ANSIBLE

Introduction to Network Automation

Ganesh Nalawade

redhat

Ageda

- Ansible overview
- Ansible in Linux vs Network device
- Ansible Network connection types
- Network modules overview
- Demo
- Contributing to Ansible

Who am I

Ganesh Nalawade

* Principal Software Engineer at Ansible by Red Hat

* Work primarily as upstream developer in Ansible Networking

* Worked extensively on Network management plane developing software for on/box automation and programmability infra.

* Co Organiser for Ansible meetup group in Pune

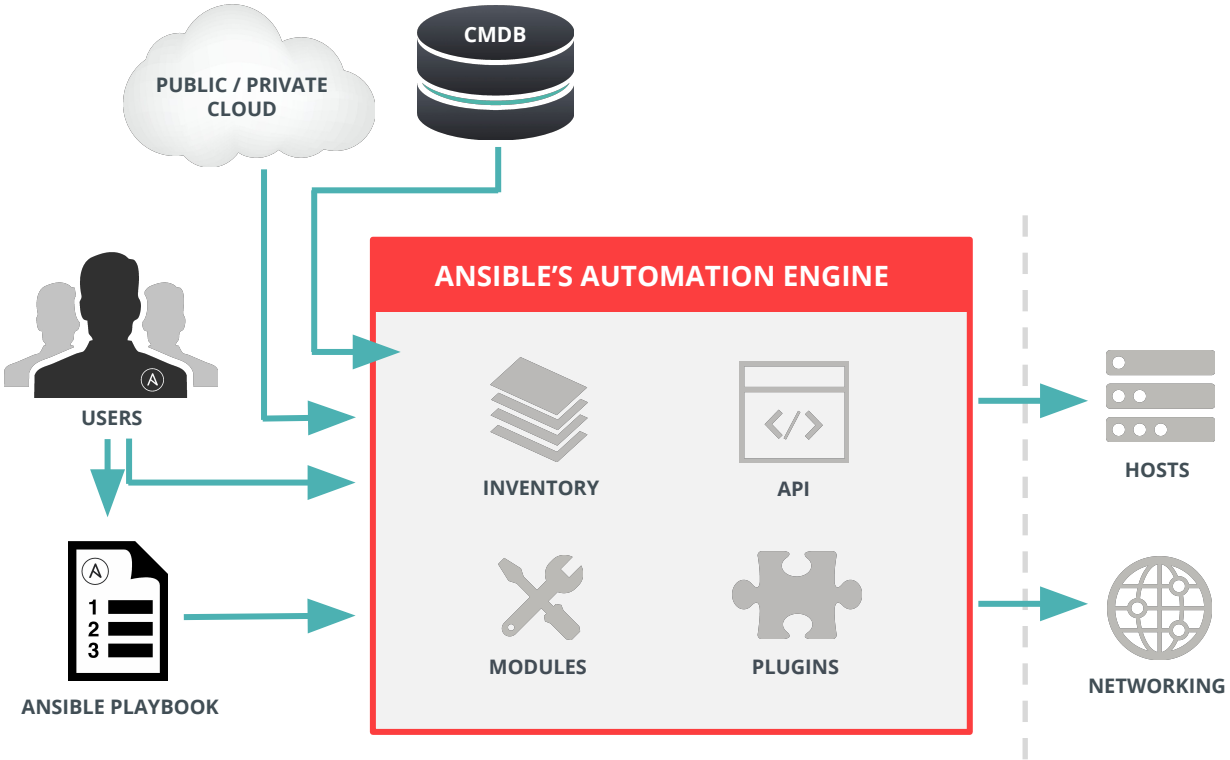
THE MOST POPULAR OPEN SOURCE AUTOMATION PLATFORM

Started in 2012
Goal: Unify provisioning, configuration, and application deployment

