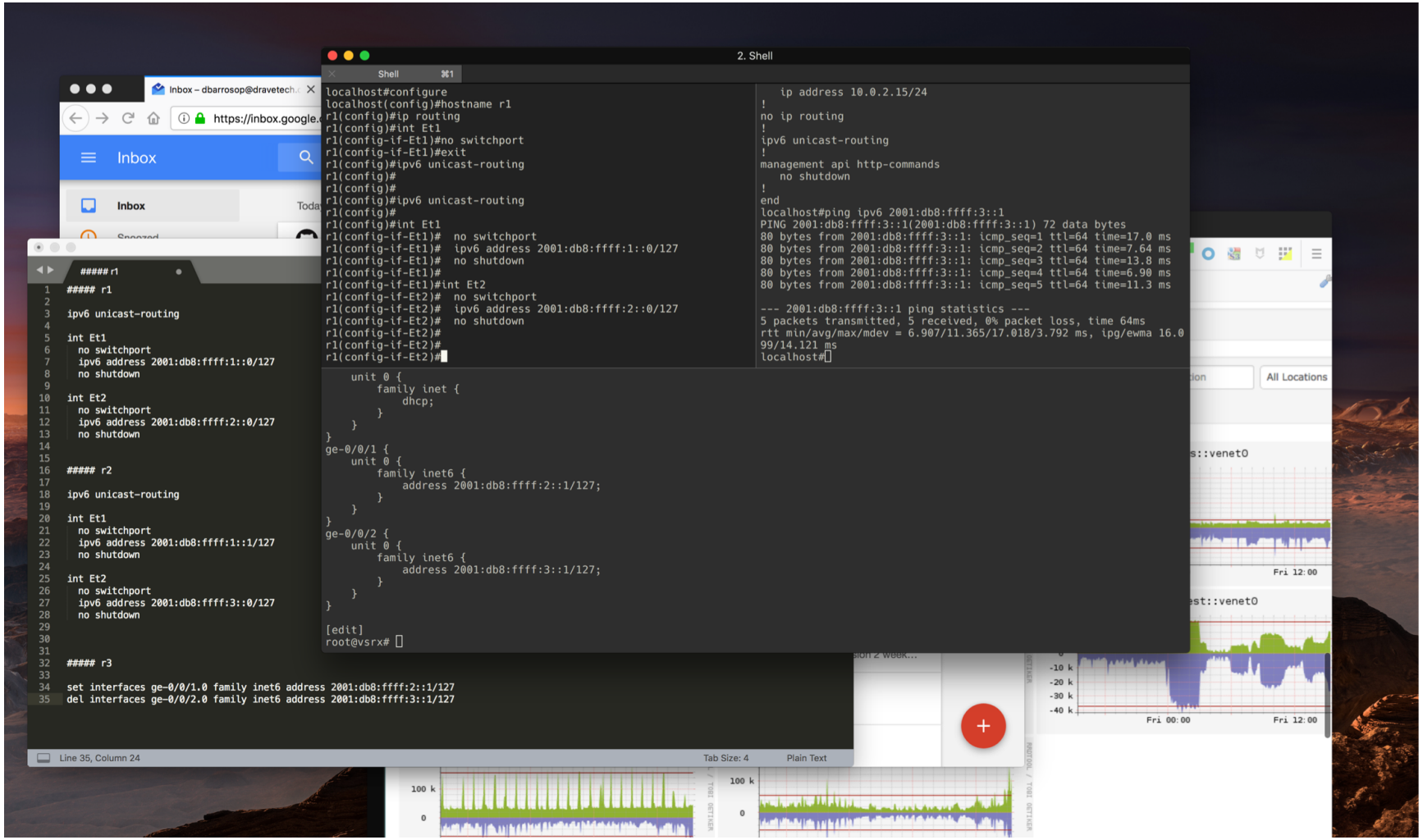
Validating Network Deployments with NAPALM

David Barroso <dbarrosop@dravetech.com>

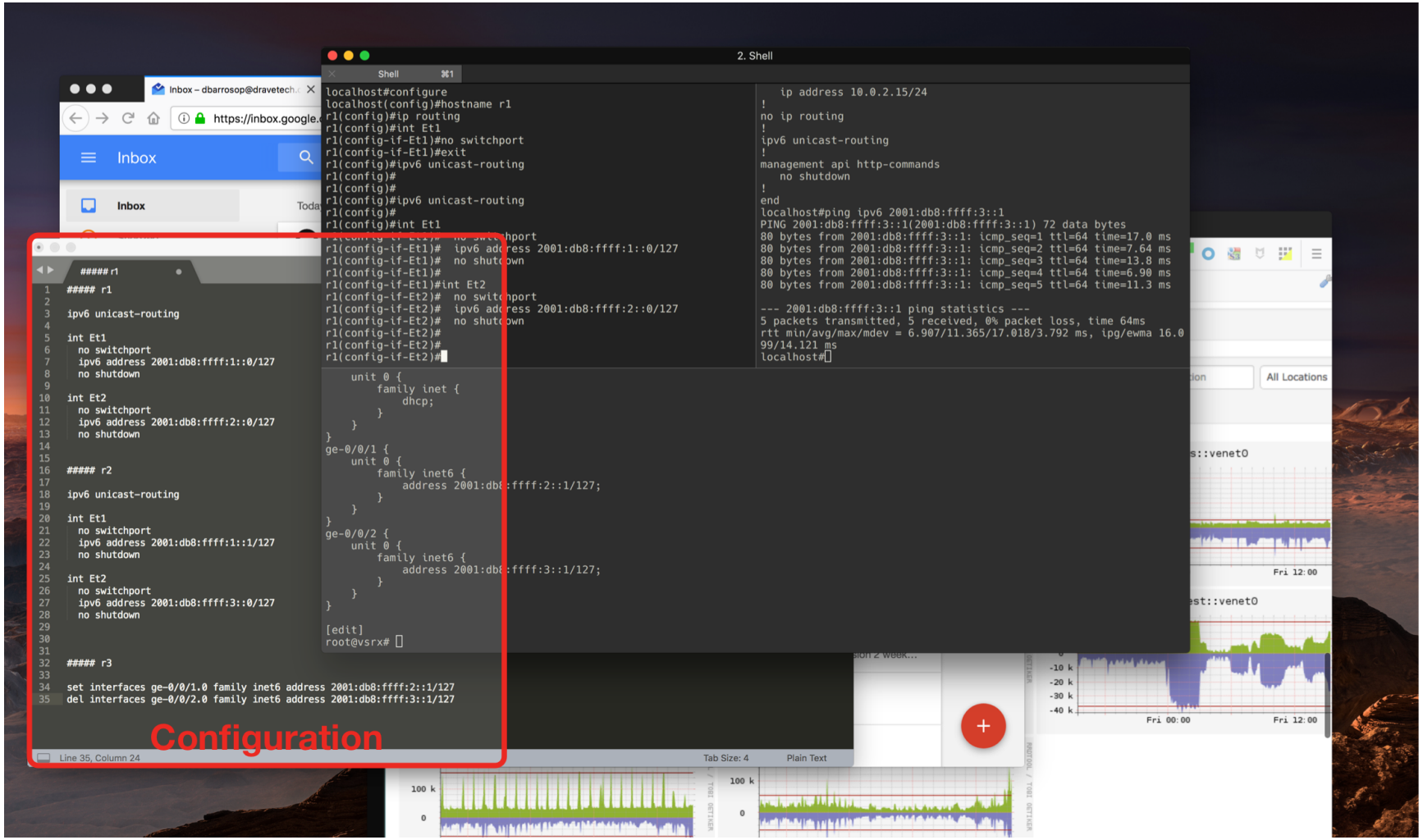
1. Introduction
2. napalm (validate)
3. ansible + napalm (validate)

1. **Introduction**
2. napalm (validate)
3. ansible + napalm (validate)

