Deep dive on Ansible collections for network automation

Ganesh B. Nalawade
Github/IRC: ganeshrn
Twitter: @ganeshrn

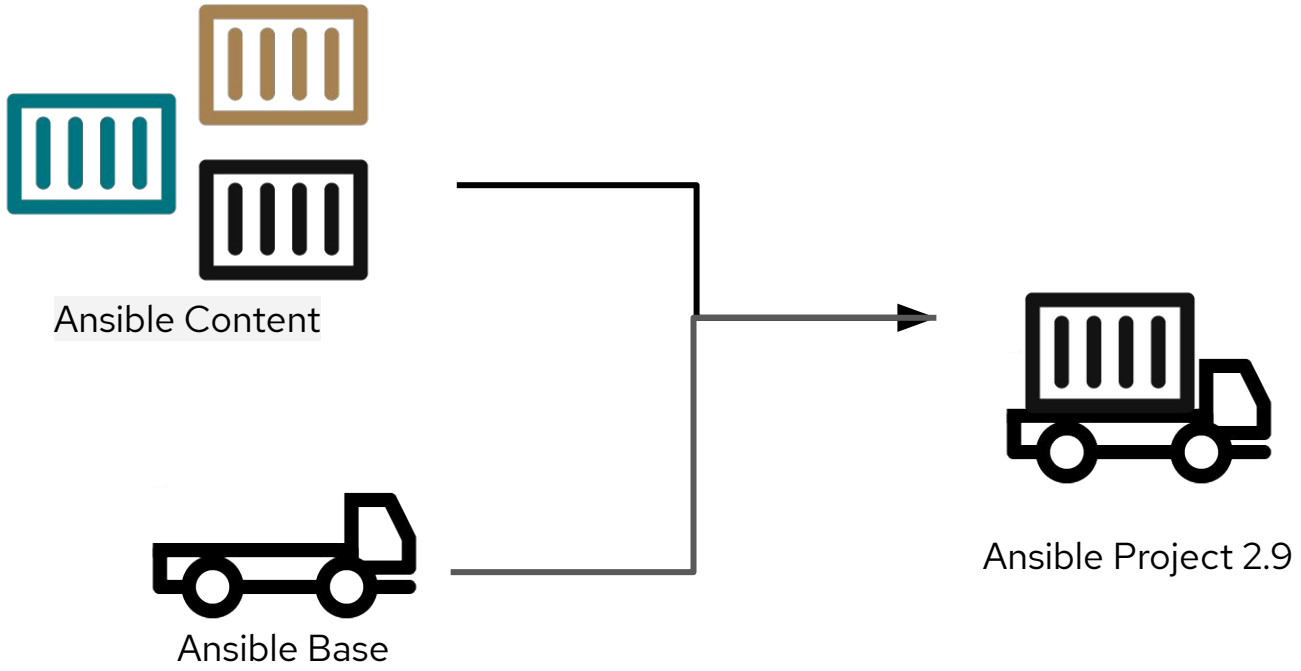
About me

- Works as Principal Software Engineer with Ansible Content Engineering team.
- Software developer with 13+ years of experience
- Worked extensively on network management plane to develop software features for automation and programmability
- Co-organiser of Ansible Pune, India meetup

Agenda

- Why and what is Ansible collection?
- Ansible Network collections
- Network specific Ansible plugins
- Using an existing network collection
- Creating your collections with network specific Ansible plugins
  - terminal
  - cliconf
  - netconf
  - httpapi
- Demo

