ANSIBLE

June 2016. Version 1.1

Khelil Sator

WHAT IS ANSIBLE?

WHAT IS ANSIBLE?

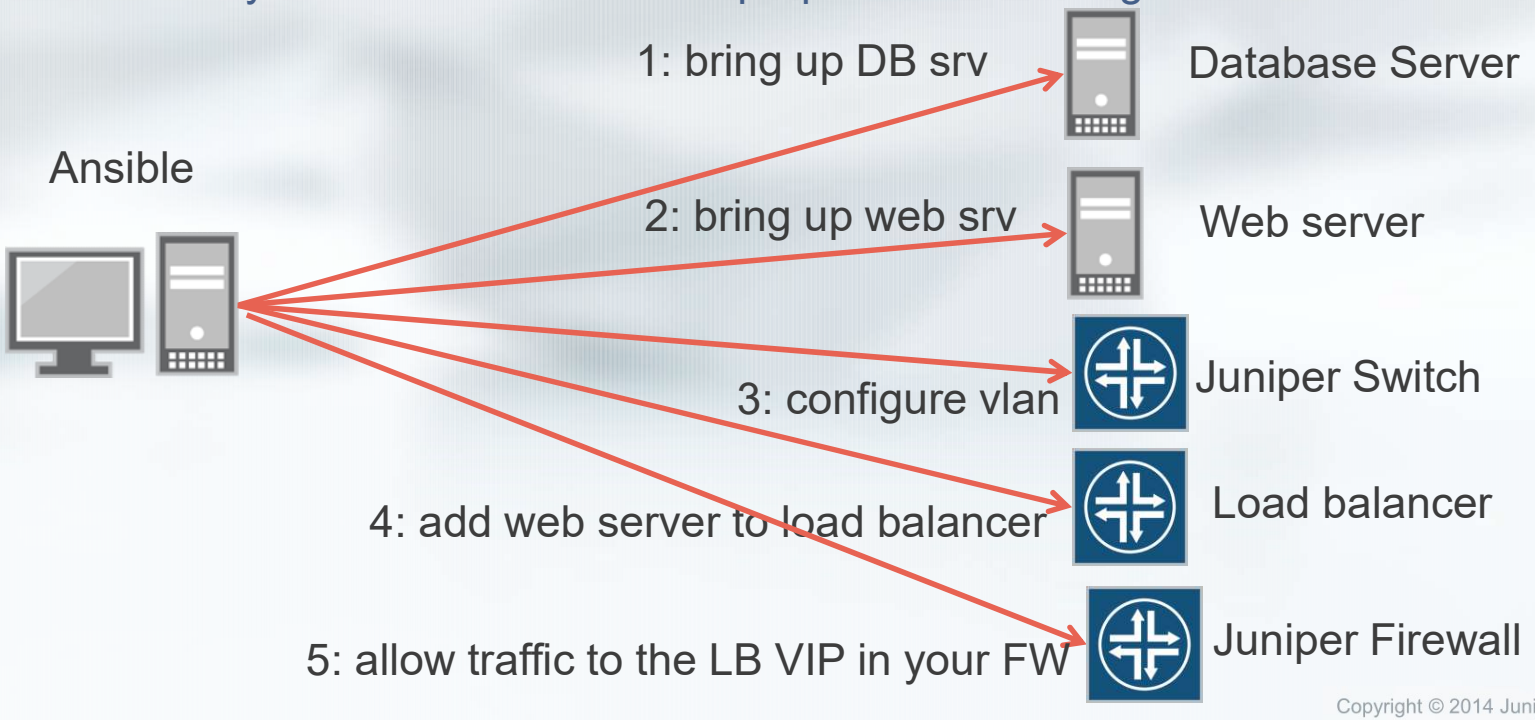
- A science fiction literature reference in the 60's ☺
- A radically simple IT automation platform
  - Ansible's goals are similar to other tools such as Puppet, Chef and Salt
  - Created by Michael DeHaan
  - Initial release in 2012
  - Current release is 2.1 (June 2016)
  - Acquired by Red Hat in 2015
    - Michael DeHaan left
  - It is very popular!
  - They hired Peter Sprygada

WHAT CAN WE DO WITH ANSIBLE?

- This is an IT automation tool
- Initially for servers and applications
- Ansible is designed for performing actions (tasks) on multiple servers.
- Ansible uses cases are:
  - Application deployment
  - Configuration management
  - Template management
  - Orchestration
- Some nice network automation capabilities added recently.