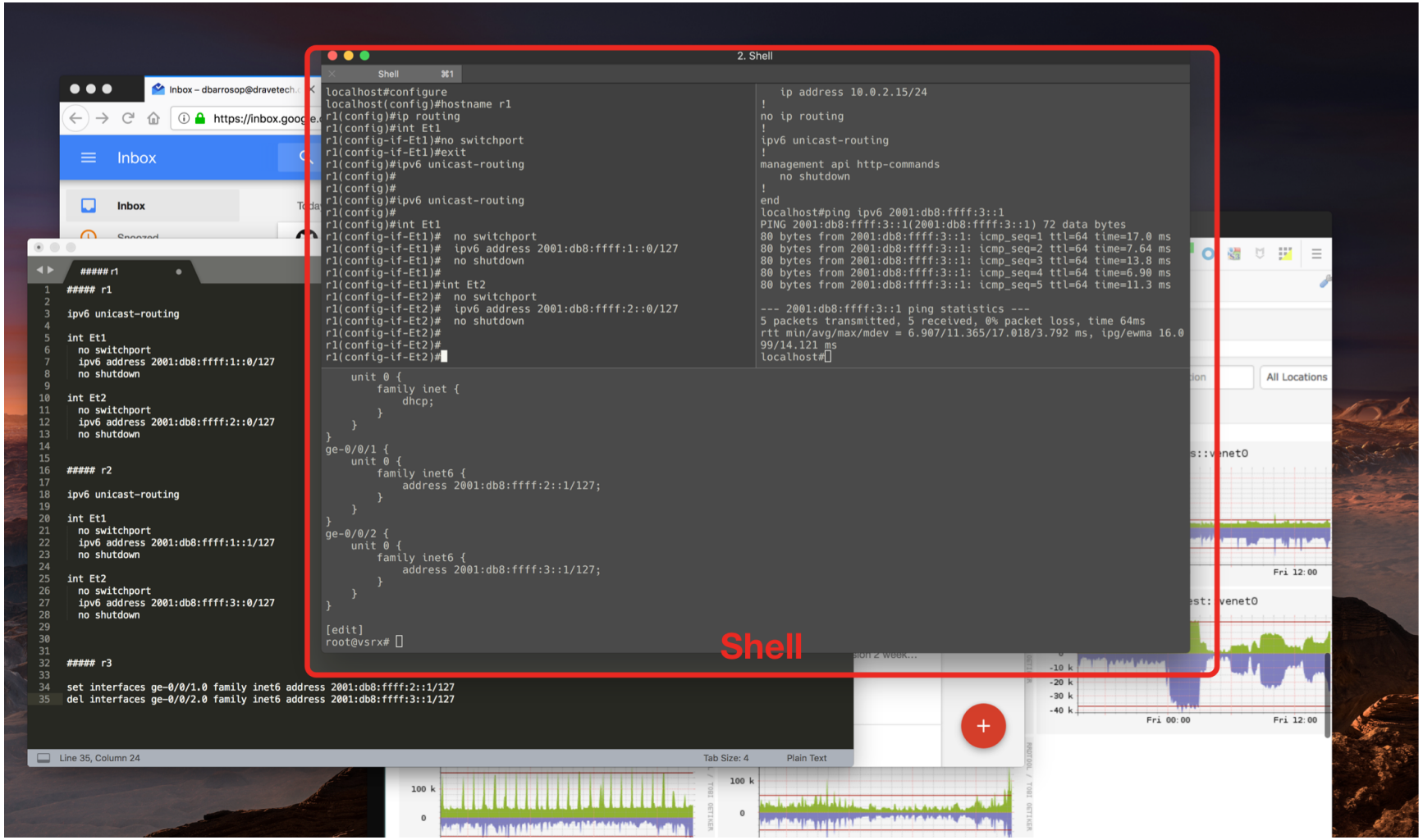
Before Automation



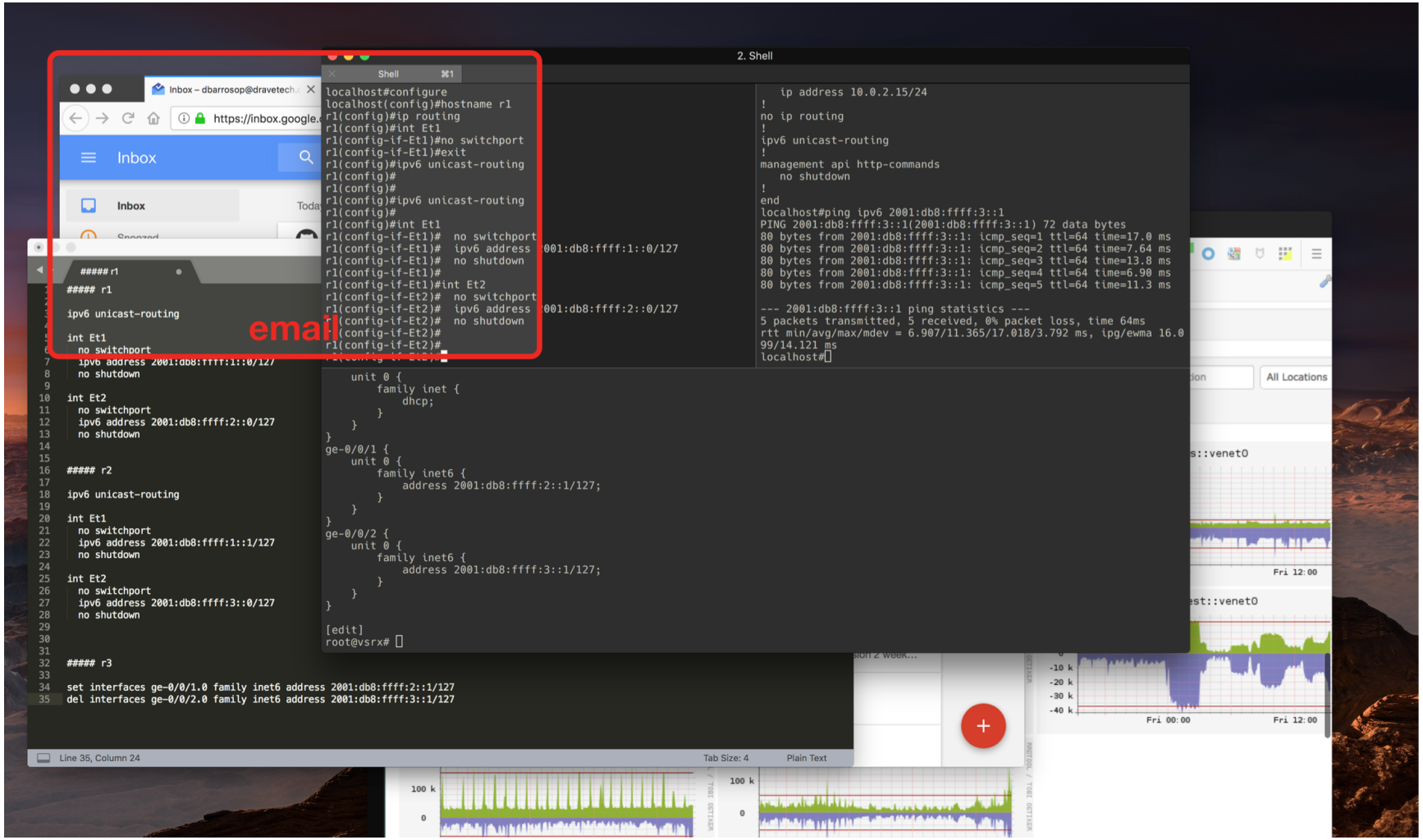
Before Automation



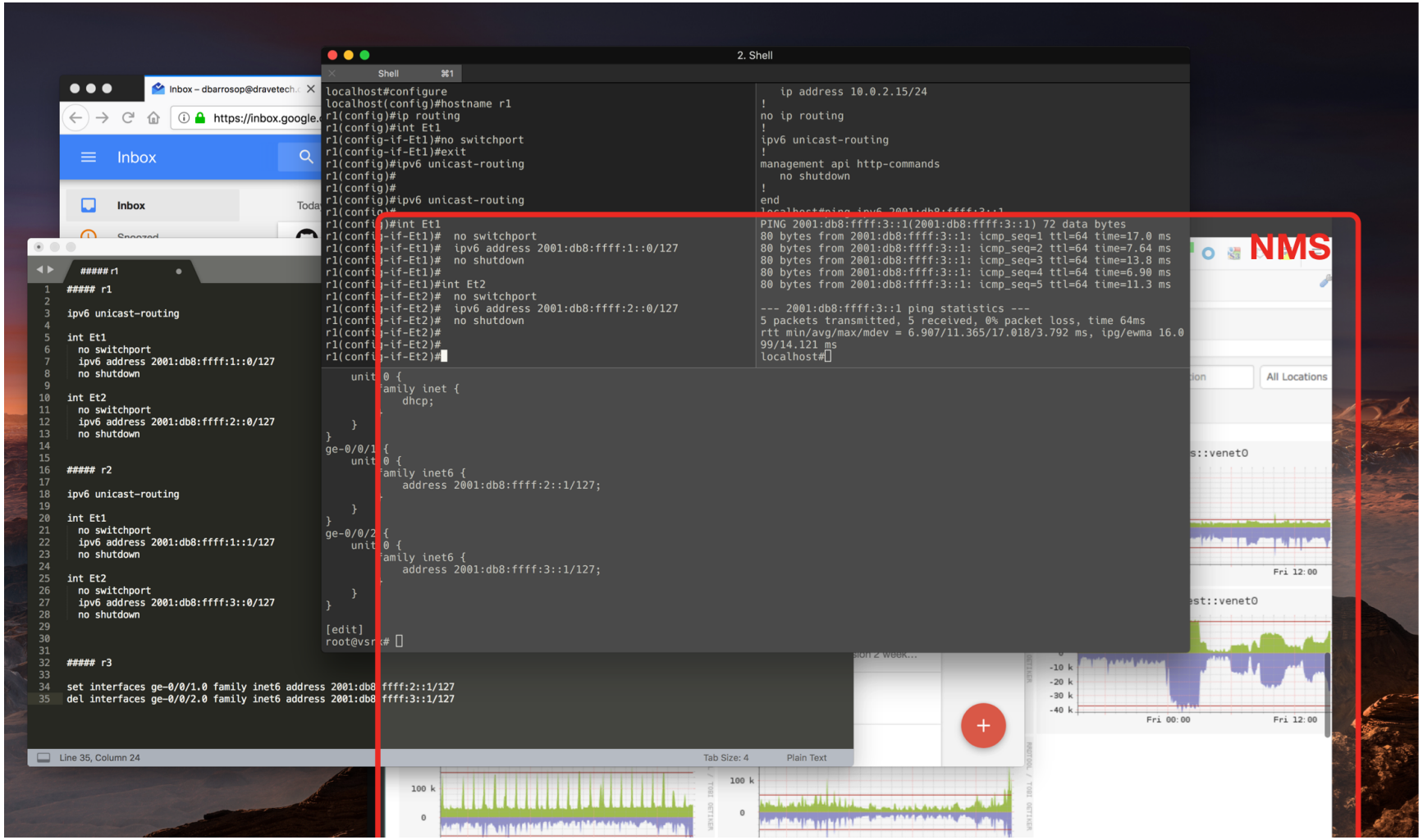
Before Automation



Before Automation



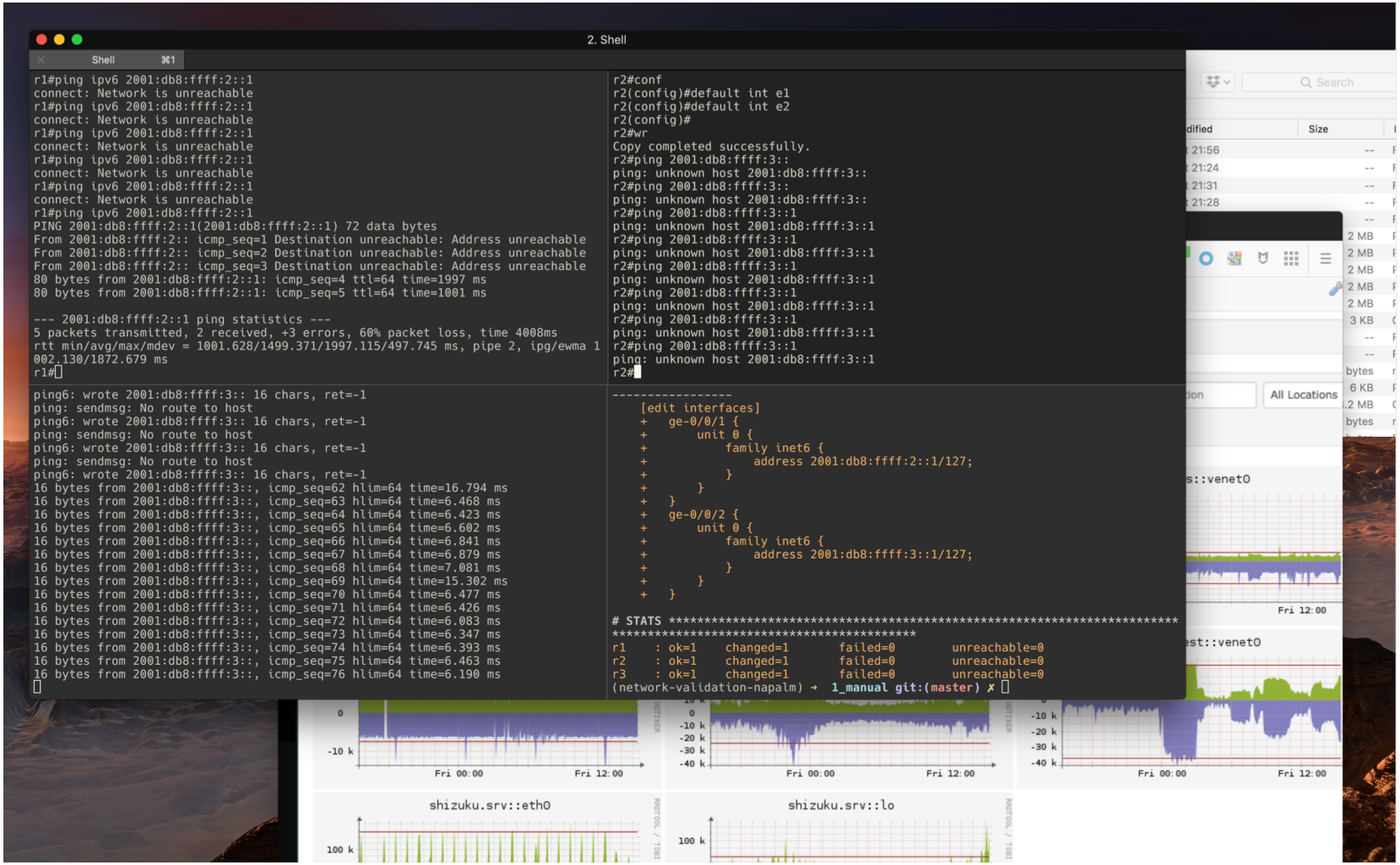
Before Automation



