

Automating Network Infrastructures with Ansible on FreeBSD

Firebird Networks Belgium



Introductions

- Network Consultant and Co-Founder of Firebird Networks
- Points of interest: datacenter networks, service provider architecture, network automation

State of Network Automation

It's 2014 on highway 101 from San Francisco to San Jose, some cars are driving themselves. Around the world there are military aircraft flying around with no pilot, being controlled by remotely from another country. In your data center there is an engineer/admin configuring a switch on a CLI. What's wrong with this picture?

Joe Onisick – Principal Engineer Cisco Systems

Case Confirms



Network Agility

1994

```
Router> enable
Router# configure terminal
Router(config)# enable secret cisco
Router(config)# ip route 0.0.0.0 0.0.0.0 20.2.2.3
Router(config)# interface ethernet0
Router(config-if)# ip address 10.1.1.1 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# interface serial0
Router(config-if)# ip address 20.2.2.2 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# router rip
Router(config-router)# network 10.0.0.0
Router(config-router)# network 20.0.0.0
Router(config-router)# exit
Router(config)# exit
Router# copy run start
```

Terminal Protocol: **Telnet**

2014

```
Router> enable
Router# configure terminal
Router(config)# enable secret cisco
Router(config)# ip route 0.0.0.0 0.0.0.0 20.2.2.3
Router(config)# interface ethernet0
Router(config-if)# ip address 10.1.1.1 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# interface serial0
Router(config-if)# ip address 20.2.2.2 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# router rip
Router(config-router)# network 10.0.0.0
Router(config-router)# network 20.0.0.0
Router(config-router)# exit
Router(config)# exit
Router# copy run start
```

Terminal Protocol: **SSH**

Automation vs. Orchestration

- What is automation?
 - Automation eliminates the necessity of repeatable manual tasks
- What is orchestration?
 - Orchestration is the manner in which automated tasks are grouped together in coordinated workflows

Part I: Introducing Ansible



CHEF™



ANSIBLE



SALTSTACK

“What is Ansible?”

- ***“Ansible is a super-simple automation platform that is agentless and extensible”***
- By simple it means that you don't need any coding knowledge to get started
- Agentless means that you do not require an agent on each device in order to be able to control them (important for vendor-locked network devices)
- Extensible means that it benefits from the open-source community and things will be built for it.

Basic Ansible Architecture

- **Ansible = automation platform**
- Can be installed on every laptop or just a central server.
- Use pip, apt or yum or pkg to install on *nix-based machines
- All automation is performed out of the device that hosts the installation of Ansible (also known as a ***control host***)
- Uses the notion of ***playbooks*** - a set of automation tasks and instructions which are pushed for execution on specific hosts.

Playbooks

- From [ansible.com](https://www.ansible.com): *“Playbooks are Ansible’s configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.”*
- Playbooks are similar to an Ikea instruction manual that breaks the entire process of configuring a router, or a BGP process or whatever else into small little tasks and delegates the interaction with the devices.
- Best feature: human-readable (if you like YAML)
- Check more examples out: <https://github.com/ansible/ansible-examples>

YAML

- Ansible Playbooks are expressed using YAML syntax
- YAML - YAML Ain't Markup Language
- YAML uses a small amount of separators - indentation gives structure, colons separate keys, and dashes create bullet lists
- Every YAML file must start with `- - -` and end with `. . .`
- Members of a list are marked with a dash (`- Apple`)
- Dictionary terms are a pair separated by a colon - `key: value` (yes, the space between the two is necessary)
- More syntax documentation: <http://docs.ansible.com/ansible/latest/YAMLSyntax.html>

Templates

- Ansible uses the Python-based Jinia2 templating language
- A template is a standard configuration without its variables
- Internally based on Unicode, it is inspired by Django's templating system
- Jinja supports a few control structures like if and for-loops making it easy to shorten your templates
- A good starter for understanding Jinia templates:
<https://realpython.com/blog/python/primer-on-jinja-templating/>