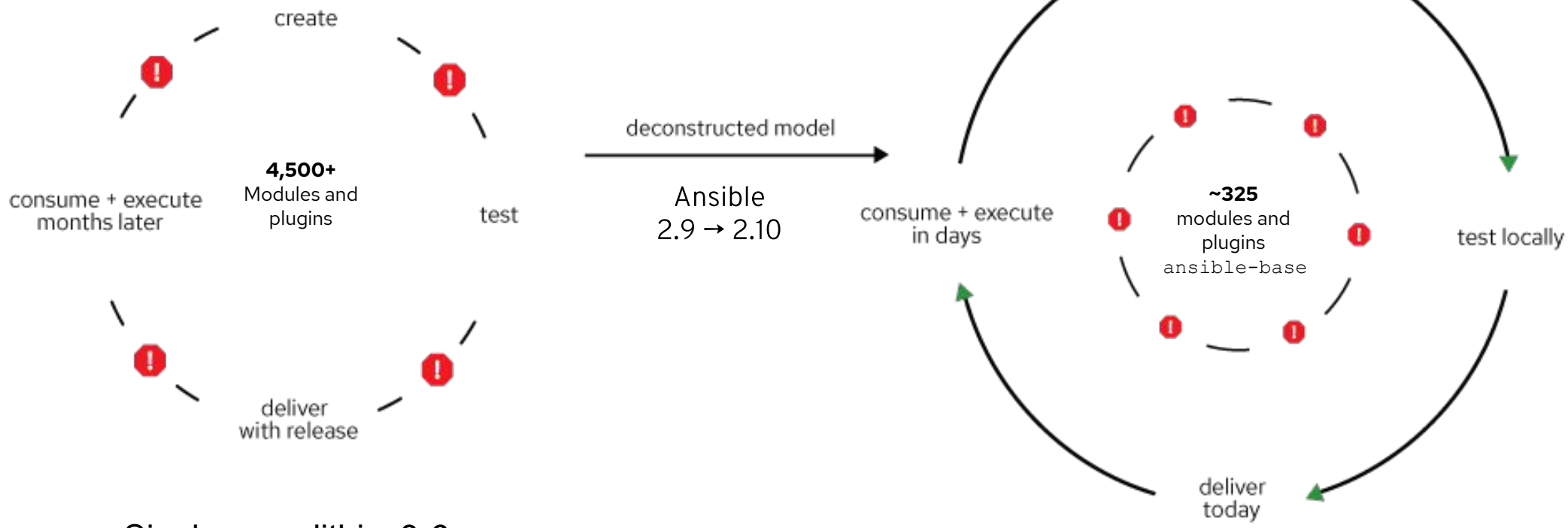
Why Ansible collection?



4

Why Ansible collection?

Separate Ansible from the content



5

5

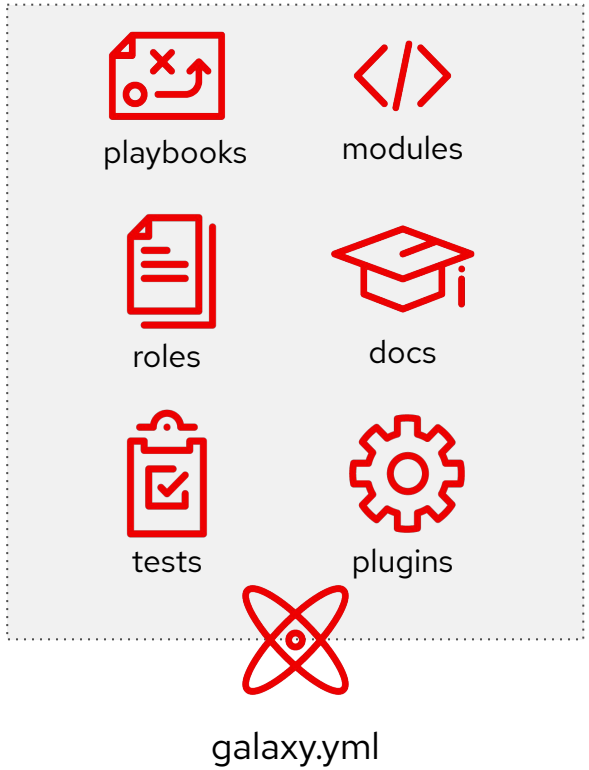
Single monolithic: 3-6 months release cycle

inner circle: slower release cycle (ansible-base)
outer circle: faster release cycle (content in collections)

What is Ansible collection?

What is in it?