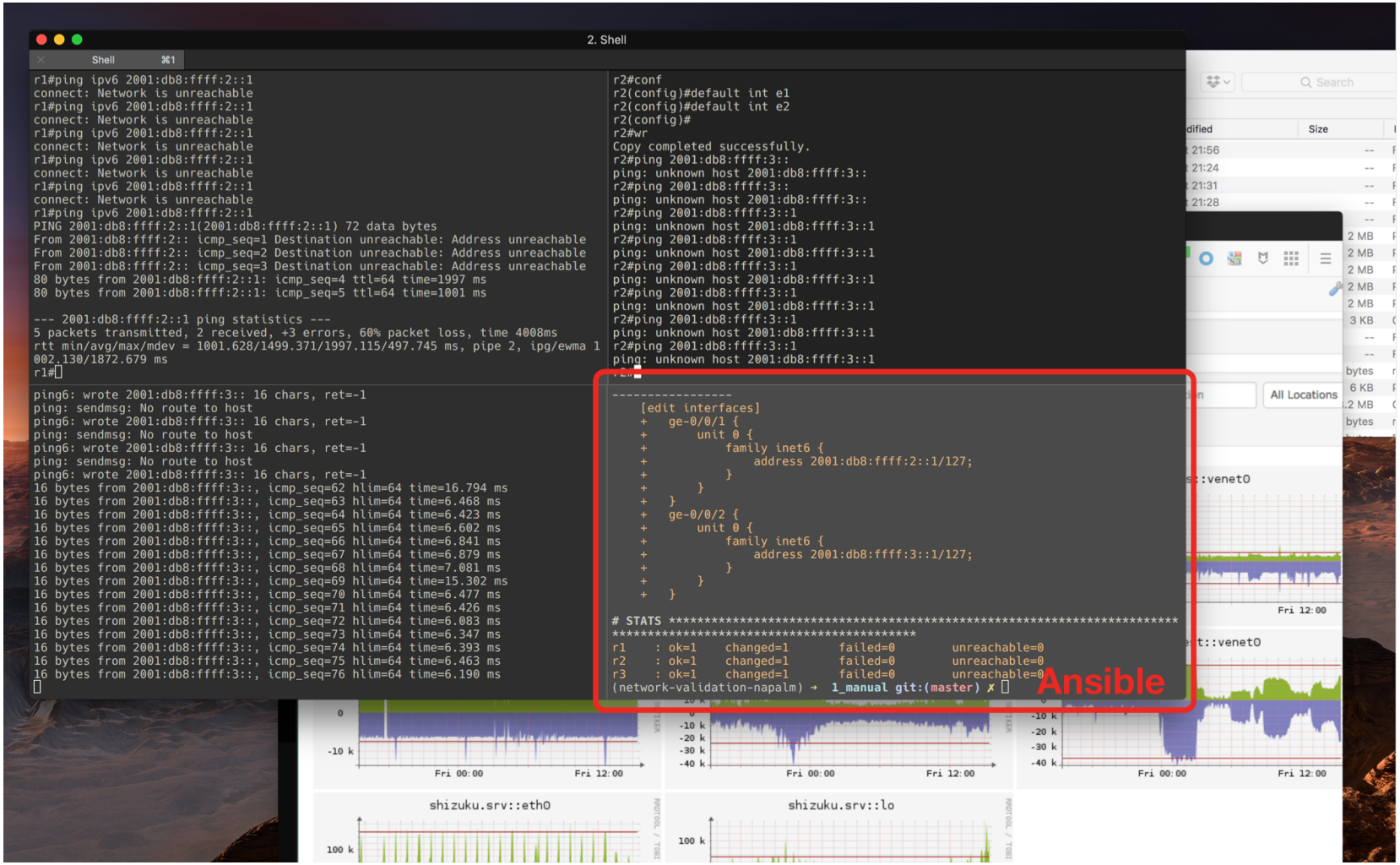
Before Automation

- Manual process end to end
- Immediate feedback
- **Complexity doesn't scale**
  - More devices, harder to check them all
  - More complex services, that harder it becomes to check the right parameters