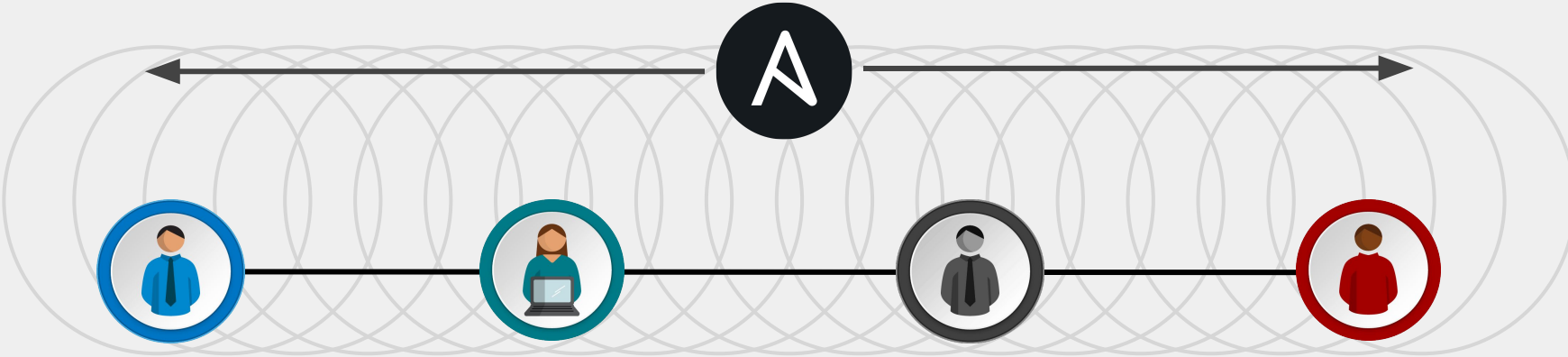
Result: Ansible — A python-based command line engine that interprets and executes YAML-based “playbooks” that contain one or more “plays” or tasks.

Five years later: 4,000+ contributors, 6,700 forks, 21,000 GitHub Stars
250,000 downloads a month
1000+ included modules (250+ networking), 34+ platforms

ANSIBLE UNDER THE HOOD



ANSIBLE IS THE UNIVERSAL LANGUAGE

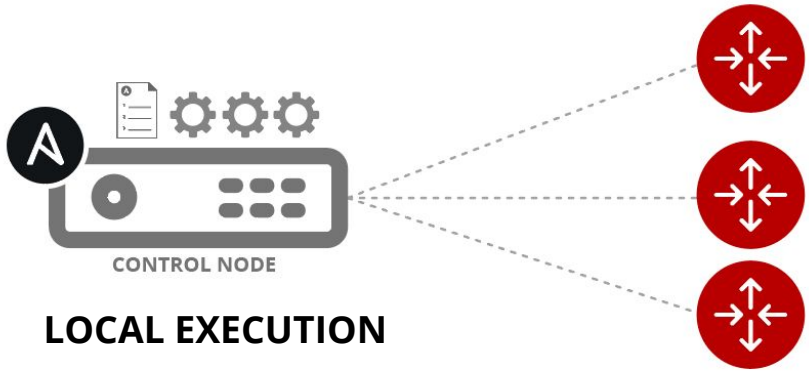


BUSINESS DEV NETWORK IT OPERATIONS

HOW DOES NETWORK AUTOMATION WORK?

ANSIBLE

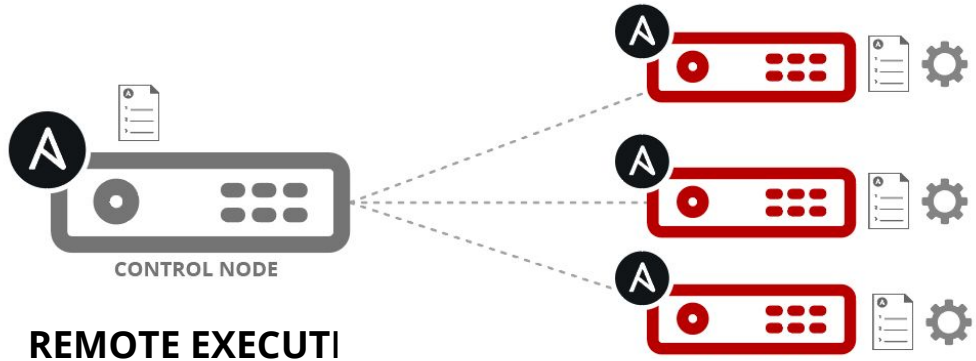
Module code is executed locally on the control node



LOCAL EXECUTION

NETWORKING DEVICES

Module code is copied to the managed node, executed, then removed



REMOTE EXECUTI

LINUX/WINDOWS HOSTS

redhat.