- Components are well defined, there is a standard for the directory structure
- Requires same standard of documentation that the Ansible Project does
- Scaffolding can be created with Ansible Galaxy command



6

What is Ansible collection?

Directory Structure

- **docs/**: local documentation for the collection
- **galaxy.yml**: source data for the MANIFEST.json that will be part of the collection package
- **playbooks/**: playbooks reside here
  - **tasks/**: this holds 'task list files' for include_tasks/import_tasks usage
- **plugins/**: all ansible plugins and modules go here, each in its own subdir
  - **modules/**: ansible modules
  - **modules_utils/** util code for modules and other plugins
  - **lookups/**: lookup plugins
  - **filters/**: Jinja2 filter plugins
  - **connection/**: connection plugins required if not using default
- **roles/**: directory for ansible roles
- **tests**/: tests for the collection's content

Where do I get it?

Ansible Galaxy

galaxy.ansible.com

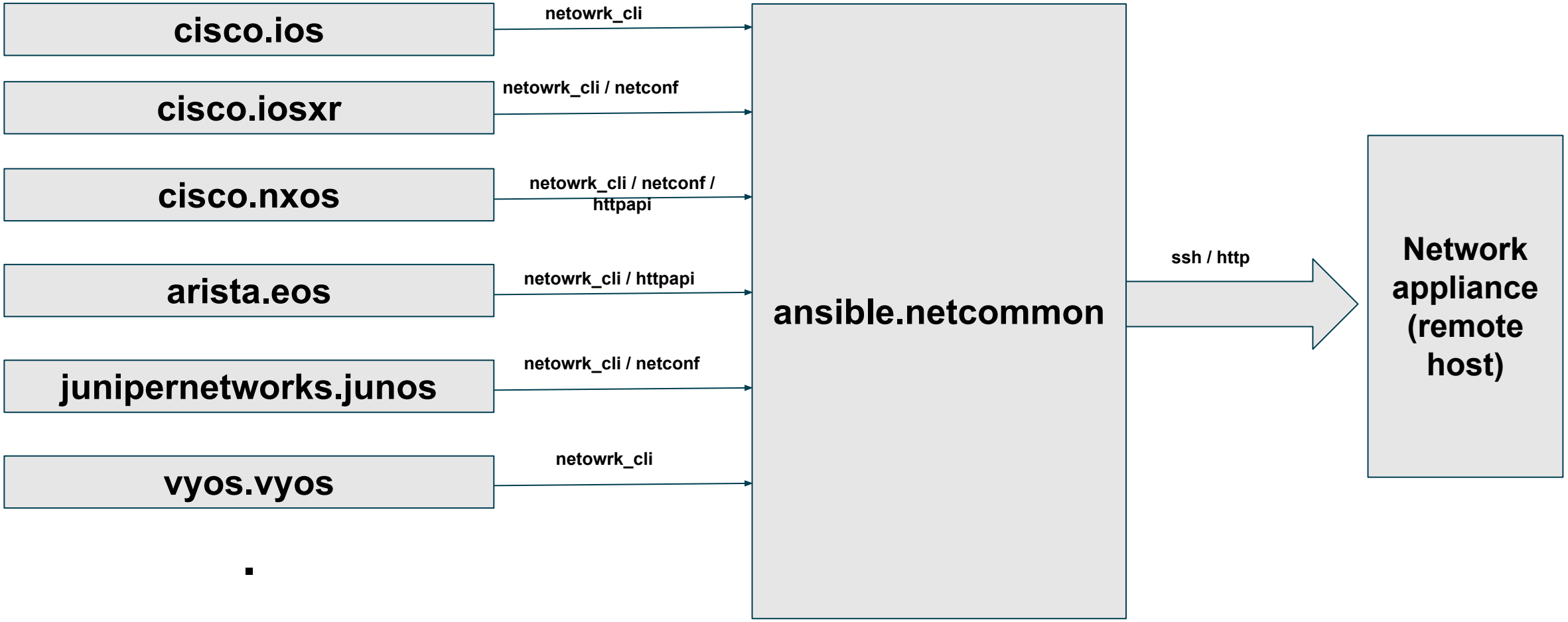
- Community supported
- Extended to leverage Collections framework
- “Latest and greatest”

Ansible Automation Hub

cloud.redhat.com

- Certified, jointly supported by Red Hat and Partner
- Access to advanced analytics
- “Slow and steady”

Ansible network collections



https://github.com/ansible-collections

Using an existing network collection

How do I install it?

