Ask an Expert: Ansible Network Automation

April 24, 2019

Ajay Chenampara
Solutions Architect
@termlen0

Andrius Benokraitis
Product Manager
@AndriusB
New York, NY
April 30, 2019
NETWORK WORKSHOP

Atlanta, GA
May 2, 2019
May 21, 2019
NETWORK WORKSHOP

Columbus, OH
May 14, 2019
NETWORK WORKSHOP

For more information or to register visit: ansible.com/workshops
WHAT WE’RE TALKING ABOUT TODAY

Writing custom command parsers

Q/A with Ajay and Andrius
What If I Told You
It's Not The Network
ANSIBLE NETWORK ENGINE

A quick introduction
“Automation, when done correctly, is about taking a layered approach. Ansible does its job and gets out of the way”

- Peter Sprygada  
  Distinguished Engineer, Office of the CTO
Ansible Network Engine

- Set of consumable functions distributed as *Ansible Roles* designed to simplify operations
- Automate the provisioning and configuration of *network infrastructure* and *multi cloud network connectivity*.
ANSIBLE NETWORK STACK ARCHITECTURE
ANSIBLE NETWORK STACK ARCHITECTURE

- NETWORK MODULES
- CONNECTION PLUGINS (CLI, API, NETCONF)
- CONFIGURATION
- OPERATIONS
- NETWORK ENGINE
- VIRTUAL NETWORK
- SDN CONTROLLERS
- PHYSICAL DEVICES
- ANSIBLE ENGINE
ANSIBLE NETWORK STACK ARCHITECTURE

- **DEVICE**
- **APPLICATION**
- **CLOUD**
  - NETWORK FUNCTIONS
    - { }
- **CONFIGURATION**
- **OPERATIONS**
  - NETWORK ENGINE
- **NETWORK MODULES**
  - CONNECTION PLUGINS (CLI, API, NETCONF)
  - ANSIBLE ENGINE

**Virtual Network**
**SDN Controllers**
**Physical Devices**
The command parser
I DON'T ALWAYS NEED STRUCTURED DATA

BUT WHEN I DO I GET IT FROM ___FACTS
<table>
<thead>
<tr>
<th>Interface</th>
<th>IP-Address</th>
<th>OK? Method</th>
<th>Status</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet0/0</td>
<td>unassigned</td>
<td>YES</td>
<td>NVRAM up</td>
<td>NVRAM up</td>
</tr>
<tr>
<td>Ethernet0/0.11</td>
<td>10.0.1.38</td>
<td>YES</td>
<td>NVRAM up</td>
<td>NVRAM up</td>
</tr>
<tr>
<td>Ethernet0/1</td>
<td>1.1.1.1</td>
<td>YES</td>
<td>NVRAM up</td>
<td>NVRAM up</td>
</tr>
<tr>
<td>Ethernet0/2</td>
<td>unassigned</td>
<td>YES</td>
<td>NVRAM administratively down</td>
<td>NVRAM administratively down</td>
</tr>
<tr>
<td>Ethernet0/3</td>
<td>unassigned</td>
<td>YES</td>
<td>NVRAM administratively down</td>
<td>NVRAM administratively down</td>
</tr>
<tr>
<td>Loopback0</td>
<td>10.0.1.2</td>
<td>YES</td>
<td>NVRAM up</td>
<td>NVRAM up</td>
</tr>
<tr>
<td>IKE Peer</td>
<td>Type</td>
<td>Dir</td>
<td>Rky</td>
<td>State</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>-------</td>
<td>-----</td>
<td>---------------</td>
</tr>
<tr>
<td>1 209.165.200.225</td>
<td>User</td>
<td>Resp</td>
<td>No</td>
<td>AM_Active</td>
</tr>
<tr>
<td>2 209.165.200.226</td>
<td>User</td>
<td>Resp</td>
<td>No</td>
<td>AM_ACTIVE</td>
</tr>
<tr>
<td>3 209.165.200.227</td>
<td>User</td>
<td>Resp</td>
<td>No</td>
<td>AM_ACTIVE</td>
</tr>
<tr>
<td>4 209.165.200.228</td>
<td>User</td>
<td>Resp</td>
<td>No</td>
<td>AM_ACTIVE</td>
</tr>
</tbody>
</table>

Devices that don’t have “facts” modules (yet)
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
inet6 ::1 prefixlen 128
inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
inet 127.0.0.1 netmask 0xff000000

en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
ether 34:15:9e:27:45:e3
inet6 fe80::3615:9eff:fe27:45e3%en0 prefixlen 64 scopeid 0x4
inet6 2001:db8::3615:9eff:fe27:45e3 prefixlen 64 autoconf
inet 192.0.2.215 netmask 0xfffffe00 broadcast 192.0.2.255
media: autoselect (1000baseT <full-duplex,flow-control>)
status: active

en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
ether 90:84:0d:f6:d1:55
media: <unknown subtype> (<unknown type>)
status: inactive

Non-network command output
What the command parser is ...

