _ url: http://10.10.100.250/configs/leaf-02.conf
    - hostname: Leaf _ 03 _ device
      mac _ address: ec:f4:bb:fd:5b:87
      fixed _ address: 10.10.100.7
      license _ url: http://10.10.100.250/licenses/leaf03 _ license.bin
      provision _ url: http://10.10.100.250/configs/leaf-03.conf
    - hostname: Leaf _ 04 _ device
      mac _ address: ec:f4:bb:fd:5b:88
      fixed _ address: 10.10.100.8
      license _ url: http://10.10.100.250/licenses/leaf04 _ license.bin
      provision _ url: http://10.10.100.250/configs/leaf-04.conf
    - hostname: Spine _ 01 _ device
      mac _ address: ec:f4:bb:fd:5b:81
      fixed _ address: 10.10.100.1
      license _ url: http://10.10.100.250/licenses/spine01 _ license.bin
      provision _ url: http://10.10.100.250/configs/spine-01.conf
    - hostname: Spine _ 02 _ device
      mac _ address: ec:f4:bb:fd:5b:82
      fixed _ address: 10.10.100.2
      license _ url: http://10.10.100.250/licenses/spine02 _ license.bin
      provision _ url: http://10.10.100.250/configs/spine-02.conf
    - hostname: Spine _ 03 _ device
      mac _ address: ec:f4:bb:fd:5b:83
      fixed _ address: 10.10.100.3
      license _ url: http://10.10.100.250/licenses/spine03 _ license.bin
      provision _ url: http://10.10.100.250/configs/spine-03.conf
    - hostname: Spine _ 04 _ device
      mac _ address: ec:f4:bb:fd:5b:84
      fixed _ address: 10.10.100.4
      license _ url: http://10.10.100.250/licenses/spine04 _ license.bin
      provision _ url: http://10.10.100.250/configs/spine-04.conf
```

Figure 5. Jinja2 dhcp.j2 Template File in the Templates Sub-folder in the dhcp Folder

```
default-lease-time 600;
max-lease-time 7200;
option ocnos-provision-url code 250 = text;
option ocnos-license-url code 251 = text;
option subnet-mask 255.255.255.0;
option broadcast-address 10.10.100.255;
```

```
option routers 10.10.100.250;
option default-url "http://10.10.100.250/images/DELL _ S6000 _ ON-OcNOS-1.2.0.199-DC _ MPLS-S0-P0-
installer";

subnet 10.10.100.0 netmask 255.255.255.0 {
range 10.10.100.1 10.10.100.200;
{% for items in item.devices %}
     host {{items.hostname}} {
            hardware ethernet {{ items.mac _ address }};
            fixed-address {{ items.fixed _ address }};
            option ocnos-license-url "{{items.license _ url}}";
            option ocnos-provision-url "{{items.provision _ url}}";
     }

{% endfor %}
}
```

Step 3: Create Variables, Templates, and Task Files for Leaf and Spine Roles

Creating variable, Template, and Task files for the switches in the leaf and spine roles uses a similar process to that described in step 2 for the DHCP role. The variables file is a YAML file named "main.yml" and the inputs for these files are based on the network parameters described in Table 7. Figure 6 shows the content of the YAML files for the leaf and spine roles. Only the first hostname is shown and the rest of the hostnames are filled up from the network parameters table.

Figure 6. Variable Files "main.yml" of the Leaf and Spine Role

```
 ---
switches:
 - hostname: leaf-01
   profile: leaf
   loopback: 5.5.5.5
   local _ ASN: 65412
   links:
    -  leaf _ port: xe1/1
       spine _ switch: Spine-01
       leaf _ port _ local _ ip: 192.168.15.1
       spine _ port _ remote _ ip: 192.168.15.0
       remote _ ASN: 65501
    -  leaf _ port: xe1/2
       spine _ switch: Spine-02
       leaf _ port _ local _ ip: 192.168.25.1
       spine _ port _ remote _ ip: 192.168.25.0
       remote _ ASN: 65502
    -  leaf _ port: xe1/3
       spine _ switch: Spine-03
       leaf _ port _ local _ ip: 192.168.35.1
       spine _ port _ remote _ ip: 192.168.35.0
       remote _ ASN: 65503
    -  leaf _ port: xe1/4
       spine _ switch: Spine-04
       leaf _ port _ local _ ip: 192.168.45.1
       spine _ port _ remote _ ip: 192.168.45.0
       remote _ ASN: 65504
 - hostname: leaf-02
 - hostname: leaf-03
 - hostname: leaf-04
```

