Interop DIGITAL October 5-8

Molecule for Testing Network Automation Ansible Projects

David Flores

Snr Automation Consultant @ NetworkToCode

About Me

- David Flores
  - Senior Network Automation Engineer at NetworkToCode.
  - Background – Software Development/Network Automation, worked for Service Provider/Datacenter networks.
- Twitter - @davidban77 (AKA: netpanda)
- Blog
  - https://networktocode.com/blog
  - https://davidban77.hashnode.dev/
- Github
  - https://github.com/davidban77

Overview

About this Talk

- How to use Molecule for Ansible projects on a Network Automation context.
- Importance of testing and common challenges in Ansible projects.
- Molecule features (commission/decommission of instances, preparation, idempotency, verification, etc…).
- Ansible project and testing scenarios explanation.
- Demo.
  - https://github.com/networktocode/interop2020-ansible-molecule

About You

- Good knowledge in Ansible and Molecule
- Experience developing Ansible collections and projects tested with Molecule
- Experience setting up CI/CD pipelines for Ansible projects

Challenges for Network Automation

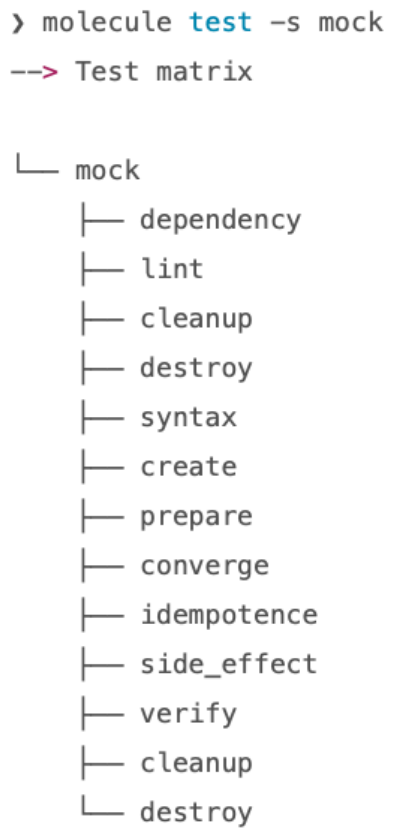
- You need network devices instances to test you Ansible projects.
- Molecule is a well-known testing tool for Ansible in the DevOps world.
- No out-of-the-box virtualization providers.
- Access to network devices virtual images can be difficult to obtain.
- Only a few vendors provide network device container images.
  - Checkout **vrnetlabs**.
- Most of the emulation/virtualization tools out there were focused on GUI, not so much in accessing the resources in a programmatic way. Although that is currently changing:
  - GNS3, EVE-NG, CML2 have clients to manage them programmatically.
  - Emulation network equipment projects like **cisshgo**, for fast and simple testing.

Molecule Overview

- Molecule is:
  - CLI tool for testing Ansible playbooks, roles or collections.
  - Brings up instances to execute the ansible projects and validate their state.
  - Provides a framework that manages the lifecycle and resources a test.
  - Built with the concept of pipeline and scenarios.
  - Supports multiple virtualization providers.
- Where is used:
  - Local development and testing for your Ansible projects.
  - CI pipeline testing