ORCHESTRATION WITH ANSIBLE

- You can manipulate an entire application stacks made up multiple tiers.
  - you need to bring up the database before bringing up the web servers
  - you need to take web servers out of the load balancer one at a time in order to upgrade them without downtime.
  - you need to ensure that your networks have their proper VLANs configured



WHY SOME PEOPLES LOVE ANSIBLE

- Agentless
- Easy to use. Human readable automation. Data, not code. Configuration in Yaml. Use Jinja2 templates.
- You can use a large library (about 500 modules) with some level of abstraction.
- Push based. No Master server. Running from everywhere (laptop).
- Networks are integral parts of IT enterprises, so Ansible has now some modules for network automation as well!
- Nice orchestration capabilities
- Has loops and conditionals
- Idempotent (but not all the modules)
- Written in Python (but no need to understand Python).
- You can add your own modules to extend Ansible capabilities.

EASY TO USE

- Get productive quickly
  - I think it is easier for regular engineers to use Ansible versus Puppet and chef
- Easy-to-Read Syntax:
  - Human readable automation. Playbooks in Yaml.
  - Its data, not code! No special coding skills needed (you still need to learn Ansible)
  - Ansible uses the Yaml and Jinja2, so you'll need to learn some Yaml and Jinja2 to use Ansible, but both technologies are easy to pick up.
- Level of abstraction
  - Declarative approach. You describe the desirated state, indicating "what", not "how".
    - "these hosts are dbservers" or "these hosts are webservers".

IDEMPOTENCY

- Some module are idempotent
  - Changes should only be applied when they need to be applied, and that it is better to describe the desired state of a system than the process of how to get to that state.
  - With idempotent modules, Ansible will first check the state of the device and will make a change only if the current state does not match the required end state".

HOW IT WORKS?

AGENT LESS

- Ansible use SSH (quite native)
  - There is no agent to install on the remote hosts
  - But some Ansible modules have requirements on the endpoints and on the Ansible server (this will be discussed later on in this presentation)
- Agent less solutions advantages:
  - when a box is not being managed by Ansible, nothing extra is running. Saves resources (memory and cpu, per vm) on endpoints.
  - more secured
  - No additional open port
  - Eliminate the need to deploy and manage agents (easier)
  - Broader application: some network devices are “closed” in a such a way 3rd party software agents cannot be loaded onto the device.
- Agent based solutions can be very nice as well (because the agent can collect many data and watch states/events on endpoints)
  - Slaltstack, Apstra, Puppet, chef.