What make network Automation difficult?

- Network device are traditionally closed system with vendor specific cli
- Configuration line are huge per system
- Configuration is not always equals to desired state or operational state
- Network engineers typically do not have a Sysadmin or programming background
- Network serve multiple applications

Cli config difference across vendors

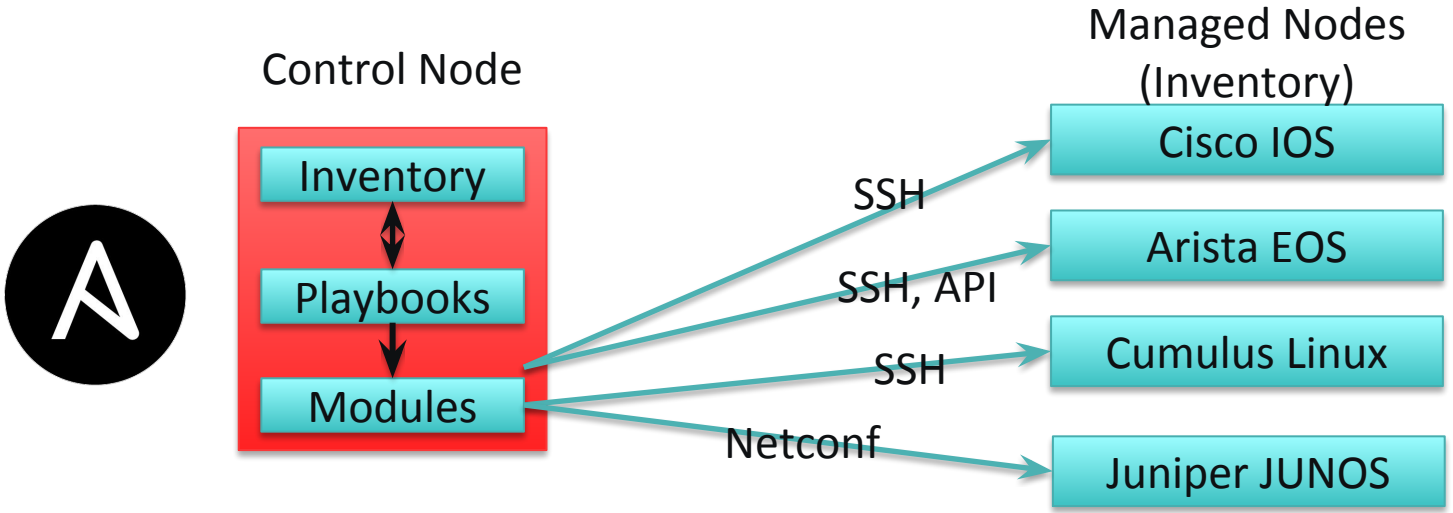
IOS

```
interface GigabitEthernet2
 description core
 ip address 192.168.2.3 255.255.255.0
 shutdown
!
```

Junos

```
interfaces {
    ge-0/0/2 {
        description core;
        disable;
        unit 0 {
            family inet {
                address 192.168.2.3/24;
            }
        }
    }
}
```

HOW DOES IT WORK?



Modules: Handles execution of remote system commands – required for most networking
Control Node: Any client system (server, laptop, VM) running Linux or Mac OSX
Requires: Python 2.6 or greater

Managed Nodes (Inventory): A collection of endpoints being managed via SSH or API.
Examples: Switches, Routers, Firewalls, Servers, VMs, Containers, Cloud Devices, etc.

