

ANSIBLE

State assessment and data validation using Ansible

Ganesh B. Nalawade
Principal Software Engineer, Ansible
Github/IRC: ganeshrn
Twitter: @ganesh634

About me - Ganesh

- 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.

Agenda

- Use cases for operational state data
- Generating structured data using cli_parse module
- Validating structured data using validate module
- Remediation
- Input data validation
- Developer notes

Common state assessment workflow

- Retrieve:
 - Collect the current operational state from the remote host
 - Convert it into normalised structure data.
- Validate:
 - Define the desired state criteria in a standard based format that can be used across enterprise infrastructure teams
 - Validate the current state data against the pre-defined criteria to identify if there is any deviation.
- Remediate:
 - Required configuration changes
 - Reporting

- Conditional task and roles within Ansible playbooks
 - Only make configuration changes if all the BGP neighbours are healthy
- Fleet health assessment and inventory
 - Ensure all configured NTP servers are in sync
- Post change validation
 - LLDP, OSPF neighbours and reachability has not changed
- Custom reports using templates
 - Interface operating state vs. configured state

Operational state

the "show" commands

show interfaces

```

mgmt0 is up
admin state is up,
  Hardware: Ethernet, address: x200.0000.f8b5 (bia
abcd.0000.f8b5)
  Internet Address is 192.168.101.14/24

{"TABLE_interface": {"ROW_interface": [{"interface": "mgmt0",
"state": "up", "admin_state": "up", "eth_hw_desc":
"Ethernet", "eth_hw_addr": "x200.0000.f8b5", "eth_bia_addr":
"x200.0000.f8b5", "eth_ip_addr": "192.168.101.14",
"eth_ip_mask": "24", "eth_ip_prefix": "192.168.101.0", "et
h_mtu": "1500", "eth_bw": "1000000", "eth_dly": "10",
"eth_reliability":

<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply
xmlns="http://www.cisco.com/nxos:1.0:if_manager"
xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nf:data>
    <show><interface>

```

Multiple formats



```

cli_command:
  register:
set_fact:
  | filter_plugin
  | lookup_plugin

```

Multiple tasks and plugins



```

interfaces:
  admin:
  state:
    operating: up
    administrative: up

```

Desired format

TL;DR

An Ansible strength

Configuration management

An Ansible weakness

Operational state assessment

Let's fix that.

ansible.utils.cli_parse

- New module available now in ansible.utils collection <https://galaxy.ansible.com/ansible/utils>
- Works with all platforms
- Work with many parsing engines
- Single task to run a command, parse & set facts
- Returns structured data from show command output


```
tasks:  
- name: Run a command and parse results  
  ansible.utils.cli_parse:  
    command: show interfaces  
    parser:  
      name: ansible.utils.xxxx  
      set_fact: interfaces
```

- Runs the command on the device
- Parse using the 'xxxx' engine
- Uses default template folder
- Parsed data set as fact
- Command output returned as stdout

Available parsing engines

- **ansible.utils.textfsm**: Python module for parsing semi-formatted text
- **ansible.utils.ttp**: Template based parsing, low regex use, jinja like DSL
- **ansible.netcommon.native**: Internal jinja, regex, yaml. No additional 3rd party libraries required
- **ansible.netcommon.ntc_templates**: Predefined textfsm templates packaged as python library
- **ansible.netcommon.pyats**: Cisco Test Automation & Validation Solution (11 OSs/2500 parsers)
- **ansible.utils.xml**: convert XML to json using xmlltodict

Thank you library developers & contributors

Smart template discovery

```
templates/{{ os }}_{{ command }}.xyz
```

```
templates/eos_show_interfaces.yaml
```

```
templates/nxos_show_ntp_peers.textfsm
```

Demo on `ansible.utils.cli_parse`

ansible.utils.validate

- New module available now in ansible.utils collection <https://galaxy.ansible.com/ansible/utils>
- Works with all platforms
- Currently works with [jsonschema](#) validation engine
- Single task to read the structured data and validate it with data model schemas
- Returns either list of errors or success (in case data is valid as per schema)

```
tasks:  
- name: "Validate structured data"  
  ansible.utils.validate:  
    data: "{{ input_data }}"  
    criteria:  
      - "{{ lookup('file', './criteria.json') | from_json }}"  
    engine: ansible.utils.xxxx
```

- Reads the input JSON data and the criteria for data (schema mode)
- Validate using the 'xxxx' engine
- Returns list of error if data does not conform to the schema criteria

Available validation engines

- **ansible.utils.jsonschema**: Python module to validate json data against a schema

More validation engines in pipeline

Demo on `ansible.utils.validate`

- Configuration changes to rectify drift in operational state
- Reporting about drift to external monitoring tools using Ansible modules and integrations.

Demo on remediation

- Ansible host/group variables validation used as source of truth
- Data passed through validate module to check the data model criteria is passed

```
tasks:  
- name: validate bgp data data with jsonschema bgp model criteria  
  ansible.utils.validate:  
    data: "{{ hostvars }}"  
    criteria:  
      - "{{ lookup('file', './validate/criterias/bgp_input_data_model.json') | from_json }}"  
    engine: ansible.utils.jsonschema  
    register: result
```

Demo on data validation

Under the hood

- **Plugin based architecture:** Loads parser sub plugins from collection *plugin/sub_plugins/cli_parsers* directory

- **Simplified plugin requirements:**

```
class CliParser(CliParserBase):  
    def parse(self, *_args, **kwargs):
```

- **Works with any collection:**

```
tasks:  
- name: Use a custom cli_parser  
  ansible.utils.cli_parse:  
    command: ls -l  
    parser:  
      name: my_organization.my_collection.custom_parser
```

Under the hood

- **Plugin based architecture:** Loads validate sub plugins from collection *plugin/sub_plugins/validate* directory

- **Simplified plugin requirements:**

```
class Validate(ValidateBase):  
    def validate(self, *_args, **kwargs):
```

- **Works with any collection:**

```
tasks:  
- name: Use a custom validate engine  
  ansible.utils.validate:  
    data: "{{ input data }}"  
    criteria:  
      - "{{ lookup('file', './custom_criteria.json | from_json')}}"  
    engine: my_organization.my_collection.custom_validate  
    register: result
```

Thank You

GitHub / IRC: @ganeshrn