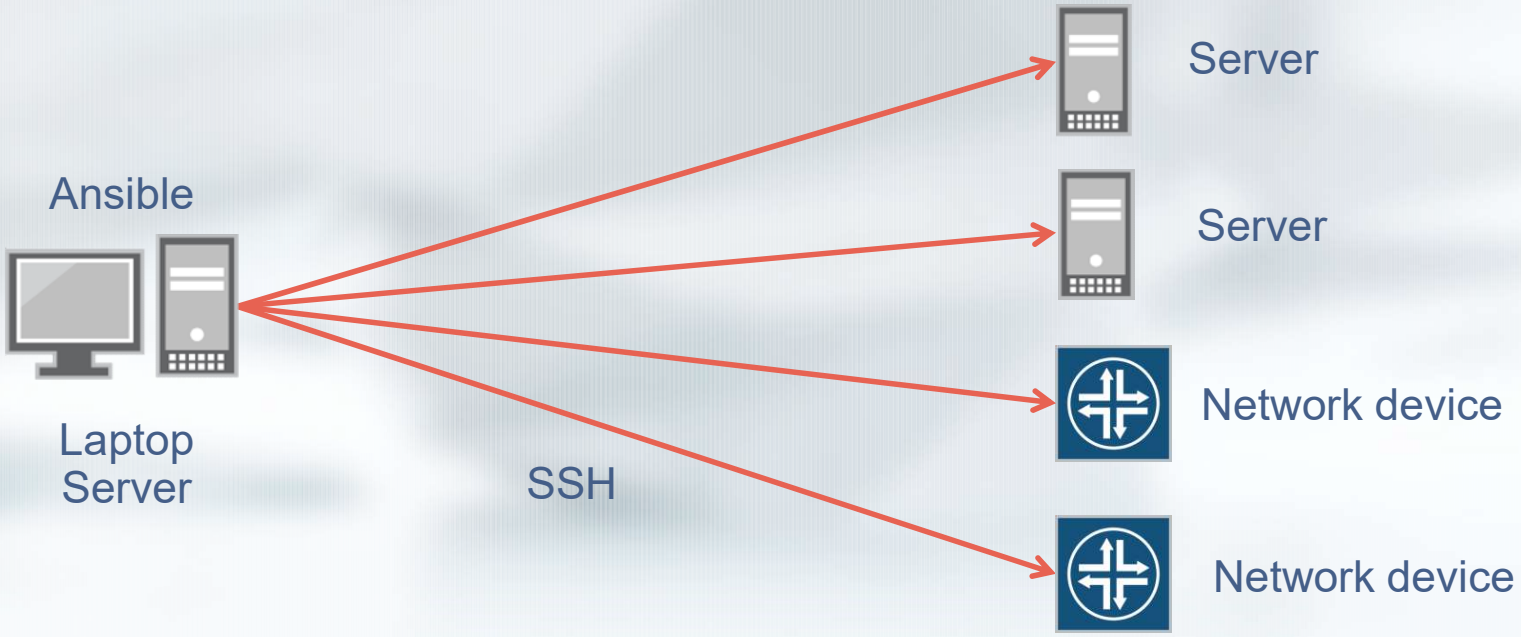
PUSH BASED

- Ansible is push based
  - It is not pull based: where an agent installed on servers periodically check with a central service (master server) and pull configuration information form the service.
- As soon as you run the ansible-playbook command (to execute a playbook), Ansible connects to the remote servers/devices and does its thing.
- The push-based approach has a significant advantage:
  - You control when the changes happen to the servers. You don’t need to wait around for a timer to expire.

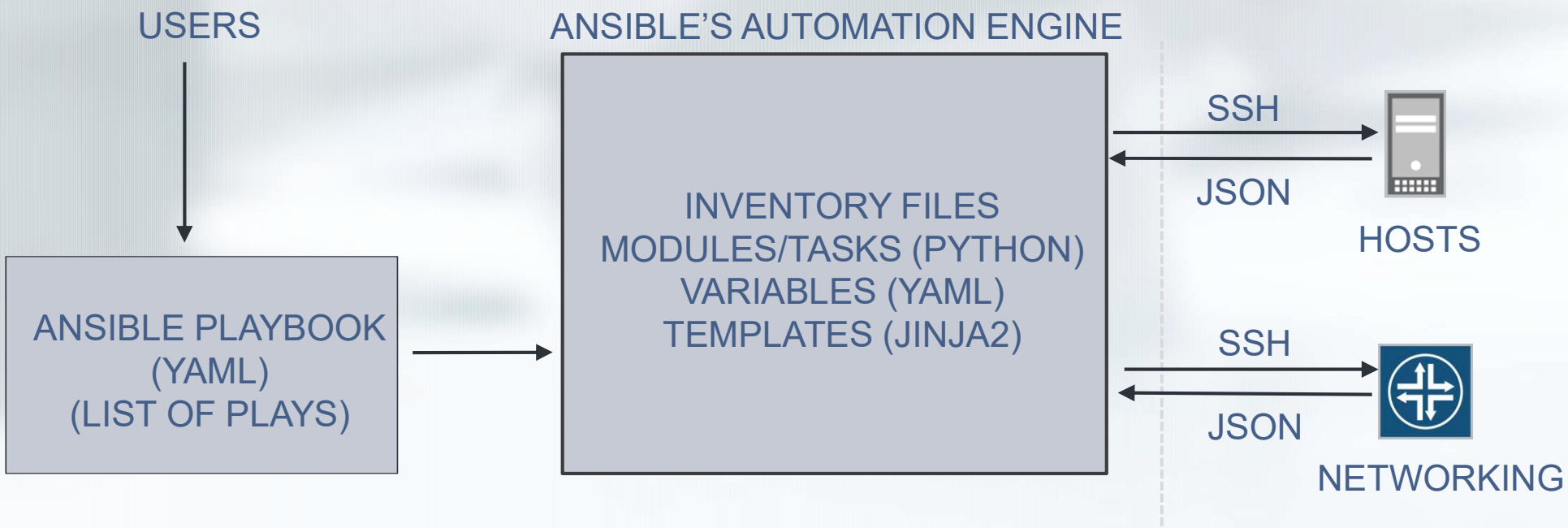
HOW ANSIBLE WORKS?

- Standard mode uses SSH to transport python modules to the remote boxes, and then execute these python modules on remote boxes, and then eliminate them from remote hosts.
  - Python modules execution will change the device configurations.
  - Requires Python installed on remotes machines.
    - You can use the Ansible module ping to check this. It returns pong on success.
    - Some Ansible modules require also some python libraries installed on the remote hosts.
- API mode: Modules run on Ansible server. Ansible then uses a channel/API (Netconf ....) to change the device configurations.
  - “connection: local” in the playbook means the actions/tasks will be run local on the server running Ansible and not on the target host.

HOW ANSIBLE WORKS?



HOW ANSIBLE WORKS?