Install an Ansible Collection:

```
ansible-galaxy collection install cisco.ios
```

This installs (by default) into:

```
~/.ansible/collections/ansible_collections
```

Using an existing network collection (example)

Platform agnostic modules using **ansible.netcommon.network_cli** connection plugin

```
---
- hosts: ios
   vars:
     ansible_connection: ansible.netcommon.network_cli
     ansible_network_os: cisco.ios.ios

 tasks:
 - name: run operational mode command
   ansible.netcommon.cli_command:
     command: show version
   register: result

 - name: run configuration mode command
   ansible.netcommon.cli_config:
     config: "{{ item }}"
   loop:
     - interface GigabitEthernet1/0/11
     - shutdown
   register: result
```

- Platform collection implements **cliconf** and **terminal** plugins (more on this in later slides) to work with network_cli connection type

Using an existing network collection (example)

Platform specific modules using **ansible.netcommon.network_cli** connection plugin

```
---
- hosts: ios
  vars:
    ansible_connection: ansible.netcommon.network_cli
    ansible_network_os: cisco.ios.ios

  tasks:
  - name: Merge interface configuration
    cisco.ios.ios_interfaces:
      config:
        - name: GigabitEthernet0/2
          description: Configured by Ansible
          enabled: true
      state: merged
    register: result
```

- Platform collection implements **cliconf** and **terminal** plugins (more on this in later slides) to work with network_cli connection type

Creating your own collection

Create an Ansible Collection:

```
ansible-galaxy collection init myorg.nos
```

```
$ tree myorg
myorg
└── nos
    ├── docs
    ├── galaxy.yml
    ├── plugins
    │   └── README.md
    ├── README.md
    └── roles
```

Ansible network specific plugins

- **terminal plugin:**
  - Should be implemented for the platform collection to work with *ansible.netcommon.network_cli* connection type.
  - This plugin controls the terminal parameters of the host after remote login is successful for eg. privilege escalation

Ansible network specific plugins

```
$cat myorg/nos/plugins/terminal/nos.py
```

```
class TerminalModule(TerminalBase):

    terminal_stdout_re = [...]

    terminal_stderr_re = [...]

    def on_open_shell(self):...

    def on_become(self, passwd=None):...

    def on_unbecome(self):...
```

Ansible network specific plugins

- **cliconf plugin:**
  - Should be implemented for the platform collection to work with *ansible.netcommon.network_cli* connection type.
  - This plugin abstracts low level command execution calls within the connection plugin and expose set to API’s to the module code

Ansible network specific plugins

```
$cat myorg/nos/plugins/cliconf/nos.py
```

```python
from ansible.plugins.cliconf import CliconfBase, enable_mode

class Cliconf(CliconfBase):

    @enable_mode
    def get_config(self, source='running', flags=None, format=None):...

    @enable_mode
    def edit_config(self, candidate=None, commit=True, replace=None, comment=None):...

    def get_diff(self, candidate=None, running=None, diff_match='line', diff_ignore_lines=None, path=None, diff_replace='line'):

    def get(self, command=None, prompt=None, answer=None, sendonly=False, output=None, newline=True, check_all=False):...

    def get_device_info(self):...

    def get_capabilities(self):...

    def run_commands(self, commands=None, check_rc=True):...
```

Ansible network specific plugins

- **netconf plugin:**
  - Should be implemented for the platform collection to work with ***ansible.netcommon.netconf*** connection to support vendor specific netconf RPC's.
  - For example in case of junos the proprietary RPC methods are implemented in ***"junipernetworks/junos/plugins/netconf/junos.py"*** and value of **"ansible_network_os"** is set the name of the netconf plugin file, that is ***"junipernetworks.junos.junos"*** in this case.
  - If the network device support standard netconf (RFC 6241) operation like "get", "get-config", "edit-config" etc set the value of **"ansible_network_os"** to **"ansible.netcommon.default"**
  - "**ansible.netcommon.netconf_get"**, "**ansible.netcommon.netconf_config**" and "**ansible.netcommon.netconf_rpc**" modules can be used to talk to netconf enable remote host.