Providers

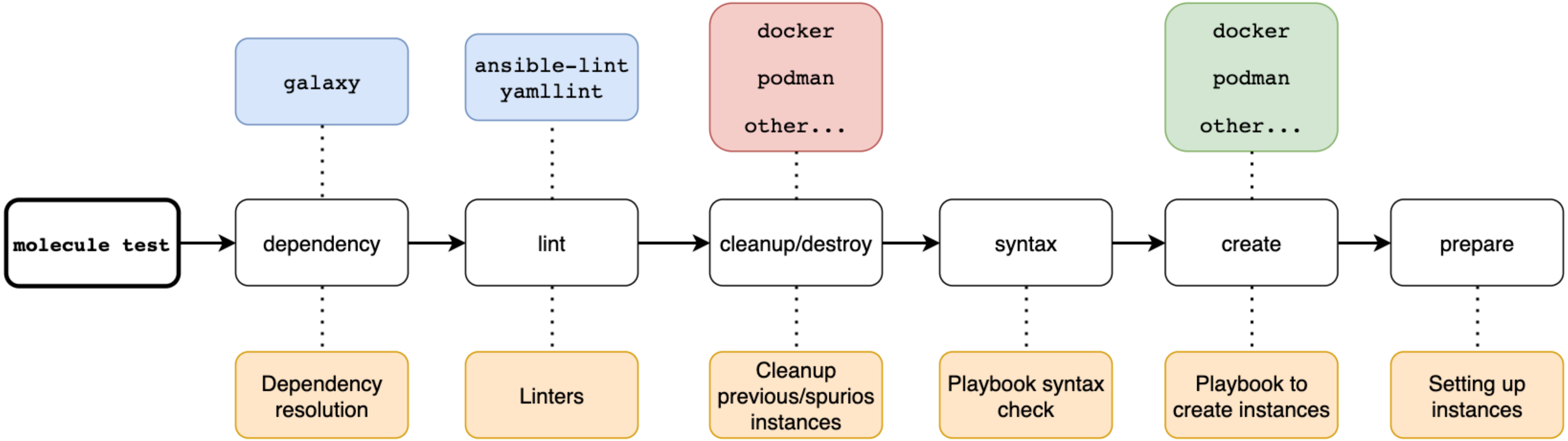
CI Pipelines



```
❯ molecule test -s mock
--> Test matrix

└── mock
    ├── dependency
    ├── lint
    ├── cleanup
    ├── destroy
    ├── syntax
    ├── create
    ├── prepare
    ├── converge
    ├── idempotence
    ├── side_effect
    ├── verify
    ├── cleanup
    └── destroy
```