ANSIBLE AND PYTHON

- Ansible is written in Python.
  - All Ansible core modules are written in Python
- No need to know Python to use Ansible
  - You can use ansible without understanding the python modules
  - Unless:
    - you want to write your own modules to extend Ansible capabilities.
    - You want to understand the source code
- Ansible 2.1 uses Python 2.7
  - Python 3 is not yet compatible with Ansible
- When you install Ansible, it installs automatically some Python modules:
  - paramiko (Ansible manages machines over SSH)
  - PyYAML (you interact with Ansible in yaml)
  - Jinja2 (Ansible has builtin templating capabilities)
  - httplib2 (a comprehensive HTTP client library)

ANSIBLE AND PYTHON

- If you already know Python, Jinja2, YAML, PyEZ, JSON, Ansible is an excellent next step! It uses all of them, but with some level of abstraction.
  - All Ansible core modules are written in Python
  - Ansible use PyEZ to interact with Junos devices
  - Jinja2 is used to build documents based on templates
  - YAML is used to defines variables values
  - YAML is used to write Ansible playbooks (Ansible scripts)
  - Ansible expects modules to output JSON

ANSIBLE KEY COMPONENTS

ANSIBLE KEY COMPONENTS

- Inventory
- Variables
- Tasks/modules
- Plays and Playbooks

INVENTORY

INVENTORY FILES

```
[spine]
qfx10002-01
qfx10002-02

[leaf]
qfx5100-01
qfx5100-02
qfx5100-03
qfx5100-04
```

- text file with your inventory
  - you can have many inventory files
  - The default ansible 'hosts' file lives in /etc/ansible/hosts
- Comments begin with the '#' character
- Blank lines are ignored
- You can enter hostnames or ip addresses
- Groups of hosts (per device type, per location, ...)
- A hostname/ip can be a member of multiple groups
- Example https://github.com/dgjnpr/ansible-template-for-junos/blob/master/hosts

```
[all:children]
spine
leaf

[spine]
qfx10002-01    junos_host=192.168.0.1
qfx10002-02    junos_host=192.168.0.2

[leaf:children]
rack01
rack02

[rack01]
qfx5100-01     junos_host=192.168.0.3
qfx5100-02     junos_host=192.168.0.4

[qfx10000]
qfx10002-01
qfx10002-02

[qfx5100]
qfx5100-01
qfx5100-02

[siteA]
qfx10002-01
qfx5100-01
```

- Simple but very powerful and important
- One device can be part of multiple groups
- Can have a hierarchy of groups
- Can groups devices by type/location/roles etc …
- qfx5100-01 is part of groups:
  - All
  - leaf
  - rack01
  - qfx5100
  - siteA
- Inventory files can take advantage enumerations
  - QFX5100-[1:13]
  - host[a:z]
  - 172.16.0.[1:254]
- Inventory files can take advantage of variables

VARIABLES

VARIABLE FILES

```
global:
    root_hash: $1$ZUlES4dp$OUwWo1g7cLoV/aMWpHUnC/
    time_zone: America/Los_Angeles
    name_servers:
    - 192.168.5.68
    - 192.168.60.131
    ntp_servers:
    - 172.17.28.5
    snmp:
        location: "Site 1"
        contact: John Doe
        polling:
        - community: public
    routes:
        default: 10.94.194.254
```

- Variable definitions is very flexible
- A variable file is a YAML file
- Variable files can live in many places
- Variables can be defined in playbooks,
- Variables can be defined in the inventory files.
- Variable files can also be referred in a playbook (with the include_vars module)
- Accessible from templates and playbooks
- The module setup is automatically called by playbooks to gather facts (variables discovered, not set by the user) about remote hosts (setup module is not valid for junos devices so use gather_facts: no in your play)

VARIABLES ASSOCIATE WITH GROUPS AND HOSTS

```
|-- group_vars
|   |-- leaf01
|   |   |-- file1.yaml
|   |   `-- file2.yml
|   |-- leaf02
|   |   |-- file1.yaml
|   |   `-- file2.yml
|   |-- spine
|   |   |-- file1.yaml
|   |   `-- file2.yml
|   |-- all.yaml
|   |-- qfx10000.yaml
|   `-- qfx5100.yaml
|-- host_vars
|   |-- qfx10002-01
|   |   |-- file1.yml
|   |   `-- file2.yaml
|   |-- qfx5100-01
|   |   |-- file1.yml
|   |   `-- file2.yaml
[…]
|   |-- qfx5100-04
|   |   |-- file1.yml
|   |   `-- file2.yaml
|   `-- qfx5100-05.yml
|   `-- qfx5100-06.yml
```

- Variable files can live in many places
  - host_vars folder has the variables per host
  - group_vars folder has the variables per group (qfx, spines, site1, all, …)
  - vars subfolders of roles can be used to store variables for roles
- One or multiple variable files per
  - Host
  - Group
- Very flexible