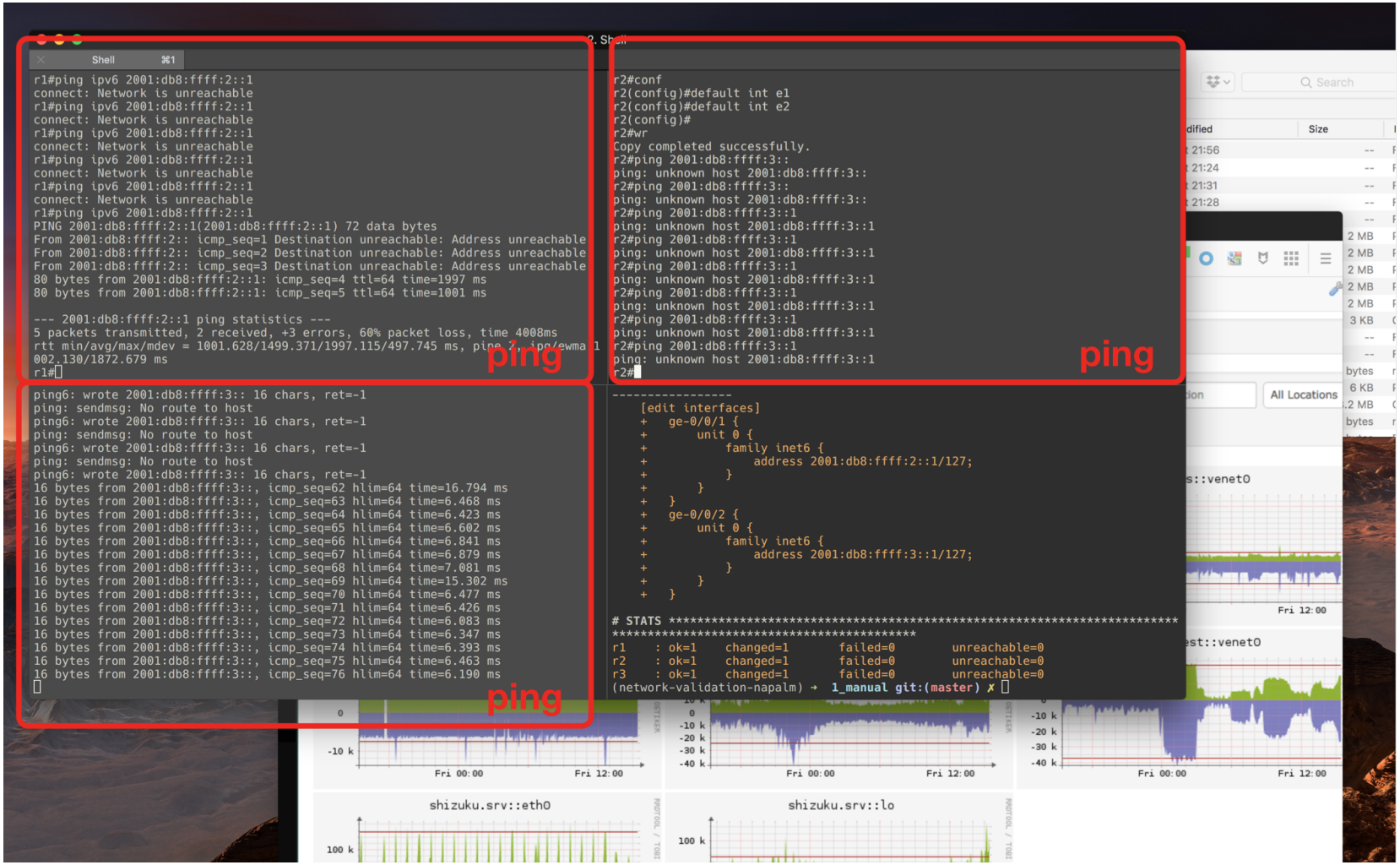
Automation v0.0.1-alpha1



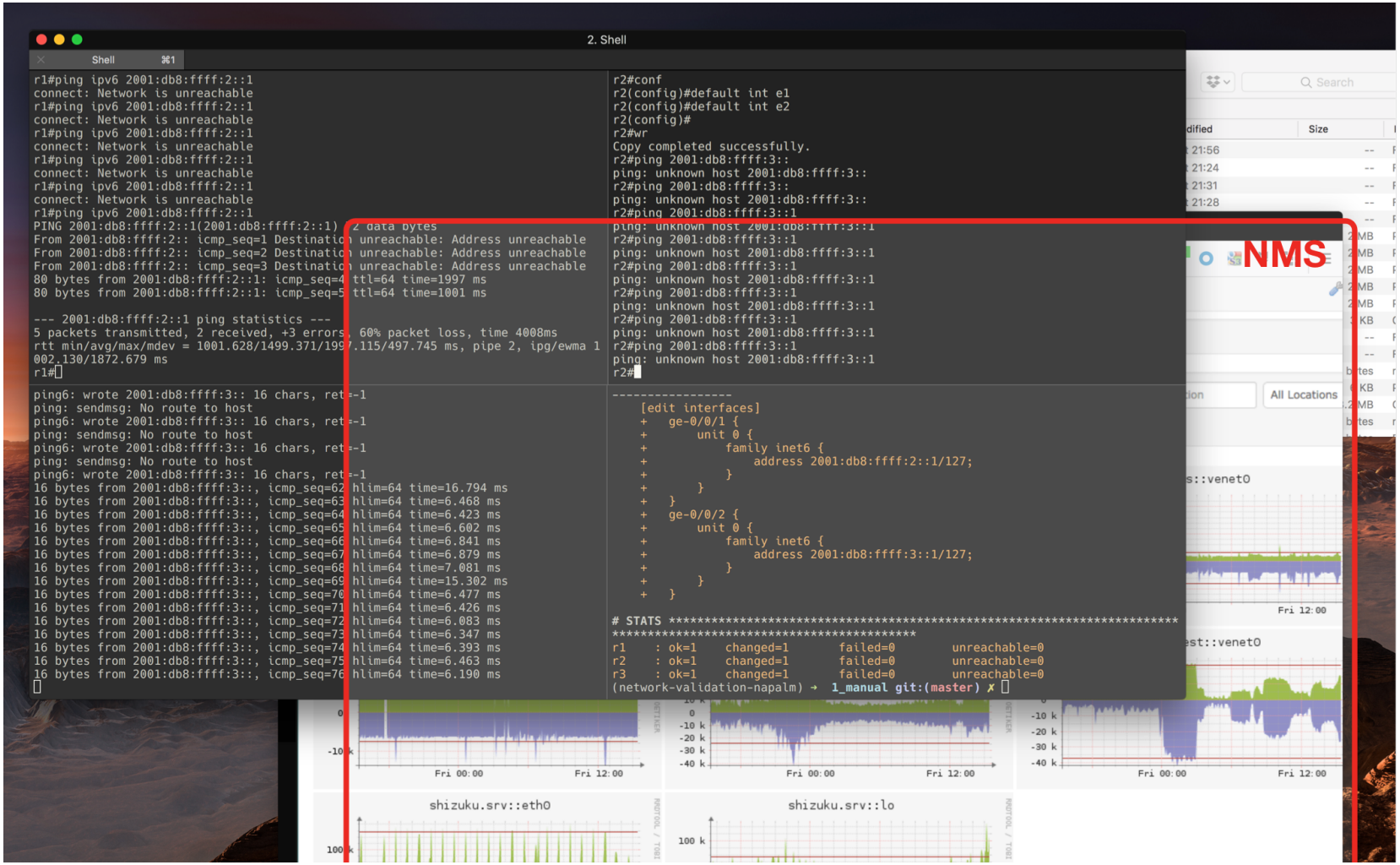
Automation v0.0.1-alpha1



Automation v0.0.1-alpha1



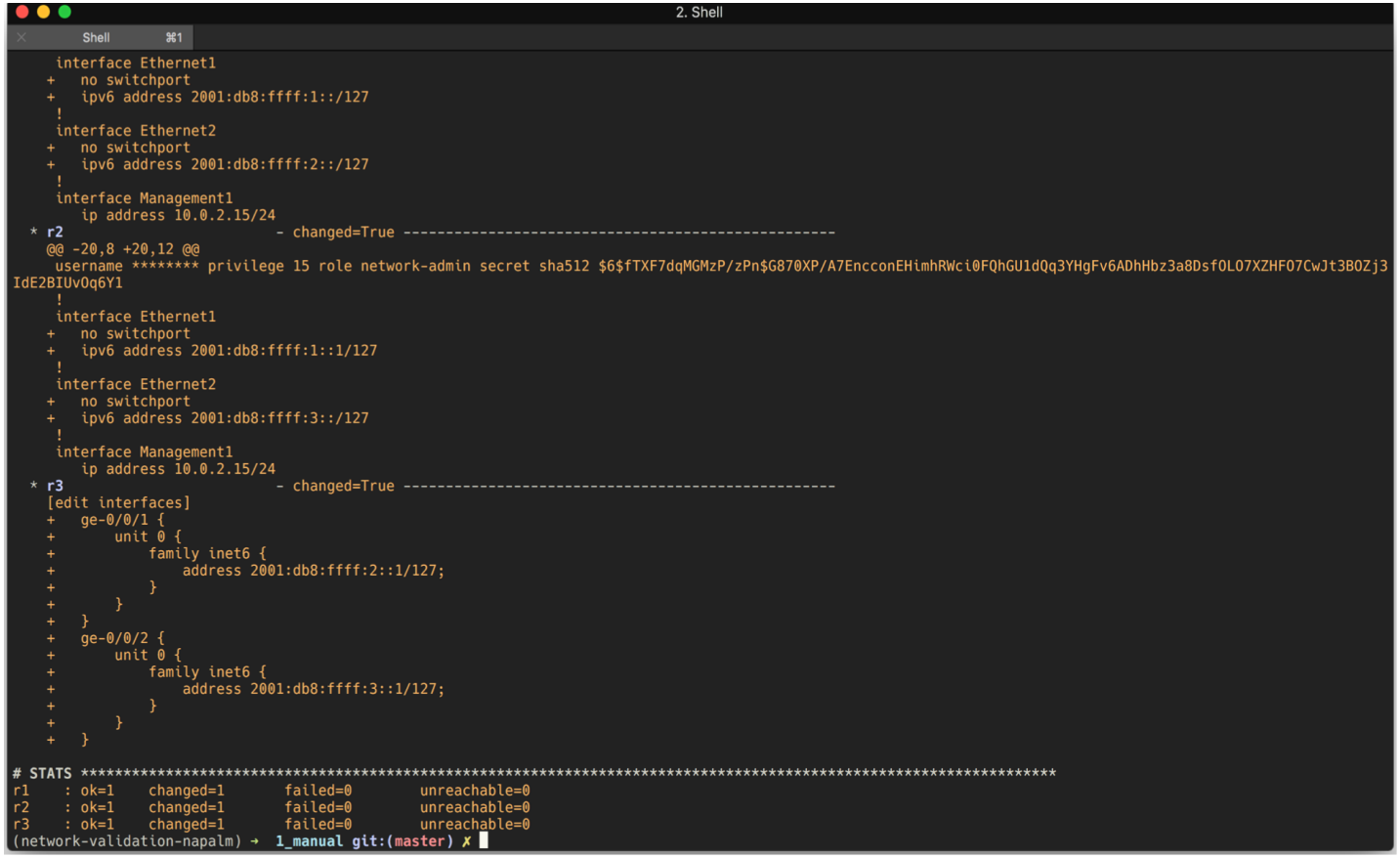
Automation v0.0.1-alpha1



Automation v0.0.1-alpha1

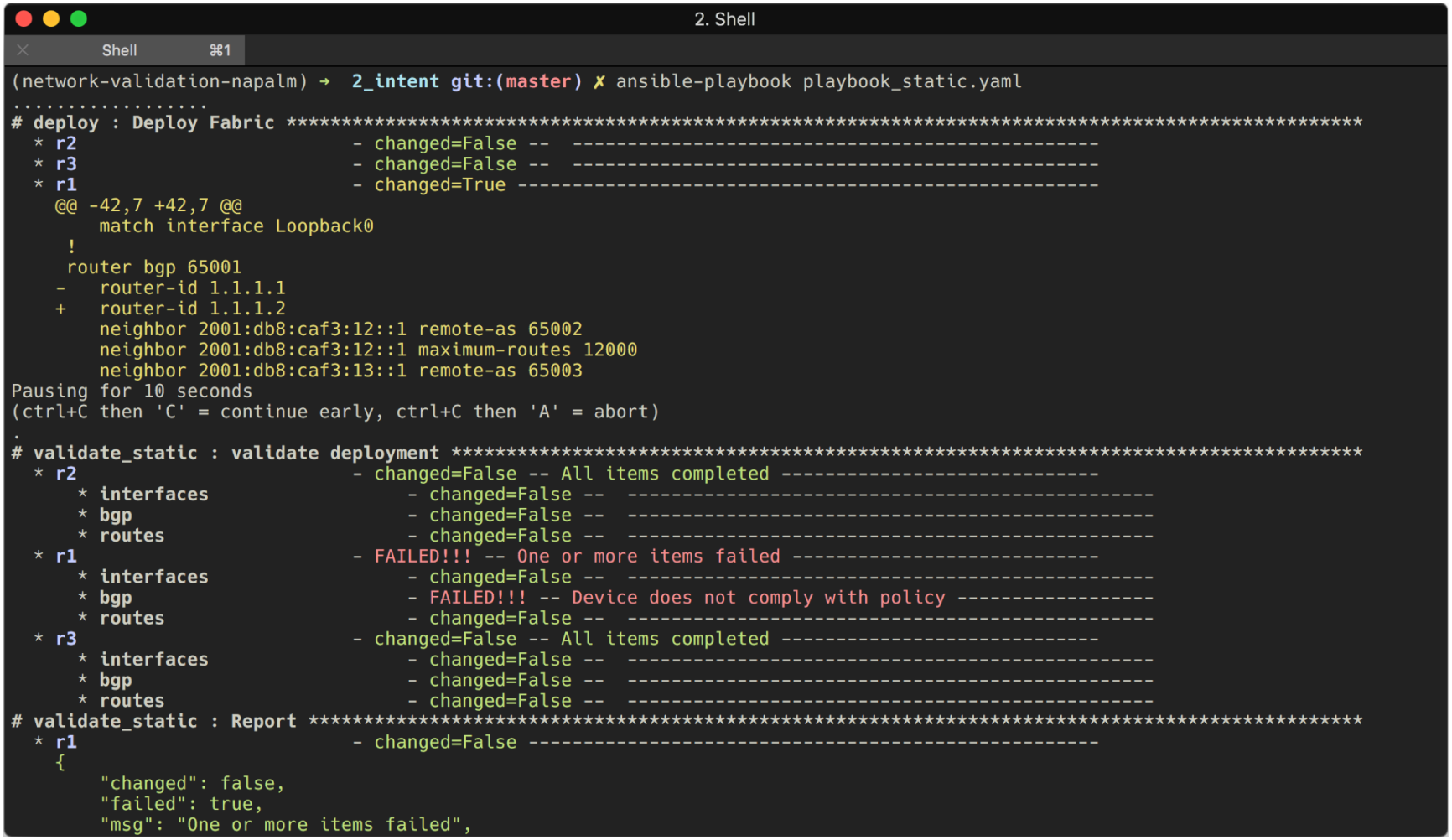