From Configuration to Jinjia Template

```
CRS# conf t
CRS(config)# router bgp 65501
CRS(config-bgp)# neighbor 10.10.10.2
CRS(config-bgp-nbr)# remote-as 2000
CRS(config-bgp-nbr)# password <MD5 password>
CRS(config-bgp-nbr)# ebgp-multihop 2
CRS(config-bgp-nbr)# update-source loopback0
CRS(config-bgp-nbr)# address-family ipv4 unicast
CRS(config-bgp-nbr-af)# route-policy xxxxx in
CRS(config-bgp-nbr-af)# exit
```

```
!
router bgp {{ as_number }}
!
neighbor {{ ip_neighbor }}
remote-as {{ as_number }}
password {{ md5_password }}
ebgp-multihop {{ mhop_value }}
update-source {{ update_if }}
address-family ipv4 unicast
!
```

From Configuration to Jinjia Template

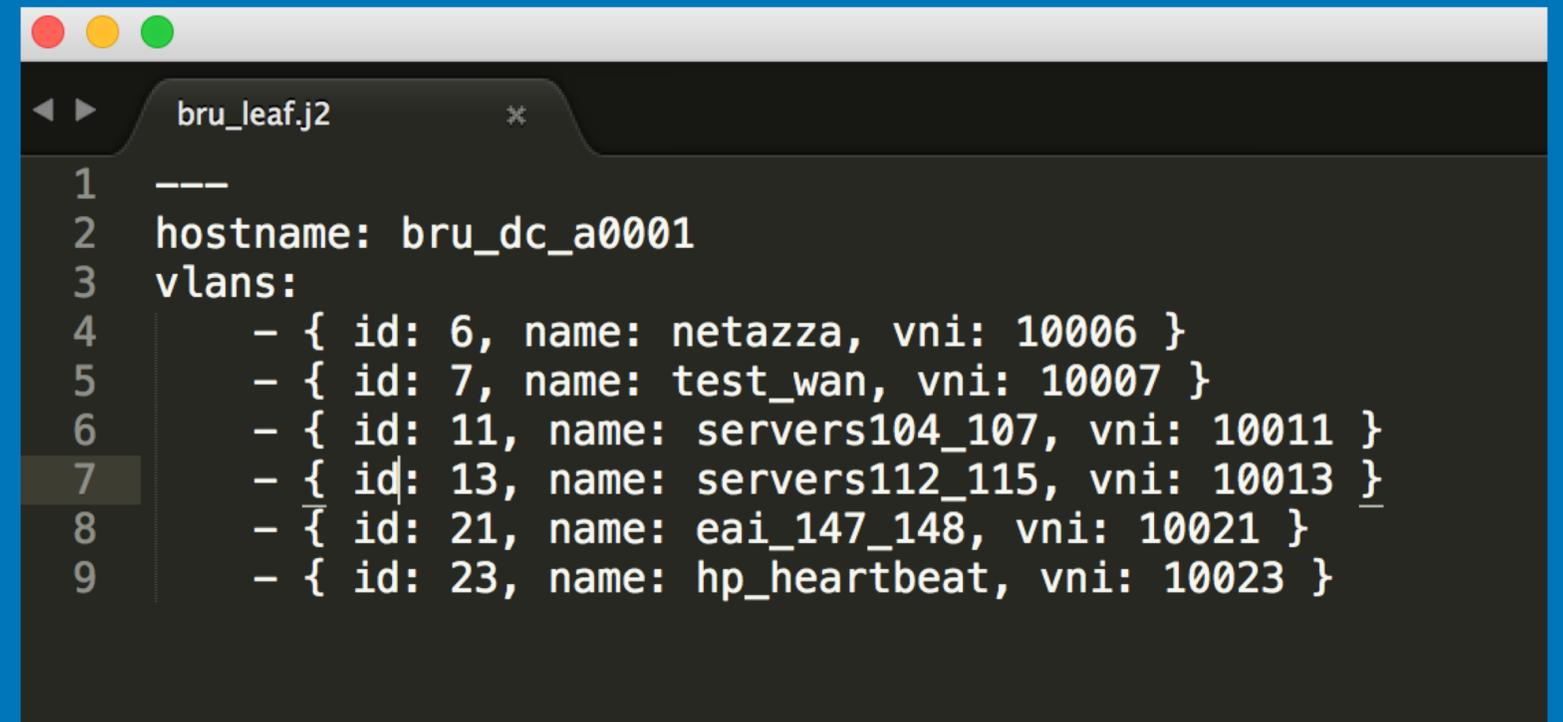
```
vlan 6
  name NETAZZA
  vn-segment 10006
vlan 7
  name Test_WAN
  vn-segment 10007
vlan 11
  name Vlan11/104-107
  vn-segment 10011
vlan 13
  name OBM/112-115
  vn-segment 10013
vlan 21
  name eai_dev_147-148
  vn-segment 10021
vlan 23
  name HP-Heartbeat
  vn-segment 10023
vlan 24
  name SAP/156-159
  vn-segment 10024
vlan 25
  name eai_prod_160-163
  vn-segment 10025
```