Netconf: IETF management standard protocol

redhat. | ANSIBLE

- RPC based request response communication over ssh
- XML based encoding

  * Supports vendor specific data model

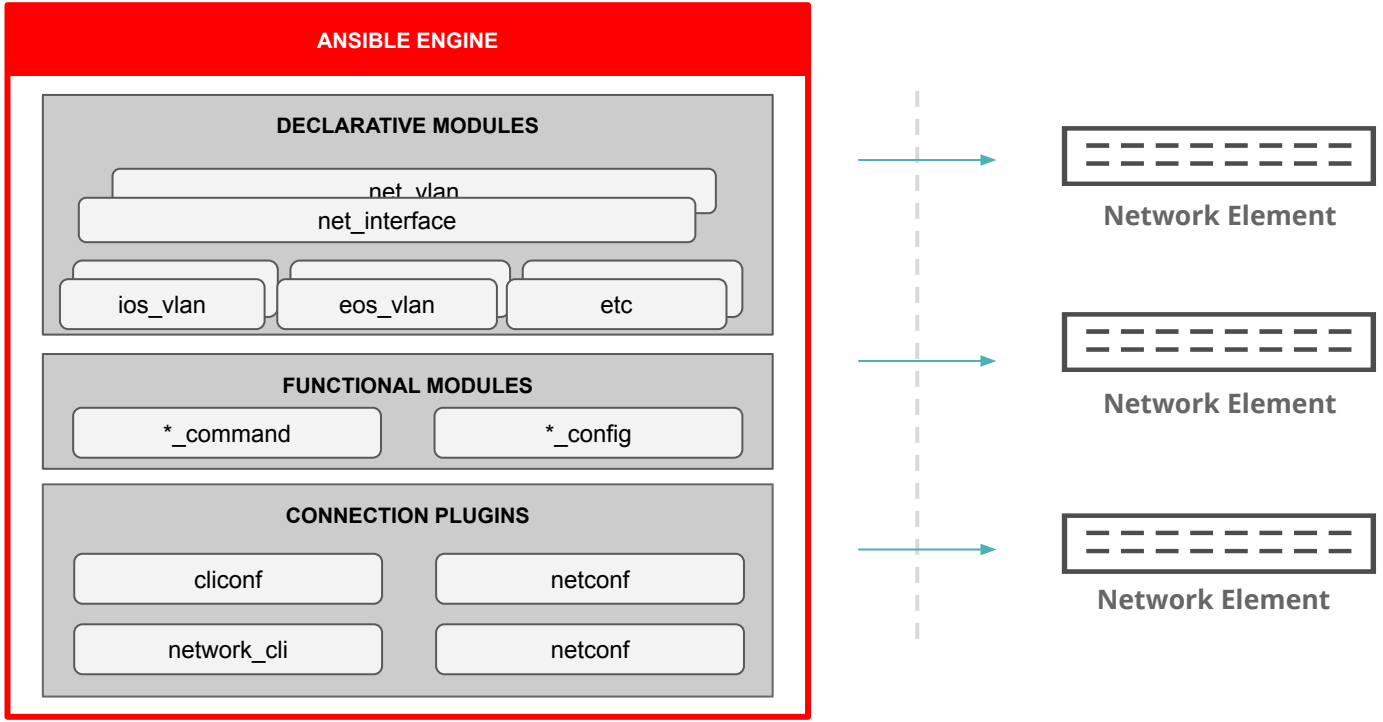
- Configuration RPC's

  * get-config, edit-config, copy-config, delete-config, lock, unlock

- Operational RPC's

  * get, Generally maps to cli "show" command

Ansible Network Overview



NETWORK MODULES: BUILT-IN DEVICE ENABLEMENT

A10

Apstra AOS

Arista EOS, CVP

Aruba Networks

AVI Networks

Big Switch Networks

Brocade Ironware

Cisco ACI, AireOS, ASA, Firepower, IOS, IOS-XR, Meraki, NSO, NX-OS

Citrix Netscaler

Cumulus Linux

Dell OS6, OS9, OS10

Exoscale

Extreme EX-OS, NOS, SLX-OS, VOSS

F5 BIG-IP, BIG-IQ

Fortinet FortIOS, FMGR

Huawei CloudEngine

Illumos

Infoblox NIOS

Juniper JunOS

Lenovo CNOS, ENOS

Mellanox ONYX

MikroTik RouterOS

Openswitch (OPX)

Ordnance

NETCONF

Netvisor

OpenSwitch

Open vSwitch (OVS)

Palo Alto PAN-OS

Nokia NetAct, SR OS

Ubiquiti EdgeOS

VyOS

NETWORK MODULES: BASIC

`*_command:`
Run command get/use output

`*_config:`
Make a change to the config with context

`*_facts:`
Get information (e.g. OS version, interfaces, etc.)