```
---
switches:
 - hostname: Spine-01
   profile: spine
   loopback: 1.1.1.1
   local _ ASN: 65501
   links:
    -  spine _ port: xe1/1
       leaf _ switch: Leaf-01
       spine _ port _ local _ ip: 192.168.15.0
       leaf _ port _ remote _ ip: 192.168.15.1
       remote _ ASN: 65412
    -  spine _ port: xe1/2
       leaf _ switch: Leaf-02
       spine _ port _ local _ ip: 192.168.16.0
       leaf _ port _ remote _ ip: 192.168.16.1
       remote _ ASN: 65413
    -  spine _ port: xe1/3
       leaf _ switch: Leaf-03
       spine _ port _ local _ ip: 192.168.17.0
       leaf _ port _ remote _ ip: 192.168.17.1
       remote _ ASN: 65414
    -  spine _ port: xe1/4
       leaf _ switch: Leaf-04
       spine _ port _ local _ ip: 192.168.18.0
       leaf _ port _ remote _ ip: 192.168.18.1
       remote _ ASN: 65415
 - hostname: Spine-02
 - hostname: Spine-03
 - hostname: Spine-04
```

Figure 7. Tasks “main.yml” File for Leaf Role ---

```
- name: Generate Leaf configuration files
  template: src={{ item.profile}}.j2 dest=/tmp/configs/{{item.hostname}}.conf
  with _ items:
   switches
```

Figure 8. Tasks” main.yml” File for Spine Role

```
---
- name: Generate Spine Configuration files
  template: src={{ item.profile }}.j2 dest=/tmp/configs/{{ item.hostname }}.conf
  with _ items:
   switches
```

Figure 9. OcNOS Configuration Spine Template”spine.j2”

```
!
hostname {{item.hostname|upper}}
!
interface lo
 ip address 127.0.0.1/8
 ip address {{item.loopback}} secondary
 ipv6 address ::1/128
!
{% for links in item.links %}
interface {{links.spine _ port}}
 description downlink to Leaf Switch {{links.leaf _ switch}}
 ip address {{links.spine _ port _ local _ ip}}/31
!
{% endfor %}
!
router bgp {{item.local _ ASN}}
 bgp bestpath as-path multipath-relax
 max-paths ebgp 16
{% for links in item.links %}
 neighbor {{links.leaf _ port _ remote _ ip}} remote-as {{links.remote _ ASN}}
{% endfor %}}
!
line con 0
 login
line vty 0 871
 exec-timeout 0 0
 login
!
end
```

Figure 10. OcNOS Configuration Template for Leaf “leaf.j2”

```
!
hostname {{item.hostname|upper}}
!
interface lo
 ip address 127.0.0.1/8
 ip address {{item.loopback}} secondary
 ipv6 address ::1/128
!
{% for links in item.links %}
interface {{links.leaf _ port}}
 description uplink to Spine Switch {{links.spine _ switch}}
 ip address {{links.leaf _ port _ local _ ip}}/31
!
{% endfor %}
!
router bgp {{item.local _ ASN}}
 bgp bestpath as-path multipath-relax
 max-paths ebgp 16
 network {{item.loopback}} mask 255.255.255.255
{% for links in item.links %}
 neighbor {{links.spine _ port _ remote _ ip}} remote-as {{links.remote _ ASN}}
{% endfor %}}
!
line con 0
 login
line vty 0 871
 exec-timeout 0 0
 login
!
end
```

After Step 3, the directory structure in the “/etc/ansible” folder will look like the one shown below. Each role has its tasks, template and vars folder populated with respective YAML and Jinja2 files.

Step 4: Create the Playbook File using YAML Format

With all the relevant files in place, one can create the playbook. The playbook named “ebgp-clos.yml” is created in the “/etc/ansible” directory. The host is used as the localhost since the same host is being used as the DHCP Server, Ansible controller, and HTTP Server. The three roles are defines as lists in the YAML file. The content of the file “ebgp-clos.yml” is shown below:

```
---
- name: Build EBGP CLOS IP Topology
  gather _ facts: no
  hosts: localhost

  roles:
  - dhcp
  - leaf
  - spine
```

Step 5: Editing the Hosts File in the Ansible Folder

The hosts file should contain the line below for the "ebgp-clos" playbook template to work.

localhost ansible_connection=local

The final structure in the ansible directory should look like the one shown below:

```
.
├── ansible.cfg
├── ebgp-clos.yml
├── hosts
└── roles
    ├── ansible.cfg
    ├── dhcp
    │   ├── tasks
    │   │   └── main.yml
    │   ├── templates
    │   │   └── dhcp.j2
    │   └── vars
    │       └── main.yml
    ├── leaf
    │   ├── tasks
    │   │   └── main.yml
    │   ├── templates
    │   │   └── leaf.j2
    │   └── vars
    │       └── main.yml
    └── spine
        ├── tasks
        │   └── main.yml
        ├── templates
        │   └── spine.j2
        └── vars
            └── main.yml
```

Step 6: Running the playbook

Before this step, all relevant device licenses should be placed in the "/tmp/licenses" folder of the host. The playbook is run with the command **ansible-playbook ebgp-clos.yml**

After this command, configuration files are pushed to respective folders. The DHCP Server is started with the command **service isc-dhcp-server start.** To start the HTTP Server, change the directory to "/tmp" and issue the command **python –m SimpleHTTPServer 80.**

After Step 6, all devices can be provisioned using ZTP.

Snippet of the dhcpd.conf file generated after running playbook

The complete directory structure is available as a GIT repository at:

https://github.com/IPInfusion/OcNOS/tree/1.2/

Snippet of the dhcpd.conf File Generated after Running the Playbook

The complete directory structure is available at: https://github.com/IPInfusion/OcNOS/tree/1.2/

```
default-lease-time 600;
max-lease-time 7200;
option ocnos-provision-url code 250 = text;
option ocnos-license-url code 251 = text;
option subnet-mask 255.255.255.0;
option broadcast-address 10.10.100.255;
option routers 10.10.100.250;
option default-url "http://10.10.100.250/images/DELL _ S6000 _ ON-OcNOS-1.2.0.199-DC _ MPLS-S0-P0-
installer";

subnet 10.10.100.0 netmask 255.255.255.0 {
range 10.10.100.1 10.10.100.200;
     host Leaf _ 01 _ device {
            hardware ethernet ec:f4:bb:fd:5b:85;
            fixed-address 10.10.100.5;
            option ocnos-license-url "http://10.10.100.250/licenses/leaf01 _ license.bin";
            option ocnos-provision-url "http://10.10.100.250/configs/leaf-01.conf";
     }

     host Leaf _ 02 _ device {
            hardware ethernet ec:f4:bb:fd:5b:86;
            fixed-address 10.10.100.6;
            option ocnos-license-url "http://10.10.100.250/licenses/leaf02 _ license.bin";
            option ocnos-provision-url "http://10.10.100.250/configs/leaf-02.conf";
     }

     host Leaf _ 03 _ device {
            hardware ethernet ec:f4:bb:fd:5b:87;
            fixed-address 10.10.100.7;
            option ocnos-license-url "http://10.10.100.250/licenses/leaf03 _ license.bin";
            option ocnos-provision-url "http://10.10.100.250/configs/leaf-03.conf";
     }

     host Leaf _ 04 _ device {
            hardware ethernet ec:f4:bb:fd:5b:88;
            fixed-address 10.10.100.8;
            option ocnos-license-url "http://10.10.100.250/licenses/leaf04 _ license.bin";
            option ocnos-provision-url "http://10.10.100.250/configs/leaf-04.conf";
     }

     host Spine _ 01 _ device {
            hardware ethernet ec:f4:bb:fd:5b:81;
            fixed-address 10.10.100.1;
            option ocnos-license-url "http://10.10.100.250/licenses/spine01 _ license.bin";
            option ocnos-provision-url "http://10.10.100.250/configs/spine-01.conf";
     }

     host Spine _ 02 _ device {
            hardware ethernet ec:f4:bb:fd:5b:82;
            fixed-address 10.10.100.2;
            option ocnos-license-url "http://10.10.100.250/licenses/spine02 _ license.bin";
            option ocnos-provision-url "http://10.10.100.250/configs/spine-02.conf";
     }
```