GENERATE C

host_vars/qfx5100-01.yaml

```
ospf:
     interfaces:
          ge-0/0/0:
               ip: 192.168.0.1
               mask: 24
          ge-0/0/1:
               ip: 192.168.1.1
               mask: 24
Host:
     mgmt:
          ip: 10.10.10.1
          mask: 24
```

group_vars/qfx5100.yaml

```
Mgmt_interface: em0
```

group_vars/all.yaml

```
Global:
     root_hash: $1$dvewqcsarwrgvwr
Host:
     mgmt:
          mask: 24
```

```
system {
     host-name {{ inventory_hostname }};
      root-authentication {
          encrypted-password "{{ global.roo
      }
}
interfaces {
   {{ mgmt_interface }} {
      unit 0 {
          family inet {
               address {{ host.mgmt.ip }}/{
          }
      }
   }
{% for interface in ospf.interfaces %}
     {{ interface }} {
        unit 0 {
          family inet {
               address {{interface.ip }}/{{
          }
        }
     }
{% endfor %}
}

protocols {
     ospf {
          area 0.0.0.0 {
{% for interface in ospf.interfaces %}
             interface {{ interface }}
{% endfor %}
}
```

Final version

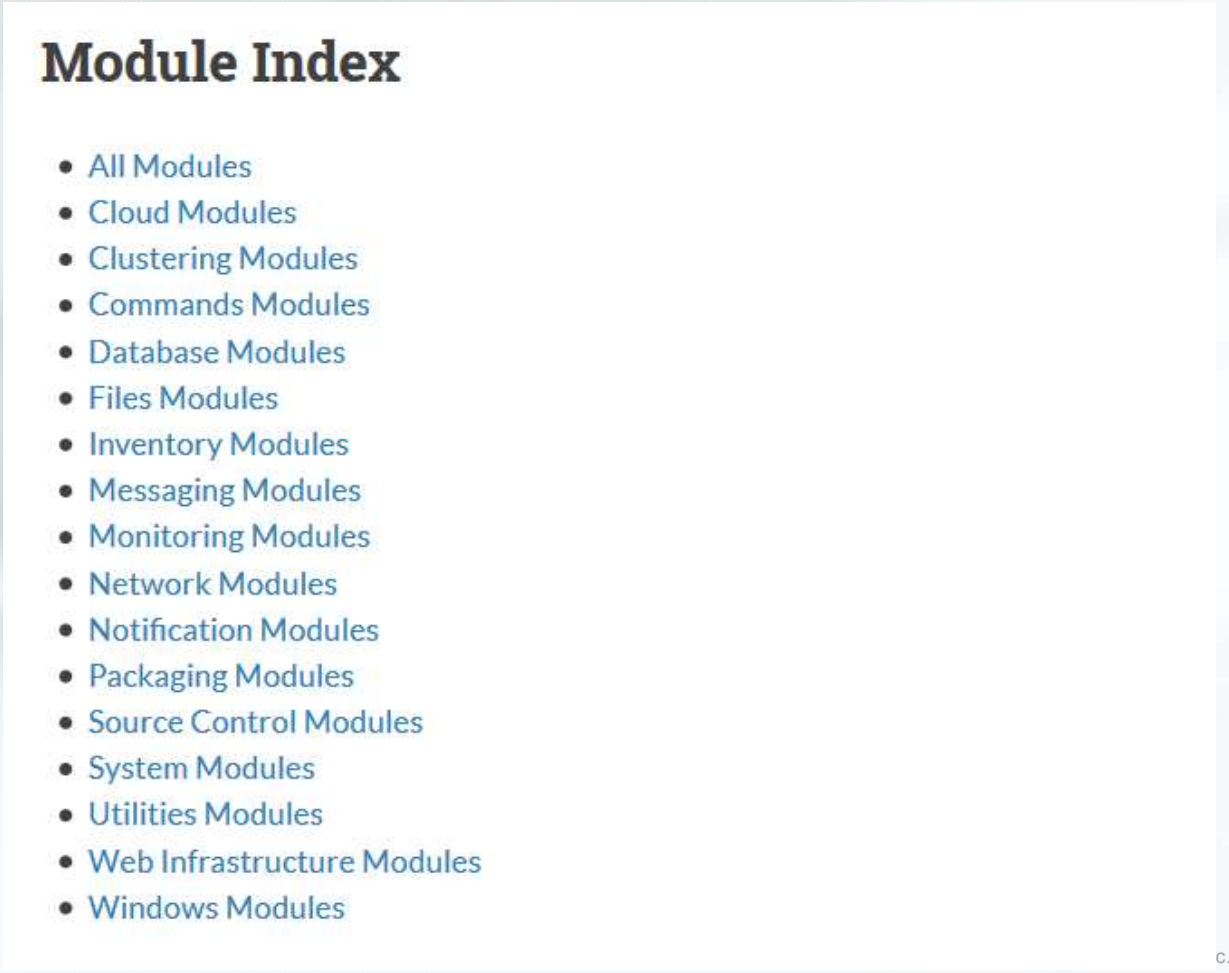
```
system {
     host-name qfx5100-01;
      root-authentication {
          encrypted-password "$1$dvewqcsarwrgvwr";
      }
}
interfaces {
   em0 {
      unit 0 {
          family inet {
               address 10.10.10.1/24;
          }
      }
   }
   ge-0/0/0{
        unit 0 {
          family inet {
               address 192.168.0.1/24
          }
        }
     }
   ge-0/0/1{
        unit 0 {
          family inet {
               address 192.168.1.1/24
          }
        }
    }
}

protocols {
     ospf {
          area 0.0.0.0 {
               interface ge-0/0/0
                interface ge-0/0/1
}
```