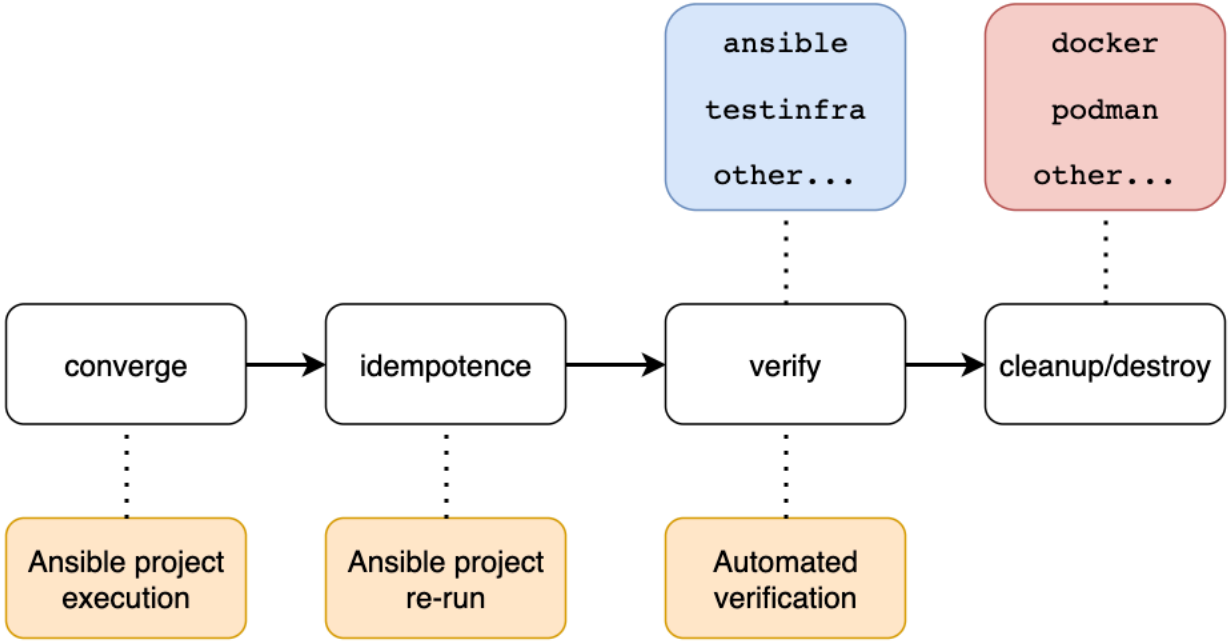
Molecule Standard Pipeline

- Ansible checks and test setup



Molecule Standard Pipeline

- Tests Execution, validation and cleanup



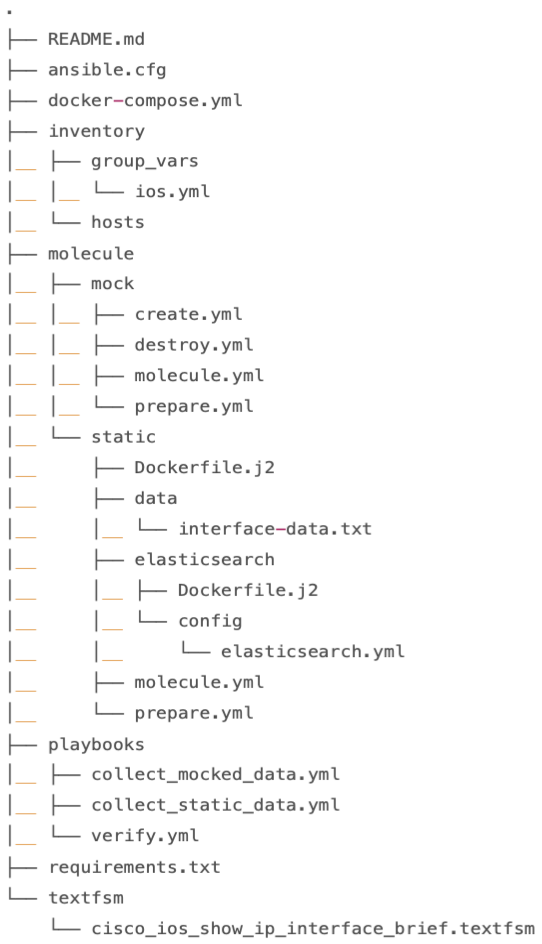
Molecule Scenario Structure

- Molecule runs the pipeline steps on a per-scenario basis.
- Each scenario must contain a `molecule.yml` file that instructs how molecule will behave on that specific scenario.
- Other playbooks may be defined to explicitly state the workflow of that stage.
- Molecule provides a default template for creating scenarios directory structure:

```
molecule init --scenario example --driver docker
```

- When using drivers like docker, you can specify `Dockerfile` if you want molecule to create the image first.
- Inventory and playbooks of the project can be linked/used in the molecule test scenario.