contents of leaf_vni.j2

```
!
!
{ % for vlan in vlans % }
vlan {{ vlan.id }}
  name {{ vlan.name }}
  vni {{ vni.id }}
{% endfor %}
!
```

Variables and Variable Files

- Double curly brackets = variables
- Variables are not stored in the templates
- Variables are stored in variable files
- Example variable file



```
bru_leaf.j2
1 ---
2 hostname: bru_dc_a0001
3 vlans:
4   - { id: 6, name: netazza, vni: 10006 }
5   - { id: 7, name: test_wan, vni: 10007 }
6   - { id: 11, name: servers104_107, vni: 10011 }
7   - { id: 13, name: servers112_115, vni: 10013 }
8   - { id: 21, name: eai_147_148, vni: 10021 }
9   - { id: 23, name: hp_heartbeat, vni: 10023 }
```

Basic Working Ansible Playbook

- Premises: Build the most basic playbook that can check time on 2 devices
- Step 1: Build the “Playbook” file

```
auxesis@ubuntu: ~/Documents/Scripts/ansible_polaris
---
- name: nx-os test
  hosts: "test-leafs"
  gather_facts: false
  connection: local

vars:
  cli:
    host: "{{ inventory_hostname }}"
    username:
    password:
    transport: cli

tasks:
  - name: show clock
    nxos_command:
      commands: show clock
      provider: "{{ cli }}"

    register: output

  - name: show output
    debug:
      var: output.stdout
```

Basic Working Ansible Playbook

- Make sure your inventory is up to date
- And then just run the playbook by typing `ansible-playbook file_name.yml`

```
auxesis@ubuntu: ~/Documents/Scripts/an
[test-leafs]
10.1.1.1
10.1.1.2
~
~
~
```

```
auxesis@ubuntu:~/Documents/Scripts/ansible_polaris$ ansible-playbook nxos_test.yml
[DEPRECATION WARNING]: [defaults]hostfile option, The key is misleading as it can also be a
version 2.8. Deprecation warnings can be disabled by setting deprecation_warnings=False in

PLAY [nx-os test] *****

TASK [show clock] *****
ok: [10.1.128.201]
ok: [10.1.128.200]

TASK [show output] *****
ok: [10.1.128.200] => {
  "output.stdout": [
    "Time source is NTP\n03:12:12.314 CET Fri Jan 26 2018"
  ]
}
ok: [10.1.128.201] => {
  "output.stdout": [
    "Time source is NTP\n15:12:52.052 CET Fri Jan 26 2018"
  ]
}

PLAY RECAP *****
10.1.128.200      : ok=2    changed=0    unreachable=0    failed=0
10.1.128.201      : ok=2    changed=0    unreachable=0    failed=0
```