MODULES

MODULES

- Lot of modules available (Batteries included: Ansible ships with about 500 modules)



Module Index

- All Modules
- Cloud Modules
- Clustering Modules
- Commands Modules
- Database Modules
- Files Modules
- Inventory Modules
- Messaging Modules
- Monitoring Modules
- Network Modules
- Notification Modules
- Packaging Modules
- Source Control Modules
- System Modules
- Utilities Modules
- Web Infrastructure Modules
- Windows Modules

MODULES

- http://docs.ansible.com/ansible/list_of_all_modules.html
- Some nice core modules:
  - template: Generates a file from a template and copies it to the hosts.
  - Assemble: Assembles a configuration file from fragments.
  - file: creates/removes files and directories.
  - copy: Copies files to remote hosts.
  - pip: installs python packages (pip is required on remote hosts)
  - apt: manages packages (using apt)
  - yum: Manages packages (with yum)
  - git: deploys files from a git repo (git is required on remote hosts)
  - service: control services on remote hosts
  - pause: Pauses playbook execution
  - uri: interacts with web services

Ansible librairies to interact with Junos

- There are two Ansible librairies to interact with Junos.
  - An Ansible library for Junos built by Juniper (hosted on the Ansible Galaxy website)
  - An Ansible library for Junos built by Ansible (since Ansible 2.1)
- You can use both of the Ansible libraries for Junos

SOME ANSIBLE USE CASES FOR JUNOS DEVICES

- Generate configuration
- Deploy configuration
- Rollback configuration
- Retrieve configuration and facts
- Upgrade devices
- Audit the devices

Ansible modules for Junos built by Juniper

- Hosted on the Ansible Galaxy website (https://galaxy.ansible.com/Juniper/junos/)
  - Version: 1.3.1 (June 2016)
  - Documentation: http://junos-ansible-modules.readthedocs.io/en/1.3.1/
  - Installation: Execute the “ansible-galaxy install Juniper.junos” command to download it
  - Source code https://github.com/Juniper/ansible-junos-stdlib
  - These modules use PyEZ
- Overview of modules:
  - junos_cli - Execute CLI on device and save the output locally
  - junos_commit - Execute commit on device
  - junos_get_config - Retrieve configuration of device
  - junos_get_facts - Retrieve facts for a device running Junos OS.
  - junos_install_config - Load a configuration file or snippet onto a device running Junos OS.
  - junos_install_os - Install a Junos OS image.
  - junos_rollback - Rollback configuration of device
  - junos_rpc - run given rpc
  - junos_shutdown - Shut down or reboot a device running Junos OS.
  - junos_srx_cluster - Create an srx chassis cluster for cluster capable srx running Junos OS.
  - junos_zeroize - Erase all data, including configuration and log files, on a device running Junos OS.

Ansible core modules for Junos built by Ansible:

- Documentation: http://docs.ansible.com/ansible/list_of_network_modules.html
- Installation: They are core modules.
  - They ship with ansible itself (from Ansible 2.1).
  - Ansible 2.1 or above is required.
- Source code: https://github.com/ansible/ansible-modules-core/tree/devel/network/junos
- Modules overview :
  - junos_command - Execute arbitrary commands on a remote device running Junos
  - junos_config - Manage configuration on remote devices running Junos
  - junos_facts - Collect facts from remote device running Junos
  - junos_netconf - Configures the Junos Netconf system service
  - junos_package - Installs packages on remote devices running Junos
  - junos_template - Manage configuration on remote devices running Junos
- These modules use PyEZ (except junos_netconf)
- Some are idempotent (junos_config, junos_template, junos_netconf, junos_package)

TASKS

- a task is an Ansible module written in python with some arguments http://docs.ansible.com/ansible/list_of_all_modules.html
- each task has a name.
- The task's name is optional, but recommended
  - for troubleshooting (Ansible print out the name of a task when it runs)
  - Names are useful when somebody else is trying to understand your playbook (including yourself in few months)
  - you can reference the name of a task with the ansible-playbook conmmand and the --start-at-task <task name> option to start the playbook at the task matching this name
- Every task must contain a key with the name of a module and a value with the arguments to that module.