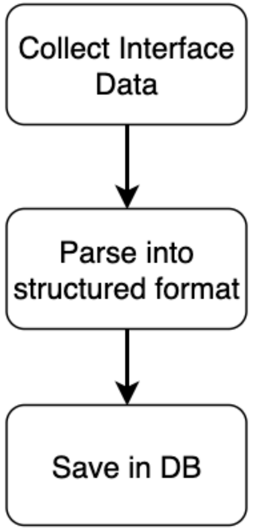
Example Project

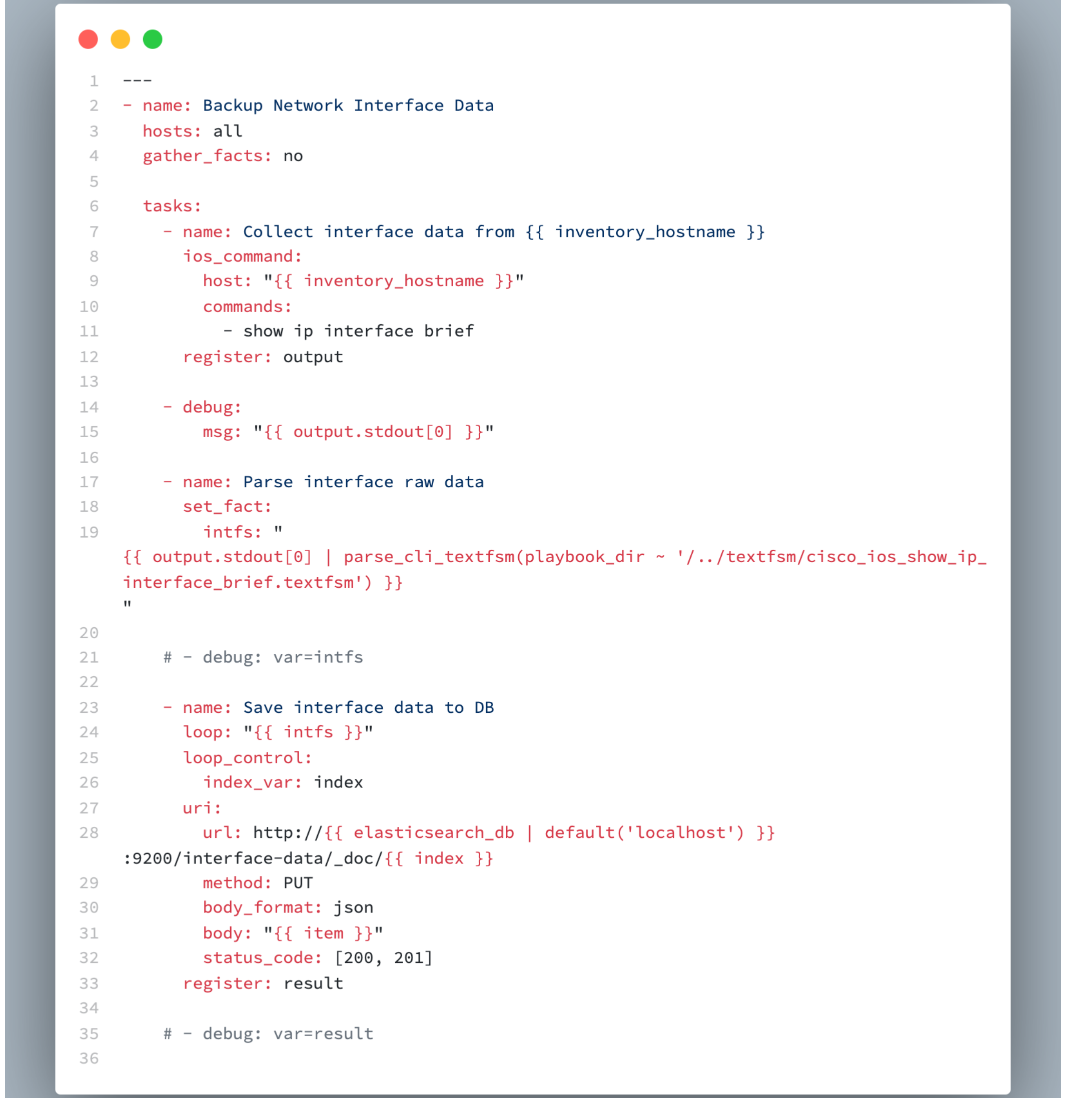


```
.
├── README.md
├── ansible.cfg
├── docker-compose.yml
├── inventory
|__ ├── group_vars
|__ |__ └── ios.yml
|__ └── hosts
├── molecule
|__ ├── mock
|__ |__ ├── create.yml
|__ |__ ├── destroy.yml
|__ |__ ├── molecule.yml
|__ |__ └── prepare.yml
|__ └── static
|__     ├── Dockerfile.j2
|__     ├── data
|__     |__ └── interface-data.txt
|__     ├── elasticsearch
|__     |__ ├── Dockerfile.j2
|__     |__ └── config
|__     |__     └── elasticsearch.yml
|__     ├── molecule.yml
|__     └── prepare.yml
├── playbooks
|__ ├── collect_mocked_data.yml
|__ ├── collect_static_data.yml
|__ └── verify.yml
├── requirements.txt
└── textfsm
    └── cisco_ios_show_ip_interface_brief.textfsm
```

Ansible Project Demo

- Ansible playbook workflow
- Molecule test will be based on the mock scenario
- Using **`cisshgo`** container to mock a network device
- Using **`elasticsearch`** container as database
- https://github.com/networktocode/interop2020-ansible-molecule

