Validate data against set criteria with Ansible

The :ref`validate <ansible_collections.ansible.utils.validate_module>` module validates data against your predefined criteria using a validation engine. You can pull this data from a device or file, validate it against your defined criteria, and use the results to identify configuration or operational state drift and optionally take remedial action.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\user_guide\[ansible-devel][docs][docsite][rst][network][user_guide]validate.rst, line 7); *backlink***
>
> Unknown interpreted text role "ref".

Understanding the validate plugin

The ansible.utils collection includes the :ref`validate <ansible_collections.ansible.utils.validate_module>` module.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\user_guide\[ansible-devel][docs][docsite][rst][network][user_guide]validate.rst, line 16); *backlink***
>
> Unknown interpreted text role "ref".

To validate data:

1. Pull in structured data or convert your data to structured format with the :ref`cli_parse <ansible_collections.ansible.utils.cli_parse_module>` module.

   > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\user_guide\[ansible-devel][docs][docsite][rst][network][user_guide]validate.rst, line 20); *backlink***
   >
   > Unknown interpreted text role "ref".

2. Define the criteria to test that data against.
3. Select a validation engine and test the data to see if it is valid based on the selected criteria and validation engine.

The structure of the data and the criteria depends on the validation engine you select. The examples here use the `jsonschema` validation engine provided in the ansible.utils collection.Red Hat Ansible Automation Platform subscription supports limited use if jsonschema public APIs as documented.

Structuring the data

You can pull previously structured data from a file, or use the :ref`cli_parse <ansible_collections.ansible.utils.cli_parse_module>` module to structure your data.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\user_guide\[ansible-devel][docs][docsite][rst][network][user_guide]validate.rst, line 29); *backlink***
>
> Unknown interpreted text role "ref".

The following example fetches the operational state of some network (Cisco NXOS) interfaces and translates that state to structured data using the `ansible.netcommon.pyats` parser.

```
---
- hosts: nxos
  connection: ansible.netcommon.network_cli
  gather_facts: false
  vars:
    ansible_network_os: cisco.nxos.nxos
    ansible_user: "changeme"
    ansible_password: "changeme"

  tasks:
  - name: "Fetch interface state and parse with pyats"
    ansible.utils.cli_parse:
      command: show interface
      parser:
        name: ansible.netcommon.pyats
    register: nxos_pyats_show_interface

  - name: print structured interface state data
    ansible.builtin.debug:
      msg: "{{ nxos_pyats_show_interface['parsed'] }}"
----
```

This results in the following structured data.

```
ok: [nxos] => {
"changed": false,
"parsed": {
    "Ethernet2/1": {
        "admin state": "down",
        "auto mdix": "off",
        "auto negotiate": false,
        "bandwidth": 1000000,
        "beacon": "off"
        <--output omitted-->
    },
    "Ethernet2/10": {
        "admin state": "down",
        "auto mdix": "off",
        "auto negotiate": false,
        "bandwidth": 1000000,
        "beacon": "off",
        <--output omitted-->
    }
  }
}
```

See :ref`cli_parsing` for details on how to parse semi-structured data into structured data.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\user_guide\[ansible-devel][docs][docsite][rst][network][user_guide]validate.rst, line 83); *backlink***
>
> Unknown interpreted text role "ref".

Defining the criteria to validate against

This example uses the jsonschema validation engine to parse the JSON structured data we created in the prior section. the criteria defines the state we want the data to conform to. In this instance, we can validate against a desired admin state of `up` for all the interfaces.

The criteria for `jsonschema` in this example is as follows:

```
$cat criteria/nxos_show_interface_admin_criteria.json
{
      "type" : "object",
      "patternProperties": {
              "^.*": {
                      "type": "object",
                      "properties": {
                              "admin_state": {
                                      "type": "string",
                                      "pattern": "up"
                              }
                      }
              }
      }
}
```

Validating the data

Now that we have the structured data and the criteria, we can validate this data with the :ref:`validate <ansible_collections.ansible.utils.validate_module>` module.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\user_guide\[ansible-devel][docs][docsite][rst][network][user_guide]validate.rst`, line 113); *backlink*
>
> Unknown interpreted text role "ref".

The following tasks check if the current state of the interfaces match the desired state defined in the criteria file.

```
- name: Validate interface admin state
  ansible.utils.validate:
    data: "{{ nxos_pyats_show_interface['parsed'] }}"
    criteria:
      - "{{ lookup('file',  './criteria/nxos_show_interface_admin_criteria.json') | from_json }}"
    engine: ansible.utils.jsonschema
  ignore_errors: true
  register: result

- name: Print the interface names that do not satisfy the desired state
  ansible.builtin.debug:
    msg: "{{ item['data_path'].split('.')[0] }}"
  loop: "{{ result['errors'] }}"
  when: "'errors' in result"
```

In these tasks, we have:

1. Set `data` to the structured JSON data from the :ref:`cli_parse <ansible_collections.ansible.utils.cli_parse_module>` module.

   > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\network\user_guide\[ansible-devel][docs][docsite][rst][network][user_guide]validate.rst`, line 137); *backlink*
   >
   > Unknown interpreted text role "ref".

2. Set `criteria` to the JSON criteria file we defined.
3. Set the validate engine to `jsonschema`.

> **Note**
>
> The value of the criteria option can be a list and should be in a format that is defined by the validation engine used. You need to install the jsonschema on the control node for this example.

The tasks output a list of errors indicating interfaces that do not have admin value in `up` state.

```
TASK [Validate interface for admin state] ***************************************************************************************
fatal: [nxos02]: FAILED! => {"changed": false, "errors": [{"data_path": "Ethernet2/1.admin_state", "expected": "up", "found": "down", "j
...ignoring


TASK [Print the interface names that do not satisfy the desired state] **********************************************************
Monday 14 December 2020  11:05:38 +0530 (0:00:01.661)       0:00:28.676 *******
ok: [nxos] => {
    "msg": "Ethernet2/1"
}
ok: [nxos] => {
    "msg": "Ethernet2/10"
}
```

This shows Ethernet2/1 and Ethernet2/10 are not in the desired state based on the defined criteria. You can create a report or take further action to remediate this to bring the interfaces to the desired state based on the defined criteria.