`*_template (D)`
Deploy Template: Apply J2 Template.
Deprecated in favor of *_config commands

Ios

- ios_command - Run commands on remote devices running Cisco IOS
- ios_config - Manage Cisco IOS configuration sections
- ios_facts - Collect facts from remote devices running IOS
- ios_template (D) - Manage Cisco IOS device configurations over SSH

Iosxr

- iosxr_command - Run commands on remote devices running Cisco iosxr
- iosxr_config - Manage Cisco IOS XR configuration sections
- iosxr_facts - Collect facts from remote devices running IOS-XR
- iosxr_template (D) - Manage Cisco IOSXR device configurations over SSH

PLAYBOOK EXAMPLE

```
---
- name: run multiple commands and evaluate the output
  hosts: ios01
  connection: network_cli
  tasks:
  - name: show version and show interfaces
    ios_command:
      commands:
         - show version
         - show interfaces
      wait_for:
         - result[0] contains IOS
         - result[1] contains Loopback0
```

PLAYBOOK EXAMPLE – REAL WORLD

redhat. | ANSIBLE

```
---
- name: configure ios interface
  hosts: ios01
  connection: network_cli
tasks:
    - name: collect device running-config
      ios_command:
        commands: show running-config interface GigabitEthernet0/2
        provider: "{{ cli }}"
      register: config

    - name: administratively enable interface
      ios_config:
        lines: no shutdown
        parents: interface GigabitEthernet0/2
        provider: "{{ cli }}"
      when: '"shutdown" in config.stdout[0]'
```

PLAYBOOK EXAMPLE – REAL WORLD (CONT.)

redhat. | ANSIBLE

```
- name: verify operational status
  connection: network_cli
  ios_command:
    commands:
      - show interfaces GigabitEthernet0/2
      - show cdp neighbors GigabitEthernet0/2 detail
    waitfor:
      - result[0] contains 'line protocol is up'
      - result[1] contains 'iosxr03'
      - result[1] contains '10.0.0.42'
```

PLAYBOOK EXAMPLE – REAL WORLD (RESULTS)

```
(ansible)[network]$ ansible-playbook ios_interface.yaml

PLAY [configure ios interface] ************************************

TASK [collect device running-config] ******************************
ok: [ios01]

TASK [administratively enable interface] **************************
changed: [ios01]

TASK [verify operational status] **********************************
ok: [ios01]

PLAY RECAP ********************************************************
ios01                      : ok=3    changed=1    unreachable=0    failed=0
```