- Half-baked automation process
- Immediate feedback
- **Complexity doesn't scale**
  - More devices, harder to check them all
  - More complex services, that harder it becomes to check the right parameters

Alternative

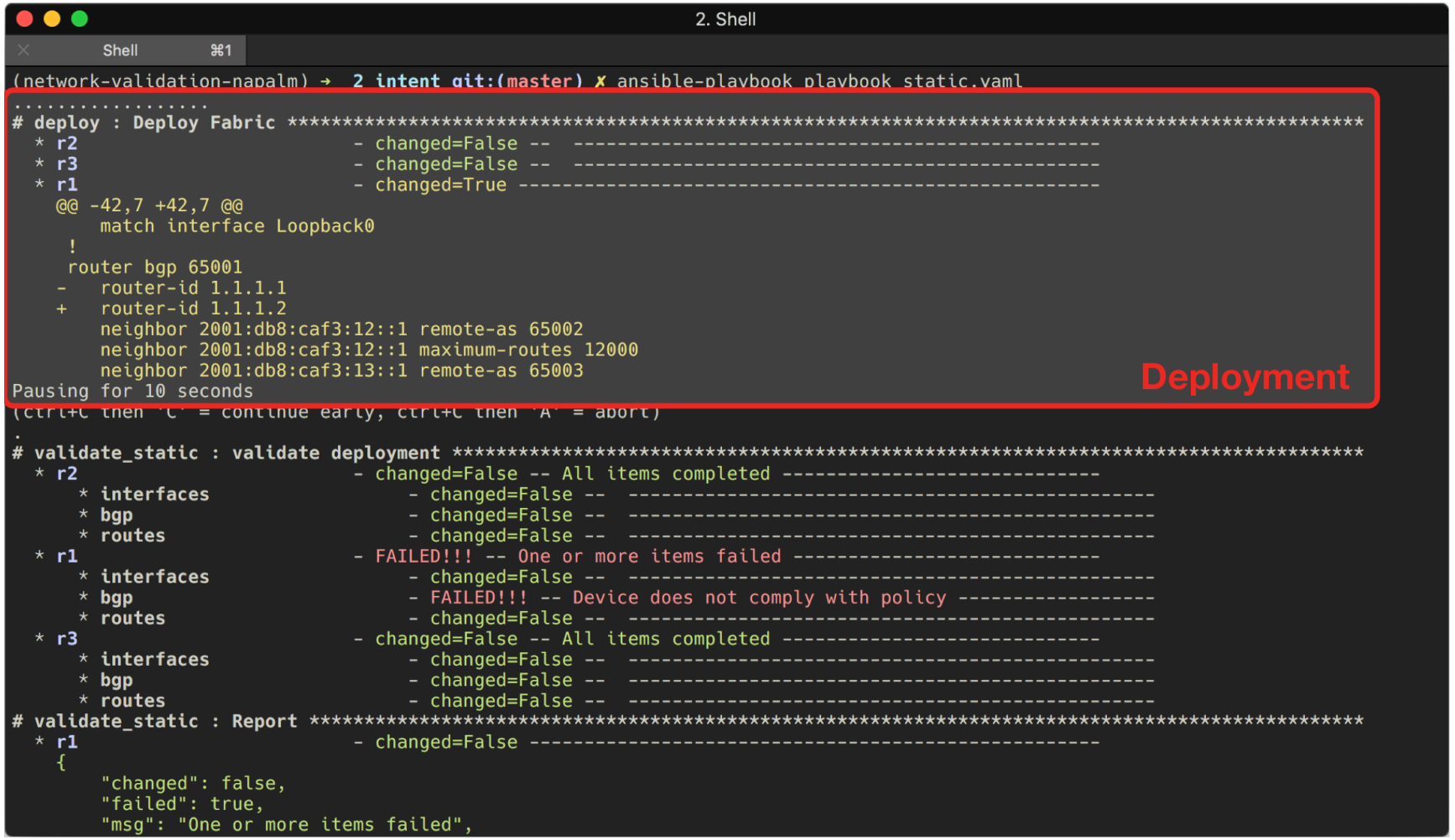


What could possible go wrong?