EXAMPLES OF TASKS WITH YUM MODULE

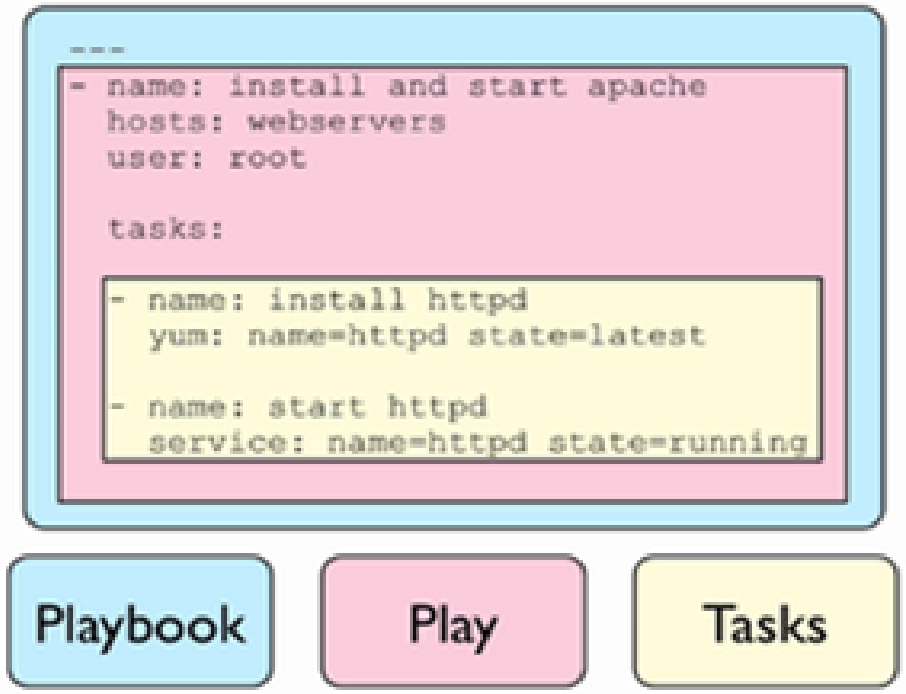
```
- name: install the latest version of Apache
  yum: name=httpd state=latest

- name: remove the Apache package
  yum: name=httpd state=absent

- name: install one specific version of Apache
  yum: name=httpd-2.2.29-1.4.amzn1 state=present
```