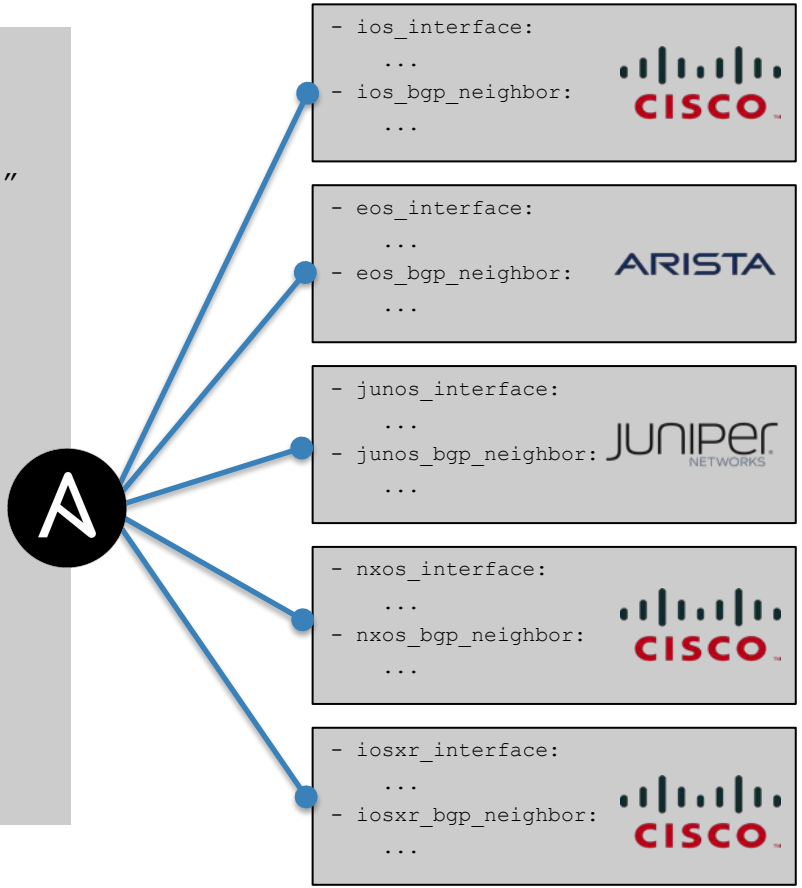
RESOURCE MODULES

```
---
- name: system node properties
  hosts: all

    tasks:
      - name: configure eos system properties
        eos_system:
          domain_name: ansible.com
          vrf: management
        when: network_os == 'eos'

      - name: configure nxos system properties
        nxos_system:
          domain_name: ansible.com
          vrf: management
        when: network_os == 'nxos'

      - name: configure ios system properties
        ios_system:
          domain_name: ansible.com
          lookup_enabled: yes
        when: network_os == 'ios'
```

- Per Platform Implementation
- Declarative by design
- Abstracted over the connection
- Makes platforms happy
- … Not so much for operators

MINIMUM VIABLE PLATFORM AGNOSTIC (MVPA)

```
- name: configure network interface
  net_interface:
    name: "{{ interface_name }}"
    description: "{{ interface_description }}"
    enabled: yes
    mtu: 9000
    state: up

- name: configure bgp neighbors
  net_bgp_neighbor:
    peers: "{{ item.peer }}"
    remote_as: "{{ item.remote_as }}"
    update_source: Loopback0
    send_community: both
    enabled: yes
    state: present
```



redhat.

DECLARATIVE INTENT

Declared Configuration

Intended State

```
- name: configure interface
  net_interface:
    aggregate:
      name: GigabitEthernet0/2
      description: public interface configuration
      enabled: yes

      # check operational state
      state: connected
      tx_rate: ge(7Gbps)
      rx_rate: ge(2Gbps)
      delay: 30
      neighbors:
        - host: core-01
          port: Ethernet5/2/6
```

Demo

Contributing to Ansible

- Community groups
- IRC channels
- Google Groups:
  https://groups.google.com/forum/#!forum/ansible-project
  https://groups.google.com/forum/#!forum/ansible-devel
- Ansible galaxy

Thank You

Github/IRC: @ganeshrn

RED HAT ANSIBLE ENGINE NETWORKING ADD-ON

NETWORK MODULES

- **Developed, maintained, tested, and supported** by Red Hat
- **140+ supported modules** and growing*
- Red Hat **reports and fixes problems**
- **Networking modules included** with Ansible Engine offering, but **the Ansible Engine Networking Add-On** SKU purchase is **required** for full support

*take special note of the specific supported platforms

NETWORKING ADD-ON INCLUDED SUPPORT:

Arista EOS

Cisco IOS

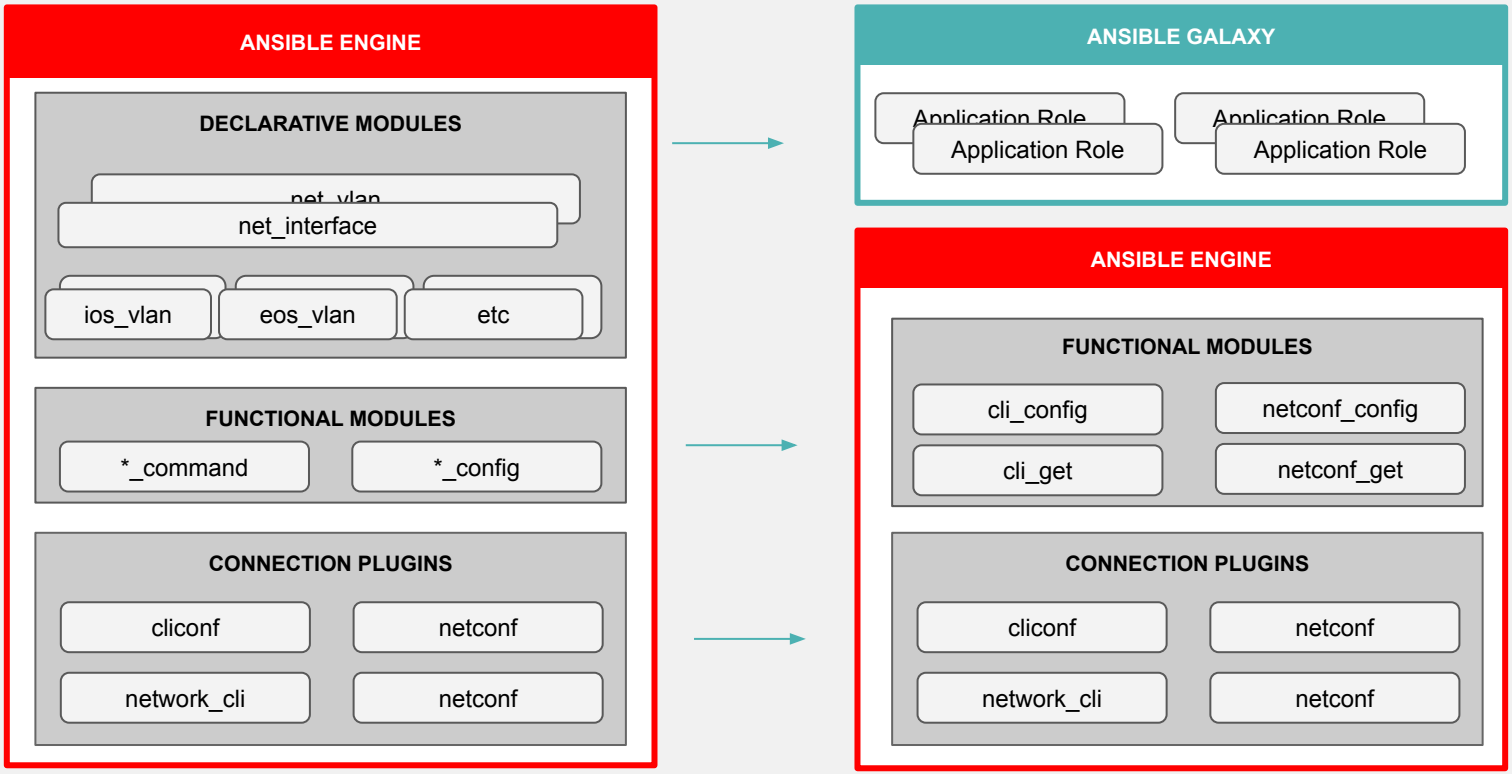
Cisco IOS XR

Cisco NX-OS

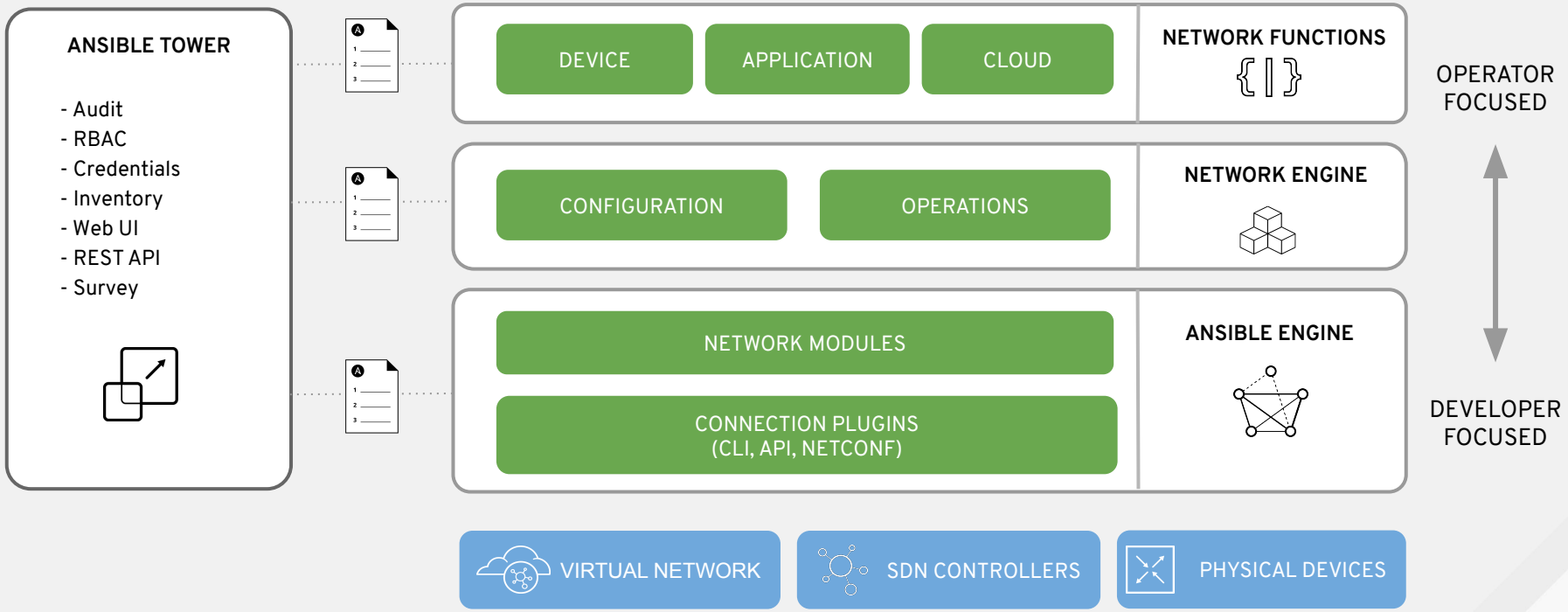
Juniper Junos

Open vSwitch

VyOS


ANSIBLE ENGINE
DECLARATIVE MODULES
net_vlan
net_interface
ios_vlan
eos_vlan
etc
FUNCTIONAL MODULES
*_command
*_config
CONNECTION PLUGINS
cliconf
netconf
network_cli
netconf
ANSIBLE GALAXY
Application Role
Application Role
Application Role
Application Role
ANSIBLE ENGINE
FUNCTIONAL MODULES
cli_config
netconf_config
cli_get
netconf_get
CONNECTION PLUGINS
cliconf
netconf
network_cli
netconf

ANSIBLE NETWORK STACK ARCHITECTURE



Application Roles

Transition away from “install everything” model

- Separation of content from engine *(starting post 2.5)*
- Maintain baseline device support and functional modules

App Store approach to integration

- Application roles initially hosted and managed at ***github.com/ansible-network***
- Roles are independently developed / managed / released
- Iterative Agile development methodology (release early, release often)
- Increase velocity of feature introduction
- Auto release roles based on PR merge and CI pass (ideally)

Create application role development strategy, environment & tooling *(as necessary)*

- Show customers “how its done” through value-add implementations
- Focus on ease of use, remove the choices

IRC:
#ansible, #ansible-devel, #ansible-meeting

Google groups:
ansible-devel@googlegroups.com
ansible-project@googlegroups.com

Freenode: ganeshrn
Github: ganeshrn

Persistent connection framework

junos_facts module

```
---
- name: Get junos facts
  hosts: junos
  connection: local
  tasks:
    - name: Collect facts for junos
      junos_facts:
        gather_subset: hardware
        config_format: text
      register: response

    - name: Print facts
      debug:
        var: response
```

junos_config module using line

```
---
- name: Run junos configuration command
  hosts: junos
  connection: local
  tasks:
    - name: Run configuration command
      junos_config:
        lines:
          - set system ntp server 1.1.1.1
          - set system services ftp
      register: response
```

junos_config module using src

```
---
- name: Run junos configuration command
  hosts: junos
  connection: local
  tasks:
    - name: Run configuration command
      junos_config:
        src: basic/junos_config.text

      register: response
```