Fast Forward

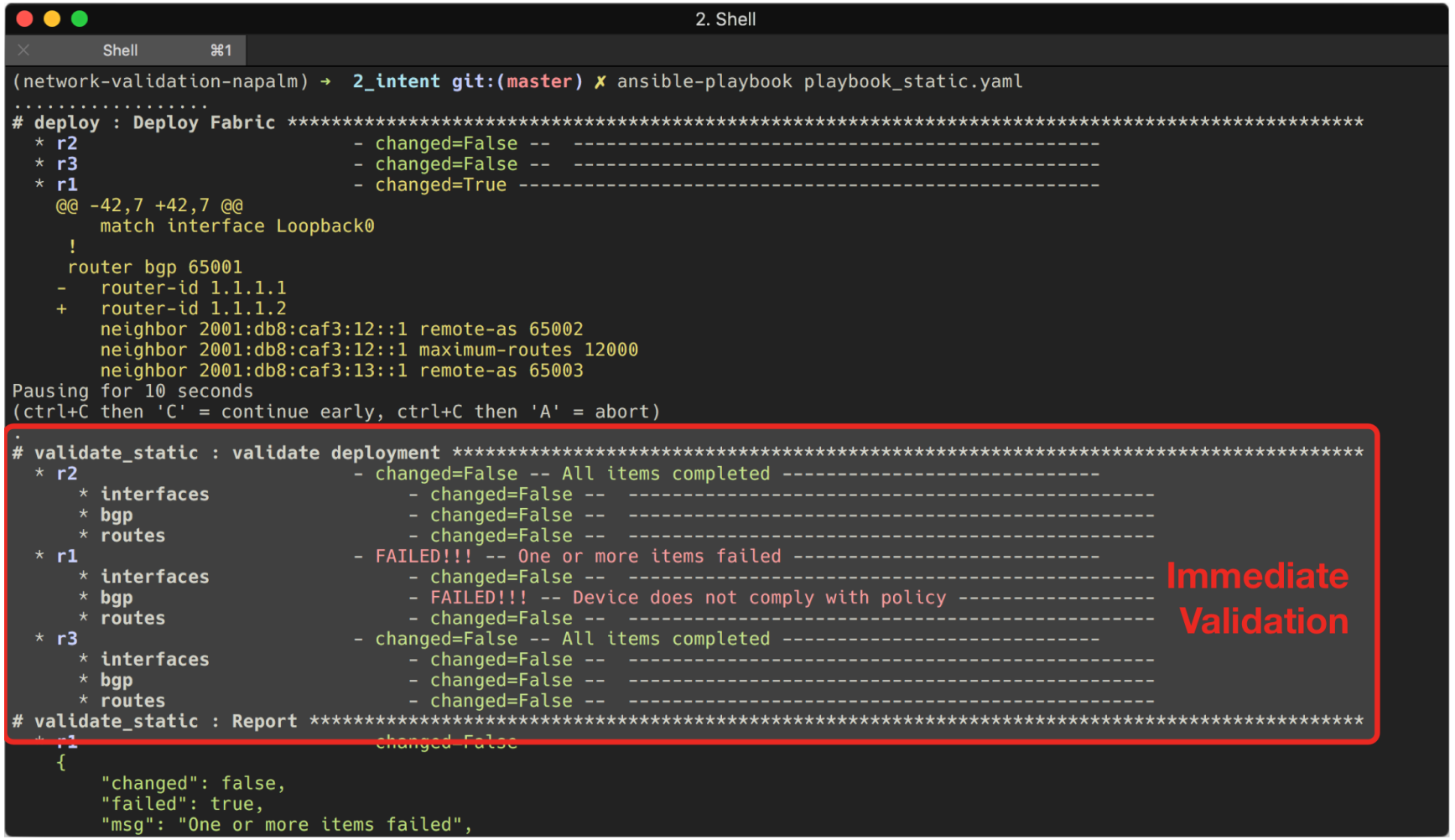


Fast Forward




```
.................
# deploy : Deploy Fabric ***************************************************************************************************
  * r2                          - changed=False --  ------------------------------------------------
  * r3                          - changed=False --  ------------------------------------------------
  * r1                          - changed=True ------------------------------------------------------
    @@ -42,7 +42,7 @@
        match interface Loopback0
     !
     router bgp 65001
    -   router-id 1.1.1.1
    +   router-id 1.1.1.2
        neighbor 2001:db8:caf3:12::1 remote-as 65002
        neighbor 2001:db8:caf3:12::1 maximum-routes 12000
        neighbor 2001:db8:caf3:13::1 remote-as 65003
Pausing for 10 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
.
# validate_static : validate deployment ************************************************************************************
  * r2                          - changed=False -- All items completed -----------------------------
      * interfaces                   - changed=False --  ------------------------------------------------
      * bgp                          - changed=False --  ------------------------------------------------
      * routes                       - changed=False --  ------------------------------------------------
  * r1                          - FAILED!!! -- One or more items failed ----------------------------
      * interfaces                   - changed=False --  ------------------------------------------------
      * bgp                          - FAILED!!! -- Device does not comply with policy ------------------
      * routes                       - changed=False --  ------------------------------------------------
  * r3                          - changed=False -- All items completed -----------------------------
      * interfaces                   - changed=False --  ------------------------------------------------
      * bgp                          - changed=False --  ------------------------------------------------
      * routes                       - changed=False --  ------------------------------------------------
# validate_static : Report *************************************************************************************************
  * r1                          - changed=False ----------------------------------------------------
    {
        "changed": false,
        "failed": true,
        "msg": "One or more items failed",
```

Fast Forward



Welcome to the intent-based hype

1. Introducation
2. **napalm (validate)**
3. ansible + napalm (validate)

napalm-validate

It's part of the napalm library

It allows you build rules to retrieve data from the devices and ensure correctness

It leverages on `getters` so if napalm can get the information it can be validated

Official documentation

napalm-validate (rules)

```
- get_facts:
    hostname: n9k2
    os_version: 7.0.*
    interface_list:
        _mode: strict
        list:
            - Vlan5
            - Vlan40
            - Vlan100,
            - GigabitEthernet0/1
            - GigabitEthernet0/2,
            - Port-channel1]
```

Getter to use. Rules will apply to the data returned by it

napalm-validate (rules)

```
- get_facts:
    hostname: n9k2
    os_version: 7.0.*
    interface_list:
        _mode: strict
        list:
            - Vlan5
            - Vlan40
            - Vlan100,
            - GigabitEthernet0/1
            - GigabitEthernet0/2,
            - Port-channel1]
```

Exact match on the hosntame

napalm-validate (rules)

```
- get_facts:
    hostname: n9k2
    os_version: 7.0.*
    interface_list:
        _mode: strict
        list:
            - Vlan5
            - Vlan40
            - Vlan100,
            - GigabitEthernet0/1
            - GigabitEthernet0/2,
            - Port-channel1]
```

Regular expression against the OS version

napalm-validate (rules)

```
- get_facts:
    hostname: n9k2
    os_version: 7.0.*
    interface_list:
        _mode: strict
        list:
            - Vlan5
            - Vlan40
            - Vlan100,
            - GigabitEthernet0/1
            - GigabitEthernet0/2,
            - Port-channel1]
```

Strict check of a list, `_mode: strict` will report if there is an exact match on elements

napalm-validate (rules)

```
- get_facts:
    hostname: n9k2
    os_version: 7.0.*
    interface_list:
        list:
            - Vlan5
            - Vlan40
            - Vlan100
```

Relaxed check of a list, will report if specified elements are missing but will not report if there are others as well

napalm-validate (BGP rule)

```
- get_bgp_neighbors:
    global:
      peers:
        _mode: strict
        2001:db8:caf3:12::1:
          is_up: yes
          address_family:
            ipv6:
              received_prefixes: '>0'
              sent_prefixes: '>0'
        2001:db8:caf3:13::1:
          is_up: yes
          address_family:
            ipv6:
              received_prefixes: '>0'
              sent_prefixes: '>0'
```

Getter to use

napalm-validate (BGP rule)

```
- get_bgp_neighbors:
    global:
      peers:
        _mode: strict
        2001:db8:caf3:12::1:
          is_up: yes
          address_family:
            ipv6:
              received_prefixes: '>0'
              sent_prefixes: '>0'
        2001:db8:caf3:13::1:
          is_up: yes
          address_family:
            ipv6:
              received_prefixes: '>0'
              sent_prefixes: '>0'
```

We want only these peers present

napalm-validate (BGP rule)

```
- get_bgp_neighbors:
    global:
      peers:
        _mode: strict
        2001:db8:caf3:12::1:
          is_up: yes
          address_family:
            ipv6:
              received_prefixes: '>0'
              sent_prefixes: '>0'
        2001:db8:caf3:13::1:
          is_up: yes
          address_family:
            ipv6:
              received_prefixes: '>0'
              sent_prefixes: '>0'
```

Check we are getting prefixes (`num_prefixes > 0`)

napalm-validate (Routing rule)

```
- get_route_to:
    _name: route to r2
    _kwargs:
      destination: "2001:db8:b33f::2"
      protocol: "bgp"
    2001:db8:b33f::2/128:
      list:
        - next_hop: "2001:db8:caf3:12::1"
        - next_hop: "2001:db8:caf3:13::1"
- get_route_to:
    _name: route to r3
    _kwargs:
      destination: "2001:db8:b33f::3"
      protocol: "bgp"
    2001:db8:b33f::3/128:
      list:
        - next_hop: "2001:db8:caf3:12::1"
        - next_hop: "2001:db8:caf3:13::1"
```

You can add as many rules as you want, you can even repeat the same getter by naming them.

napalm-validate (Routing rule)

```
- get_route_to:
    _name: route to r2
    _kwargs:
      destination: "2001:db8:b33f::2"
      protocol: "bgp"
    2001:db8:b33f::2/128:
      list:
        - next_hop: "2001:db8:caf3:12::1"
        - next_hop: "2001:db8:caf3:13::1"
 - get_route_to:
    _name: route to r3
    _kwargs:
      destination: "2001:db8:b33f::3"
      protocol: "bgp"
    2001:db8:b33f::3/128:
      list:
        - next_hop: "2001:db8:caf3:12::1"
        - next_hop: "2001:db8:caf3:13::1"
```

You can also pass arguments to the getters if needed

napalm-validate (Routing rule)

```
- get_route_to:
    _name: route to r2
    _kwargs:
      destination: "2001:db8:b33f::2"
      protocol: "bgp"
    2001:db8:b33f::2/128:
      list:
        - next_hop: "2001:db8:caf3:12::1"
        - next_hop: "2001:db8:caf3:13::1"
- get_route_to:
    _name: route to r3
    _kwargs:
      destination: "2001:db8:b33f::3"
      protocol: "bgp"
    2001:db8:b33f::3/128:
      list:
        - next_hop: "2001:db8:caf3:12::1"
        - next_hop: "2001:db8:caf3:13::1"
```

Check we have at least those paths with those exact `next_hop`. Other attributes or other potential paths don't matter

How to use it

ansible

```
- name: "validate deployment"
  napalm_validate:
    hostname: "{{ host }}"
    username: "{{ username }}"
    dev_os: "{{ os }}"
    password: "{{ password }}"
    optional_args:
      port: "{{ port }}"
    validation_file: "validate.yaml"
  register: report
```

python

```
from napalm import get_network_driver

eos_driver = get_network_driver("eos")
eos_config = {
    "hostname": "localhost",
    "username": "vagrant",
    "password": "vagrant",
    "optional_args": {"port": 12443},
}

with eos_driver(**eos_config) as eos:
    eos.compliance_report("validate.yml")
```

Result

Not very human-readable but easy to parse programmatically

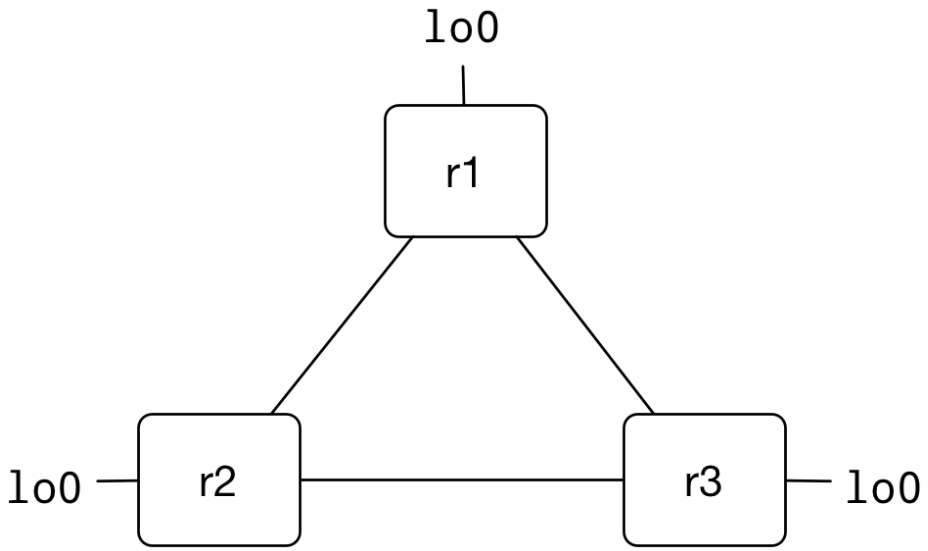

```
        "2001:db8:caf3:12::1": {
         "complies": false,
         "diff": {
          "complies": false,
          "extra": [],
          "missing": [],
          "present": {
             "missing": [],
             "present": {
              "ipv6": {
               "complies": false,
               "diff": {
                "complies": false,
                "extra": [],
                "missing": [],
                "present": {
                 "received_prefixes": {
                  "complies": true,
                  "nested": false
                 },
                 "sent_prefixes": {
                  "actual_value": 0,
                  "complies": false,
                  "expected_value": ">0",
                  "nested": false
```

```
                 "nested": true
                },
                "is_up": {
                 "actual_value": false,
                 "complies": false,
                 "expected_value": true,
                 "nested": false
                }
               }
              },
              "nested": true
             },
             "2001:db8:caf3:13::1": {
              "complies": true,
              "nested": true
             }
            }
           },
           "nested": true
          },
          "router_id": {
           "complies": true,
           "nested": false
          }
         }
```

1. Introducation
2. napalm (validate)
3. **ansible + napalm (validate)**
    1. **Objective**
    2. assertions
    3. static rules
    4. dynamic rules

Objective

To interconnect three devices forming a triangle and ensure BGP sessions come up and loopbacks are redistributed correctly and reachable.