```
    host Spine _ 03 _ device {
            hardware ethernet ec:f4:bb:fd:5b:83;
            fixed-address 10.10.100.3;
            option ocnos-license-url "http://10.10.100.250/licenses/spine03 _ license.bin";
            option ocnos-provision-url "http://10.10.100.250/configs/spine-03.conf";
    }

    host Spine _ 04 _ device {
            hardware ethernet ec:f4:bb:fd:5b:84;
            fixed-address 10.10.100.4;
            option ocnos-license-url "http://10.10.100.250/licenses/spine04 _ license.bin";
            option ocnos-provision-url "http://10.10.100.250/configs/spine-04.conf";
    }

}
```

Configuration During Run Time using OcNOS Ansible Module

The OcNOS Ansible module is available for configuring devices during run time. This can be used if there is a common task that needs to be executed on all devices. The common tasks such as configuring or changing logging servers. NTP Servers for all devices can make use of this module, as well. The module can also be used to add device specific configurations based on the host file variables.

Example: Configuring Logging Server for all Devices in the EBGP CLOS Network

Let us take the example of configuring a logging server on all eight devices that were provisioned using ZTP. The logging server's IP address is 172.16.1.1, and is reachable through the management Ethernet interface. This essentially is a two-command set per device. In this case, it is 16 commands since it has to span across eight devices. Using a playbook, Ansible automation can be used to configure all devices with a single command. This helps in the context of a large data center where there are several hundred devices to be configured.

Step 1: Installing the Module

The OcNOS Ansible module is a Python script named "ocnos_congfigs.py," which is provided because the OcNOS Package has to be copied to the host machine that is running Ansible. The module has to be copied to the "/usr/lib/python2.7/dist-packages/ansible/modules/core/system/" location. The module supports "exec_cmds" and "config_cmds."

Step 2: Creating the Inventory File

The hosts file in the ansible directory can be used in this case to create an inventory file. For the sake of simplicity, all devices' usernames and passwords are kept similar. The hosts file has a group named "CLOS" which has the list of all eight devices. The usernames and passwords are placed in the "CLOS:vars" group. The hosts file contents are as shown in the listing below.

HOSTS File Contents

```
[CLOS]
SPINE-01 anisble _ ssh _ host=10.10.100.1
SPINE-02 anisble _ ssh _ host=10.10.100.2
SPINE-03 anisble _ ssh _ host=10.10.100.3
SPINE-04 anisble _ ssh _ host=10.10.100.4
LEAF-01 anisble _ ssh _ host=10.10.100.5
LEAF-02 anisble _ ssh _ host=10.10.100.6
LEAF-03 anisble _ ssh _ host=10.10.100.7
LEAF-04 anisble _ ssh _ host=10.10.100.8

[CLOS:vars]
ansible _ ssh _ user=ocnos
ansible _ ssh _ pass=ocnos
```

Playbook File

Once the Log.yml YAML Playbook file is created, the hosts variable calls the CLOS group in which the devices are listed. The tasks section has the OcNOS Module using the “config_cmds” mode to configure a static route and logging server.

```
---
- hosts: CLOS
  tasks:
  - ocnos_configs:
      config_cmds:
       - “ip route vrf management 172.16.1.1/32 eth0”
       - “logging server 172.16.1.1 7”
```

Step 3: Run the Playbook

Once the hosts file and the playbook are ready, the playbook is run using the command

ansible-playbook Log.yml

All eight devices are now configured for sys logging.

Conclusion

OcNOS Zero Touch Provisioning along with Ansible modules greatly reduces the burden in provisioning the network elements, and increases the efficiency of any large-scale deployment project. With templates already in place, network operators can quickly replace the faulty elements and reduce downtime.

Refer to https://github.com/IPInfusion/OcNOS/tree/1.2/ for sample Ansible code.

Phone: +1 877-MYZEBOS
Email: sales@ipinfusion.com
Web: www.ipinfusion.com

U.S. (Santa Clara), +1 408-400-1912
Japan (Tokyo), +81 03-5259-3771
Korea (Seoul) +82 (2) 3153-5224

India (Bangalore), +91 (80) 6728 7000
China (Shanghai), +86 186 1658-6466
EMEA (Stockholm), +46 8 566 300 00

IP Infusion
An ACCESS Company
(408) 400-3000
www.ipinfusion.com
3965 Freedom Circle, Suite 200
Santa Clara, CA 95054