- Ansible action plugin
- YAML based regex parser
- Native integration with Ansible Engine
- Allows for import and export of variables to and from playbooks

.... and not

- Pre-written parsers
- Require a new DSL
- Magic...well, maybe a little.
"ip_interface_facts": [
    {
        "Ethernet0/0.11": {
            "data": {
                "admin_state": "up",
                "ip": "10.0.1.38",
                "name": "Ethernet0/0.11",
                "protocol_state": "up"
            }
        }
    },
    {
        "Ethernet0/1": {
            "data": {
                "admin_state": "up",
                "ip": "1.1.1.1",
                "name": "Ethernet0/1",
                "protocol_state": "up"
            }
        }
    }
],

"asa_ikev1_sa_detail": [
    {
        "asa_ikev1_sa_detail": {
            "data": {
                "auth_type": "preshrd",
                "direction": "Resp",
                "encryption": "3des",
                "hash_type": "SHA",
                "lifetime": "86400",
                "peer": "209.165.200.225",
                "rekey": "No",
                "sequence": "1",
                "state": "AM_Active",
                "type": "User"
            }
        }
    }
],

"interface_facts": [
    {
        "en0": {
            "data": {
                "ip": "192.0.2.215",
                "ipv4_address": "192.0.2.215",
                "ipv6_address": "2001:db8::3615:9eff:fe27:45e3",
                "link_status": "active",
                "mtu": "1500"
            }
        }
    },
    {
        "en1": {
            "data": {
                "ip": "1.1.1.1",
                "ipv4_address": null,
                "ipv6_address": null,
                "link_status": "inactive",
                "mtu": "1500"
            }
        }
    }
]
How to write command parsers
Command Parsers are YAML files that look and feel like Ansible playbooks that you know and love.
---
- name: DYNAMIC REPORTING PART
  hosts: cisco
  gather_facts: no
  connection: network_cli

roles:
- ansible-network.network-engine

tasks:
- name: CAPTURE SHOW IP INTERFACE
  ios_command:
    commands:
    - show ip interface brief
  register: output

- name: PARSE THE RAW OUTPUT
  command_parser:
    file: "parsers/ios/show_ip_interface_brief.yaml"
    content: "{{ output.stdout[0] }}"
A workflow

Create a regex pattern to match data

Use the “parser directives” and test

Build and export the structured output
Build the regex

```
^
  (?:(?:St)\s+(\d+\.\d+\.\d+\.\d+))\s+(up|administratively\s+down)\s+(up|down)
```

MATCH INFORMATION

<table>
<thead>
<tr>
<th>Match 1</th>
<th>Ethernet0/1</th>
<th>1.1.1.1</th>
<th>YES NVRAM</th>
<th>up</th>
<th>Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>0-11</td>
<td>Ethernet0/1</td>
<td>1.1.1.1</td>
<td>up</td>
<td>YES</td>
</tr>
<tr>
<td>Group 2</td>
<td>27-34</td>
<td>1.1.1.1</td>
<td>up</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group 3</td>
<td>54-56</td>
<td>up</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group 4</td>
<td>76-78</td>
<td>up</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The parser directives

- pattern_match
- pattern_group
- json_template
- export_facts
Walkthrough of the workflow

Test the regex until you see all the data you are trying to capture, exporting at each stage

```yaml
name: PARSER META DATA
parser metadata:
  version: 1.0
  command: show ip interface brief
  network_os: ios

name: MATCH PATTERN
pattern match:
  regex: "\[(S+)[s+](\d+).\d+\.\d+\]]\d+. *(up|administratively down). *(up|down)
  match_all: yes
  register: section
  export: yes

TASK [debug] *******************************************************
ok: [localhost] => {
  "output": {
    "ansible_facts": {
      "section": [{
        "matches": ["Ethernet0/0.11", "10.0.1.38", "up", "up"]
      }, {
        "matches": ["Ethernet0/1", "1.1.1.1", "up", "up"]
      ]
    }
  }
```
Walkthrough of the workflow

Build out the final data structure and export

```yaml
- name: PARSER META DATA
  parser_metadata:
    version: 1.0
    command: show ip interface brief
    network_os: IOS

- name: MATCH PATTERN
  pattern_match:
    regex: "\s+s+(\d+\.\d+\.\d+\.\d+).*(up|administratively down).*(up|down)"
    match all: yes

- name: GENERATE JSON DATA STRUCTURE
  json_template:
    template:
      - key: "{{ item.matches.0 }}"
        object:
          - key: data
            object:
              - key: name
                value: "{{ item.matches.0 }}"
              - key: ip
                value: "{{ item.matches.1 }}"
              - key: admin state
                value: "{{ item.matches.2 }}"
              - key: protocol state
                value: "{{ item.matches.3 }}"
    export: yes
    register: ip_interface_facts

TASK [debug]
ok: [localhost] => {
  "output": {
    "ansible_facts": {
      "ip_interface_facts": [
        {
          "Ethernet0/0.11": {
            "admin_state": "up",
            "ip": "10.0.1.38",
            "name": "Ethernet0/0.11",
            "protocol_state": "up"
          }
        },
        {
          "Ethernet0/1": {
            "admin_state": "up",
            "ip": "1.1.1.1",
            "name": "Ethernet0/1",
            "protocol_state": "up"
          }
        }
      ]
    }
  }
}````
Call to action!
Contributing to the public parsers repo

https://github.com/network-automation/parser_templates