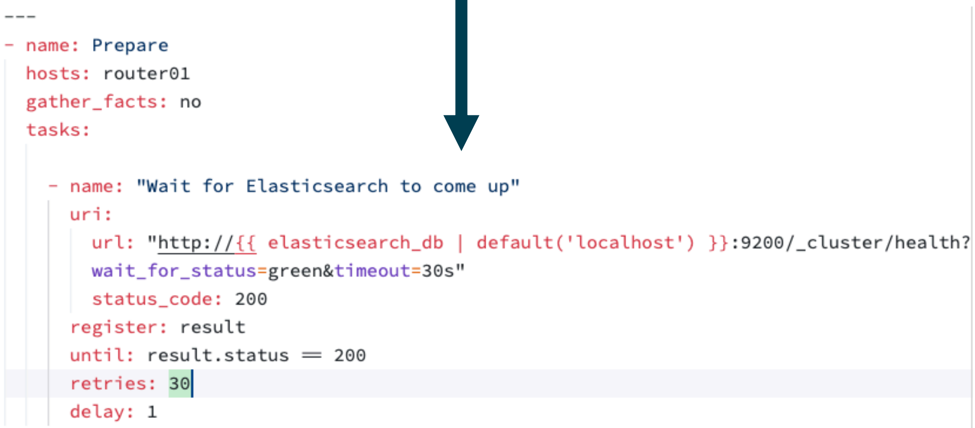




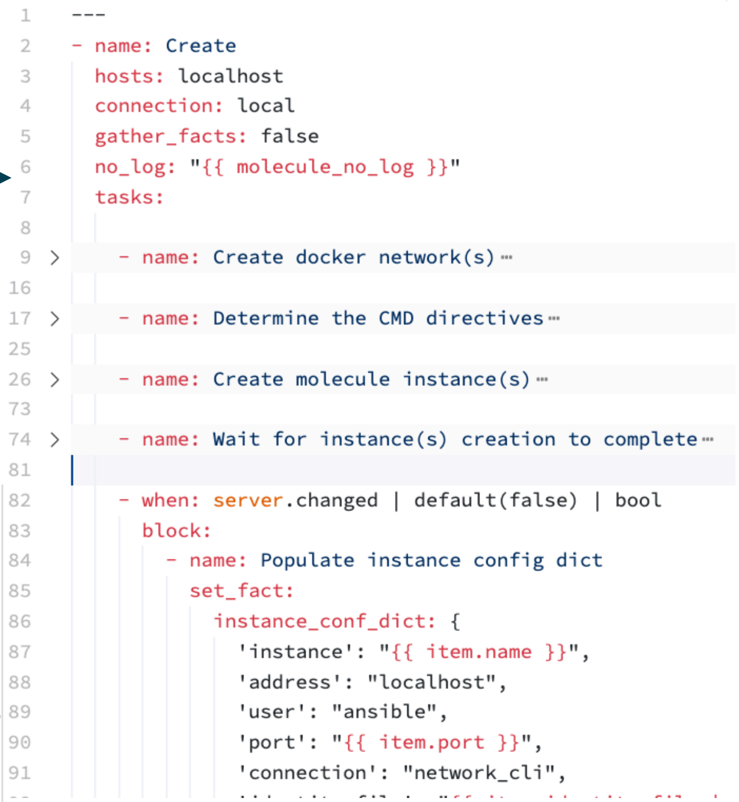
```yaml
---
- name: Backup Network Interface Data
  hosts: all
  gather_facts: no

  tasks:
    - name: Collect interface data from {{ inventory_hostname }}
      ios_command:
        host: "{{ inventory_hostname }}"
        commands:
          - show ip interface brief
      register: output

    - debug:
        msg: "{{ output.stdout[0] }}"

    - name: Parse interface raw data
      set_fact:
        intfs: "
{{ output.stdout[0] | parse_cli_textfsm(playbook_dir ~ '/../textfsm/cisco_ios_show_ip_
interface_brief.textfsm') }}
"

    # - debug: var=intfs

    - name: Save interface data to DB
      loop: "{{ intfs }}"
      loop_control:
        index_var: index
      uri:
        url: http://{{ elasticsearch_db | default('localhost') }}
:9200/interface-data/_doc/{{ index }}
        method: PUT
        body_format: json
        body: "{{ item }}"
        status_code: [200, 201]
      register: result

    # - debug: var=result