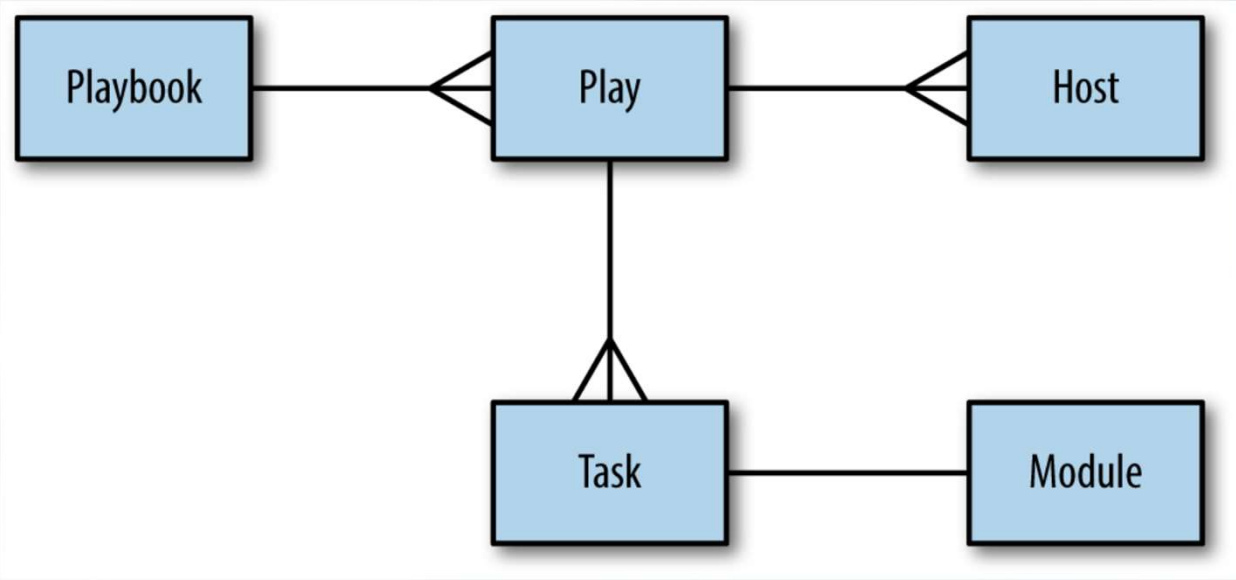
PLAYBOOK

A Playbook



ANATOMY OF A PLAYBOOK

- Relationship between playbooks, plays, hosts, tasks, and modules



PLAYBOOK

- Ansible scripts are called playbooks.
- Playbooks are written in YAML (easy for humans to read and write)
- a playbook is a YAML list of plays.
- a play has a list of tasks.
- it is a list of dictionaries
  - a playbook is a YAML list of plays
  - sometimes there is a single play
  - a play maps a selection of hosts to a list of tasks.
- playbook runs top to bottom
- you execute a playbook with ansible-playbook command
  - you can use the option -i to specify inventory host file (default=/etc/ansible/hosts)

PLAYS

- each play has a name (key/value pair).
  - the name is displayed when the playbook is run.
- a play maps a selection of hosts to a list of tasks.
- a play has a list of tasks.
  - a task is a module written in python
- Every play must contain:
  - A set of hosts
  - A list of tasks to be executed on those hosts
- Tasks are executed in order, one at a time, against the selection of hosts, before moving on to the next task.
- Tasks can be
  - Merge files
  - Render Jinja2 Template
  - Push configuration on Junos
  - …
- a play can map a group of hosts to roles.
  - access-switch
  - Spines
  - underlay-ebgp
  - common
  - ...

PLAYBOOK

```
---
- name: Get Junos facts
  hosts: leaves
  roles:
   - Juniper.junos
  connection: local
  gather_facts: no

  vars_prompt:
   - name: DEVICE_PASSWORD
     prompt: Device password
     private: yes

  tasks:

  - name: Retrieve information from devices running Junos
    junos_get_facts:
     host={{ inventory_hostname }}
     user=pytraining
     passwd={{ DEVICE_PASSWORD }}
     savedir=inventory
    register: junos

  - name: Print some facts when devices are not running junos 12.3R11.2
    debug: msg="device {{junos["facts"]["hostname"]}} runs version {{junos.facts.version}}"
    when: junos.facts.version != "12.3R11.2"
```

This playbook has one play (“Get Junos facts”).

This play executes two tasks on all the hosts from the group “leaves”.

The first task execute the module “junos_get_facts” with some arguments.

The second task is a conditional task (“when”) and execute the module “debug”.

ANSIBLE JARGON

ANSIBLE JARGON

- Playbook: Ansible scripts. Written in Yaml. A list of Plays.
- Plays: contains a set of hosts, and a list of tasks to be executed on those hosts
- Inventory file: your hosts and groups. Can be dynamic
- Variables: there are various options to define variables.
- Task: a Python/Ansible module with some arguments.
- Handlers: a triggered task
- Server facts: Like puppet. Automatically gather by setup module.
- Roles: a reusable automation content you can assign to hosts
- Galaxy: repository for Ansible roles like the ones for Junos built by Juniper.

ROLE

```
---
- name: Create and apply configuration for Leaves QFX / L2
  hosts: leaf-qfx-l2
  connection: local
  gather_facts: no
  roles:
    - common
    - underlay-ebgp
    - overlay-evpn-qfx-l2
    - overlay-evpn-access
```

- Roles are called inside a playbook
- Very useful to reuse the same automation content across playbook
  - Tasks
  - Templates
  - variables
- You can use the “ansible-galaxy init” command to create the skeleton's role

ANSIBLE DEMO

GIT CLONE

- The project https://github.com/ksator/ansible-training-for-junos has many ready to use Ansible playbooks to interact with Junos devices
- Read the instructions in the readme file:
  - https://github.com/ksator/ansible-training-for-junos/blob/master/README.md
- To get the scripts in your laptop, clone it:
  - git clone https://github.com/ksator/ansible-training-for-junos.git
- Playbooks:
  - All playbooks are named pb.*.yml
  - You will find them in different directories. Each directory has a readme file as well.
  - use the ansible-playbook command to execute them

OTHER REPOSITORIES

- Some others nice repositories regarding Ansible and Juniper:
  - https://github.com/JNPRAutomate/ansible-junos-examples
  - https://github.com/dgjnpr/ansible-template-for-junos
  - https://github.com/JNPRAutomate/ansible-junos-evpn-vxlan
  - https://github.com/JNPRAutomate/ansible-demo-ip-fabric


JUNIPer
NETWORKS
THANK YOU!