Objective (ansible preparation)

```
roles
├── base
|   ├── tasks/main.yaml
|   |   └── main.yml
|   └── templates
|       ├── eos
|       |   └── simple.j2
|       └── junos
|           └── simple.j2
├── ipfabric
|   ├── tasks/main.yaml
|   └── templates
|       ├── eos
|       |   └── ipfabric.j2
|       └── junos
|           └── ipfabric.j2
└── deploy
    └── tasks/main.yaml
```

Three roles

Objective (ansible preparation)

```
roles
├── base
|   ├── tasks/main.yaml
|   |   └── main.yml
|   └── templates
|       ├── eos
|       |   └── simple.j2
|       └── junos
|           └── simple.j2
├── ipfabric
|   ├── tasks/main.yaml
|   └── templates
|       ├── eos
|       |   └── ipfabric.j2
|       └── junos
|           └── ipfabric.j2
└── deploy
    └── tasks/main.yaml
```

Base configuration; users, snmp, hostname, etc…

Objective (ansible preparation)

```
roles
├── base
|   ├── tasks/main.yaml
|   |   └── main.yml
|   └── templates
|       ├── eos
|       |   └── simple.j2
|       └── junos
|           └── simple.j2
├── ipfabric
|   ├── tasks/main.yaml
|   └── templates
|       ├── eos
|       |   └── ipfabric.j2
|       └── junos
|           └── ipfabric.j2
└── deploy
    └── tasks/main.yaml
```

Interfaces, IP addresses, BGP sessions and policies…

Objective (ansible preparation)

```
roles
├── base
|   ├── tasks/main.yaml
|   |   └── main.yml
|   └── templates
|       ├── eos
|       |   └── simple.j2
|       └── junos
|           └── simple.j2
├── ipfabric
|   ├── tasks/main.yaml
|   └── templates
|       ├── eos
|       |   └── ipfabric.j2
|       └── junos
|           └── ipfabric.j2
└── deploy
    └── tasks/main.yaml
```

Role using napalm to deploy the configuration

Objective (ansible data)

```
---
hostname: r1

asn: 65001
router_id: "1.1.1.1"

interfaces:
  - name: "lo0"
    ip_address: "2001:db8:b33f::1/128"
  - name: "et1"
    ip_address: "2001:db8:caf3:12::/127"
  - name: "et2"
    ip_address: "2001:db8:caf3:13::/127"
peers:
  - ip: "2001:db8:caf3:12::1"
    asn: 65002
  - ip: "2001:db8:caf3:13::1"
    asn: 65003
```

Data to build a fabric for demonstration purposes

Objective (validations)

Three different roles with three different approaches for validation:

validate_assert - using ansible's `assert`
validate_static - using static rules for `napalm_validate`
validate_dynamic - using dynamic rules for `napalm_validate`

1. Introducation
2. napalm (validate)
3. **ansible + napalm (validate)**
    1. Objective
    2. **assertions**
    3. static rules
    4. dynamic rules

Assert (role)

```
- pause:
    seconds: 10
- name: get facts from device
  napalm_get_facts:
    hostname: "{{ host }}"
    username: "{{ username }}"
    dev_os: "{{ os }}"
    password: "{{ password }}"
    optional_args:
      port: "{{ port }}"
    filter: ['bgp_neighbors']
- name: Check BGP sessions are healthy
```

(role continues next slide)

Wait 10 seconds so BGP converges

Assert (role)

```
- pause:
    seconds: 10
- name: get facts from device
  napalm_get_facts:
    hostname: "{{ host }}"
    username: "{{ username }}"
    dev_os: "{{ os }}"
    password: "{{ password }}"
    optional_args:
      port: "{{ port }}"
    filter: ['bgp_neighbors']
- name: Check BGP sessions are healthy
```

(role continues next slide)

Get BGP information from the device with napalm

Assert (role)

```
- name: Check BGP sessions are healthy
  assert:
    that:
      - item.value.is_up
    msg: "{{ item.key }} is down"
  with_dict: "{{ napalm_bgp_neighbors.global.peers }}"
  tags: [print_action]
- name: Check BGP sessions are receiving prefixes
  assert:
    that:
      - item.value.address_family.ipv6.received_prefixes > 0
    msg: "{{ item.key }} is not receiving any prefixes"
  with_dict: "{{ napalm_bgp_neighbors.global.peers }}"
  tags: [print_action]
```

(continues)

Check that all BGP sessions are up

Assert (role)

```
- name: Check BGP sessions are healthy
  assert:
    that:
      - item.value.is_up
    msg: "{{ item.key }} is down"
  with_dict: "{{ napalm_bgp_neighbors.global.peers }}"
  tags: [print_action]
- name: Check BGP sessions are receiving prefixes
  assert:
    that:
      - item.value.address_family.ipv6.received_prefixes > 0
    msg: "{{ item.key }} is not receiving any prefixes"
  with_dict: "{{ napalm_bgp_neighbors.global.peers }}"
  tags: [print_action]
```

(continues)

Check we are receiving prefixes

Assert (example 1)

```
Pausing for 10 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
....
# validate_assert : Check BGP sessions are healthy ********************************
  * r1                          - changed=False -- All items completed -------------
      * 2001:db8:caf3:12::1         - changed=False -- All assertions passed ------
      * 2001:db8:caf3:13::1         - changed=False -- All assertions passed ------
  * r2                          - changed=False -- All items completed -------------
      * 2001:db8:caf3:23::1         - changed=False -- All assertions passed ------
      * 2001:db8:caf3:12::          - changed=False -- All assertions passed ------
  * r3                          - changed=False -- All items completed -------------
      * 2001:db8:caf3:13::          - changed=False -- All assertions passed ------
      * 2001:db8:caf3:23::          - changed=False -- All assertions passed ------
# validate_assert : Check BGP sessions are receiving prefixes *********************
  * r1                          - changed=False -- All items completed -------------
      * 2001:db8:caf3:12::1         - changed=False -- All assertions passed ------
      * 2001:db8:caf3:13::1         - changed=False -- All assertions passed ------
  * r2                          - changed=False -- All items completed -------------
      * 2001:db8:caf3:23::1         - changed=False -- All assertions passed ------
      * 2001:db8:caf3:12::          - changed=False -- All assertions passed ------
  * r3                          - changed=False -- All items completed -------------
      * 2001:db8:caf3:13::          - changed=False -- All assertions passed ------
      * 2001:db8:caf3:23::          - changed=False -- All assertions passed ------

# STATS ***************************************************************************
r1     : ok=11       changed=0       failed=0       unreachable=0
r2     : ok=10       changed=0       failed=0       unreachable=0
r3     : ok=10       changed=0       failed=0       unreachable=0
```

All green

Assert (example 2)

```
Pausing for 10 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
....
# validate_assert : Check BGP sessions are healthy ********************************
  * r1                          - FAILED!!! -- One or more items failed ------------
      * 2001:db8:caf3:12::1          - FAILED!!! -- 2001:db8:caf3:12::1 is down ----
      * 2001:db8:caf3:13::1          - changed=False -- All assertions passed ------
  * r2                          - FAILED!!! -- One or more items failed ------------
      * 2001:db8:caf3:23::1          - changed=False -- All assertions passed ------
      * 2001:db8:caf3:12::           - FAILED!!! -- 2001:db8:caf3:12:: is down -----
  * r3                          - changed=False -- All items completed -------------
      * 2001:db8:caf3:13::           - changed=False -- All assertions passed ------
      * 2001:db8:caf3:23::           - changed=False -- All assertions passed ------
# validate_assert : Check BGP sessions are receiving prefixes *********************
  * r3                          - changed=False -- All items completed -------------
      * 2001:db8:caf3:13::           - changed=False -- All assertions passed ------
      * 2001:db8:caf3:23::           - changed=False -- All assertions passed ------

# STATS ***************************************************************************
r1     : ok=9        changed=0       failed=1        unreachable=0
r2     : ok=8        changed=0       failed=1        unreachable=0
r3     : ok=10       changed=0       failed=0        unreachable=0
```

Something failed, note tests got interrupted

1. Introducation
2. napalm (validate)
3. **ansible + napalm (validate)**
    1. Objective
    2. assertions
    3. **static rules**
    4. dynamic rules

Static (rules)

```
2_intent/roles/validate_static/
├── files
│   ├── r1
│   │   ├── bgp.yaml
│   │   ├── interfaces.yaml
│   │   └── routes.yaml
│   ├── r2
│   │   ├── bgp.yaml
│   │   ├── interfaces.yaml
│   │   └── routes.yaml
│   └── r3
│       ├── bgp.yaml
│       ├── interfaces.yaml
│       └── routes.yaml
└── tasks
    └── main.yaml
```