```

Molecule Mock Scenario

- Molecule YAML is used as a configuration file for molecule on that specific scenario.
- Create and Destroy playbooks. Based on the **`docker`** driver but tweaked to change the connection type to **`network_cli`**
- Prepare playbook which waits for Elasticsearch to complete bootup process

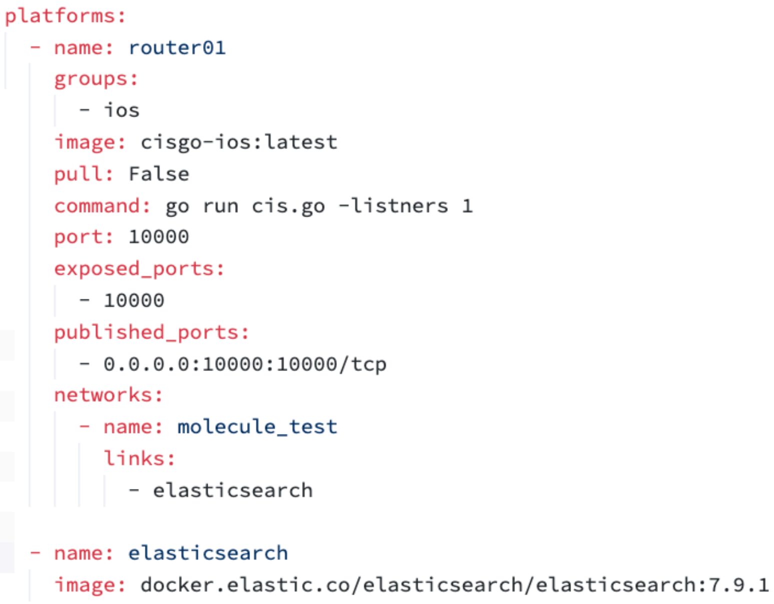


```yaml
---
- name: Prepare
  hosts: router01
  gather_facts: no
  tasks:

    - name: "Wait for Elasticsearch to come up"
      uri:
        url: "http://{{ elasticsearch_db | default('localhost') }}:9200/_cluster/health?wait_for_status=green&timeout=30s"
        status_code: 200
      register: result
      until: result.status == 200
      retries: 30
      delay: 1
```



```yaml
---
- name: Create
  hosts: localhost
  connection: local
  gather_facts: false
  no_log: "{{ molecule_no_log }}"
  tasks:

    - name: Create docker network(s) ⋯

    - name: Determine the CMD directives ⋯

    - name: Create molecule instance(s) ⋯

    - name: Wait for instance(s) creation to complete ⋯

    - when: server.changed | default(false) | bool
      block:
        - name: Populate instance config dict
          set_fact:
            instance_conf_dict: {
              'instance': "{{ item.name }}",
              'address': "localhost",
              'user': "ansible",
              'port': "{{ item.port }}",
              'connection': "network_cli",
```

Sections of **`molecule.yml`**



```yaml
platforms:
  - name: router01
    groups:
      - ios
    image: cisgo-ios:latest
    pull: False
    command: go run cis.go -listners 1
    port: 10000
    exposed_ports:
      - 10000
    published_ports:
      - 0.0.0.0:10000:10000/tcp
    networks:
      - name: molecule_test
        links:
          - elasticsearch

  - name: elasticsearch
    image: docker.elastic.co/elasticsearch/elasticsearch:7.9.1
```

Demo

Conclusions

- Treating Ansible projects using a TDD-like approach it will take more development time, but your change management windows will thank it later.
- Molecule might be overkill for simple, one-off use cases, but it will be an asset for more complex scenarios, where integration with other systems are expected.
- It is flexible enough to test each phase independently or all by running:

```
molecule test –s <scenario>
```

- It consumes lots of resources when performing the tests, specially CPU.
- Since the application is CLI-based and returns correct stdout/stderr codes, it works well with CI pipeline tools.

Links and Resources

- Molecule
  - Github: https://github.com/ansible-community/molecule
  - Docs: https://molecule.readthedocs.io/en/latest/getting-started.html
- Ansible Molecule Demo:
  - Github: https://github.com/networktocode/interop2020-ansible-molecule
- Other resources
  - Cisshgo: https://github.com/tbotnz/cisshgo
  - Vrnetlabs: https://github.com/plajjan/vrnetlab