Summary of a Simple Ansible Playbook

- Every Ansible Playbook will be written in YAML, has a specific necessity for beginning with `- - -` and ending with `. . .`
- It needs a method to connect to its devices which are stored in an inventory file
- When templates need to be applied, a template file is used - Jinjia2 is the preferred templating language
- The values that are introduced for each of the devices for each of the templates are stored in a variable file which is also a `.j2` file

Part II: FreeBSD for Network Engineers

- Setting up a new VM with ansible & python in a matter of minutes
- Executing playbooks and working around known caveats

Getting Ansible on Your FreeBSD Machine

- Setting up Ansible on FreeBSD means setting up your control machine. You can do this in a jail, you can have it running as a VM somewhere, or as a bare-metal machine.
- The obvious requirement is that it needs to be able to access the hosts that it should “manage” (more a network problem than a server problem)
- **Tip:** Make sure your username is in the sudoers group (especially if you provision a new machine)

Preparing your machine

- Have OpenSSH up and running on your machine

```
$ service -e | grep sshd
```

```
/etc/rc.d/sshd
```

- If it's not running, make sure you add & activate it at boot:

```
# echo 'sshd_enable="YES"' >> /etc/rc.conf
```

```
# service sshd start
```

- Install ansible & python in one command:

```
# pkg install ansible python
```

Quick Checks

```
$ freebsd-version
11.1-RELEASE
$ service -e | grep sshd
/etc/rc.d/sshd
$
$
$ python -V
Python 2.7.15
$
```

The `ansible_python_interpreter` caveat

- FreeBSD (OpenBSD too for that matter) doesn't come with `/usr/bin/python`
- The ports don't install a "python" package, actually: they install a version of python, named after the version

The ansible_python_interpreter caveat

```
TASK [Gathering Facts] *****
fatal: [BRA-R8-DC-S0002]: FAILED! => {"changed": false, "module_stderr": "/bin/sh: /usr/bin/python: not found\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 127}
fatal: [BRA-R8-DC-A0001]: FAILED! => {"changed": false, "module_stderr": "/bin/sh: /usr/bin/python: not found\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 127}
fatal: [CEN-HQA-DC-S0002]: FAILED! => {"changed": false, "module_stderr": "/bin/sh: /usr/bin/python: not found\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 127}
fatal: [CEN-HQA-DC-S0001]: FAILED! => {"changed": false, "module_stderr": "/bin/sh: /usr/bin/python: not found\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 127}
fatal: [BRA-R8-DC-A0002]: FAILED! => {"changed": false, "module_stderr": "/bin/sh: /usr/bin/python: not found\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 127}
fatal: [BRA-R8-DC-A0013]: FAILED! => {"changed": false, "module_stderr": "/bin/sh: /usr/bin/python: not found\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 127}
fatal: [BRA-R8-DC-S0001]: FAILED! => {"changed": false, "module_stderr": "/bin/sh: /usr/bin/python: not found\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 127}
fatal: [BRA-R8-DC-A0015]: FAILED! => {"changed": false, "module_stderr": "/bin/sh: /usr/bin/python: not found\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 127}
fatal: [BRA-R8-DC-A0014]: FAILED! => {"changed": false, "module_stderr": "/bin/sh: /usr/bin/python: not found\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 127}
fatal: [BRA-R8-DC-A0016]: FAILED! => {"changed": false, "module_stderr": "/bin/sh: /usr/bin/python: not found\n", "module_stdout": "", "msg": "MODULE FAILURE", "rc": 127}
```

The `ansible_python_interpreter` caveat

- One solution is to just add it to your hosts files and pass it as a variable.
- This also allows you to use different versions of Python depending on the scripts that you want to use.

```
$ cat hosts | grep -A 2 -B 2 ansible_python  
  
[all:vars]  
#ansible_python_interpreter='/usr/local/bin/python2.7'
```

Examples of Playbooks and Network Applications

“To err is human, to apply that error to 1000 servers at once is DevOps.”

-Unknown

Questions?