Ansible network specific plugins

```
$cat ansible/nos/plugins/netconf/nos.py
```

```
from ansible.plugins.netconf import NetconfBase

class Netconf(NetconfBase):

    def load_configuration(self, format='xml', action='merge',
                           target='candidate', config=None):...

    def compare_configuration(self, rollback=0):...

    def reboot(self):...
```

Ansible network specific plugins

$cat ansible/lib/ansible/plugins/netconf/__init__.py

```
from ansible.plugins import AnsiblePlugin

class NetconfBase(AnsiblePlugin):

    def get_config(self, source=None, filter=None):...

    def get(self, filter=None, with_defaults=None):...

    def edit_config(self, config=None, format='xml', target='candidate',
                    default_operation=None, test_option=None, error_option=None):...

    def validate(self, source='candidate'):...

    def copy_config(self, source, target):...

    def lock(self, target="candidate"):...

    def unlock(self, target="candidate"):...

    def discard_changes(self):...

    def commit(self, confirmed=False, timeout=None, persist=None):...

    def get_schema(self, identifier=None, version=None, format=None):...

    def delete_config(self, target):...

    def get_capabilities(self):...
```

Ansible network specific plugins

- **httpapi plugin:**
  - Should be implemented for the platform collection to work with ***ansible.netcommon.httpapi*** connection.
  - Can be implemented for remote host that supports http and not specifc for networking
  - A well-constructed httpapi plugin should take in structured data and return structured data.

Ansible network specific plugins

```
$cat myorg/nos/plugins/httpapi/nos.py
```

```
class HttpApi(HttpApiBase):
    def __init__(self, connection):...

    def set_become(self, become_context):...

    def login(self, username, password):...

    def logout(self):...

    def update_auth(self, response, response_text):...

    def handle_httperror(self, exc):...

    def send_request(self, data, **message_kwargs):...
```

Creating your own collection

```
$tree myorg/
myorg/
`-- nos
    |-- README.md
    |-- docs
    |-- galaxy.yml
    |-- plugins
    |   |-- README.md
    |   |-- cliconf
    |   |   `-- nos.py
    |   |-- httpapi
    |   |   `-- nos.py
    |   |-- netconf
    |   |   `-- nos.py
    |   `-- terminal
    |       `-- nos.py
    `-- roles
```

Building and installing your own collection

Building an Ansible Collection:

```
[user@rhel8 ~]$ cd /home/user/myorg/nos

[user@rhel8 ~]$ ansible-galaxy collection build
Created collection for myorg.nos at
/home/user/myorg/nos/myorg-nos-1.0.0.tar.gz
```

Install Ansible Collection from local file:

```
[user@rhel8 ~]$ ansible-galaxy collection install myorg-nos-1.0.0.tar.gz
myorg-nos-1.0.0.tar.gz
Process install dependency map
Starting collection install process
Installing 'myorg-nos:1.0.0' to
'/home/student1/.ansible/collections/ansible_collections/myorg/nos'
```

Use you new content

Platform agnostic modules using **ansible.netcommon.network_cli** connection plugin

```
---
- hosts: nos_hosts
  vars:
    ansible_connection: ansible.netcommon.network_cli
    ansible_network_os: myorg.nos.nos

 tasks:
 - name: run operational mode command
   ansible.netcommon.cli_command:
     command: show version
   register: result
```

Publishing collection to ansible galaxy

```
[user@rhel8 ~]$ $ansible-galaxy collection publish
myorg-nos-1.0.0.tar.gz --token <API-token from ansible-galaxy>
```

Demo

Questions?

Thank you!