Each hosts gets its own set of rules

Static (rules)

```
2_intent/roles/validate_static/
├── files
│   ├── r1
│   │   ├── bgp.yaml
│   │   ├── interfaces.yaml
│   │   └── routes.yaml
│   ├── r2
│   │   ├── bgp.yaml
│   │   ├── interfaces.yaml
│   │   └── routes.yaml
│   └── r3
│       ├── bgp.yaml
│       ├── interfaces.yaml
│       └── routes.yaml
└── tasks
    └── main.yaml
```

We divide the rules per purpose (optional)

Static (Interfaces rule)

```
---
- get_interfaces_ip:
    Ethernet1:
      ipv6:
        2001:db8:caf3:12:::
          prefix_length: 127
    Ethernet2:
      ipv6:
        2001:db8:caf3:13:::
          prefix_length: 127
    Loopback0:
      ipv6:
        2001:db8:b33f::1:
          prefix_length: 128
```

Simple check to guarantee interfaces have the correct IPs

Note, this is tailored for one of the routers, the other routers will have this data tailored for them as well

Static (BGP rule)

```
---
- get_bgp_neighbors:
    global:
      router_id: 1.1.1.1
      peers:
        _mode: strict
        2001:db8:caf3:12::1:
          is_up: yes
          address_family:
            ipv6:
              received_prefixes: '>0'
              sent_prefixes: '>0'
        2001:db8:caf3:13::1:
          is_up: yes
          address_family:
            ipv6:
              received_prefixes: '>0'
              sent_prefixes: '>0'
```

We check we have the right peers, that they are up and sending and receiving prefixes

Note, this is tailored for one of the routers, the other routers will have this data tailored for them as well

Static (Routing rule)

```
---
- get_route_to:
    _name: route to r2
    _kwargs:
      destination: "2001:db8:b33f::2"
      protocol: "bgp"
    2001:db8:b33f::2/128:
      list:
        - next_hop: "2001:db8:caf3:12::1"
        - next_hop: "2001:db8:caf3:13::1"

- get_route_to:
    _name: route to r3
    _kwargs:
      destination: "2001:db8:b33f::3"
      protocol: "bgp"
    2001:db8:b33f::3/128:
      list:
        - next_hop: "2001:db8:caf3:12::1"
        - next_hop: "2001:db8:caf3:13::1"
```

We check we have all paths to the rest of the loopbacks via the other two routers

Note, this is tailored for one of the routers, the other routers will have this data tailored for them as well

Static (role)

```
- pause:
    seconds: 10
- block:
  - name: "validate deployment"
    napalm_validate:
      hostname: "{{ host }}"
      username: "{{ username }}"
      dev_os: "{{ os }}"
      password: "{{ password }}"
      optional_args:
        port: "{{ port }}"
      validation_file: "{{ role_path }}/files/{{ inventory_hostname }}/{{ item }}.y
    register: report
    tags: [print_action]
    with_items: ["interfaces", "bgp", "routes"]
  rescue:
    - name: "Report"
      debug:
        msg: "{{ report|to_nice_json }}"
      tags: [print_action]
```

Wait 10 seconds so BGP converges

Static (role)

```
- pause:
    seconds: 10
- block:
  - name: "validate deployment"
    napalm_validate:
      hostname: "{{ host }}"
      username: "{{ username }}"
      dev_os: "{{ os }}"
      password: "{{ password }}"
      optional_args:
        port: "{{ port }}"
      validation_file: "{{ role_path }}/files/{{ inventory_hostname }}/{{ item }}.y
    register: report
    tags: [print_action]
    with_items: ["interfaces", "bgp", "routes"]
  rescue:
    - name: "Report"
      debug:
        msg: "{{ report|to_nice_json }}"
      tags: [print_action]
```

Load validation rules for each case

Static (role)

```
- pause:
    seconds: 10
- block:
  - name: "validate deployment"
    napalm_validate:
      hostname: "{{ host }}"
      username: "{{ username }}"
      dev_os: "{{ os }}"
      password: "{{ password }}"
      optional_args:
        port: "{{ port }}"
      validation_file: "{{ role_path }}/files/{{ inventory_hostname }}/{{ item }}.y
    register: report
    tags: [print_action]
    with_items: ["interfaces", "bgp", "routes"]
  rescue:
    - name: "Report"
      debug:
        msg: "{{ report|to_nice_json }}"
      tags: [print_action]
```

If `napalm_validate` fails print it as `nice_json`

Static (example 1)

```
Pausing for 10 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
.
# validate_static : validate deployment ***************************************
  * r2                          - changed=False -- All items completed ------------
      * interfaces                   - changed=False --  -------------------------
      * bgp                          - changed=False --  -------------------------
      * routes                       - changed=False --  -------------------------
  * r1                          - changed=False -- All items completed ------------
      * interfaces                   - changed=False --  -------------------------
      * bgp                          - changed=False --  -------------------------
      * routes                       - changed=False --  -------------------------
  * r3                          - changed=False -- All items completed ------------
      * interfaces                   - changed=False --  -------------------------
      * bgp                          - changed=False --  -------------------------
      * routes                       - changed=False --  -------------------------

# STATS ***********************************************************************
r1     : ok=9        changed=0       failed=0        unreachable=0
r2     : ok=8        changed=0       failed=0        unreachable=0
r3     : ok=8        changed=0       failed=0        unreachable=0
```

All green!

Static (example 2)

```
Pausing for 10 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
.
# validate_static : validate deployment ***************************************
  * r2                          - FAILED!!! -- One or more items failed ----------
      * interfaces                  - changed=False --  -------------------------
      * bgp                         - FAILED!!! -- Device does not comply with poli
      * routes                      - FAILED!!! -- Device does not comply with poli
  * r1                          - FAILED!!! -- One or more items failed ----------
      * interfaces                  - changed=False --  -------------------------
      * bgp                         - FAILED!!! -- Device does not comply with poli
      * routes                      - FAILED!!! -- Device does not comply with poli
  * r3                          - FAILED!!! -- One or more items failed ----------
      * interfaces                  - FAILED!!! -- Device does not comply with poli
      * bgp                         - changed=False --  -------------------------
      * routes                      - FAILED!!! -- Device does not comply with poli
# validate_static : Report ****************************************************
  * r1                          - changed=False ----------------------------------
    {
        "changed": false,
        "failed": true,
        "msg": "One or more items failed",
        "results": [
            {
                "_ansible_item_result": true,
                "_ansible_no_log": false,
                "_ansible_parsed": true,
                "changed": false,
                "compliance_report": {
                    "complies": true,
```

We can see which reports actually failed, not all of them did

Static (example 2)


```
        "2001:db8:caf3:12::1": {
         "complies": false,
         "diff": {
          "complies": false,
          "extra": [],
          "missing": [],
          "present": {
           "address_family": {
            "complies": false,
            "diff": {
             "complies": false,
             "extra": [],
             "missing": [],
             "present": {
              "ipv6": {
               "complies": false,
               "diff": {
                "complies": false,
                "extra": [],
                "missing": [],
                "present": {
                 "received_prefixes": {
                  "complies": true,
                  "nested": false
                 },
                 "sent_prefixes": {
                  "actual_value": 0,
                  "complies": false,
                  "expected_value": ">0",
                  "nested": false
```


EXAMPLE: If we inspect the report we can see that peer `2001:db8:caf3:12::1` is receiving prefixes but not sending any

1. Introducation
2. napalm (validate)
3. **ansible + napalm (validate)**
    1. Objective
    2. assertions
    3. static rules
    4. **dynamic rules**

Dynamic rules

Pretty much like static rules

However, instead of having static files per device you have a generic template that resolves the data

Those templates will be similar to the equivalent to configure the service

Dynamic rules (example)

data

```
interfaces:
  - name: "Loopback0"
    ip_address: "2001:db8:b33f::1/128"
  - name: "Ethernet1"
    ip_address: "2001:db8:caf3:12::/127"
  - name: "Ethernet2"
    ip_address: "2001:db8:caf3:13::/127"
```

template for interfaces

```
{% for interface in interfaces %}

default interface {{ interface.name }}
interface {{ interface.name }}
  {{ 'no switchport' if not interface.name.startswith("lo") else "" }}
  ipv6 address {{ interface.ip_address }}
  ipv6 address fe80::{{ inventory_hostname[-1] }}/64 link-local

{% endfor %}
```

rules for interfaces (static)

```
---
- get_interfaces_ip:
    Ethernet1:
      ipv6:
        2001:db8:caf3:12:::
          prefix_length: 127
    Ethernet2:
      ipv6:
        2001:db8:caf3:13:::
          prefix_length: 127
    Loopback0:
      ipv6:
        2001:db8:b33f::1:
          prefix_length: 128
```

rules for interfaces (dynamic)

```
---
- get_interfaces_ip:
{% for interface in interfaces %}
{{ interface.name }}:
      ipv6:
        {{ interface.ip_address.split("/")[0] }}:
          prefix_length: {{ interface.ip_address.split("/")[1] }}
{% endfor %}
```

Dynamic rules (exercise for the reader)

Create a role similar to `/2_intent/roles/validate_static` but using dynamic rules

Comparison (assert)

assert

```
- name: Check BGP sessions are healthy
  assert:
    that:
      - item.value.is_up
    msg: "{{ item.key }} is down"
  with_dict: "{{ napalm_bgp_neighbors.global.peers }}"
  tags: [print_action]
- name: Check BGP sessions are receiving prefixes
  assert:
    that:
      - item.value.address_family.ipv6.received_prefixes > 0
    msg: "{{ item.key }} is not receiving any prefixes"
  with_dict: "{{ napalm_bgp_neighbors.global.peers }}"
  tags: [print_action]
```

cumbersome to write

the more complex the data the more cumbersome it will become

if a test fails, rest won't run

Comparison (static rules)

static rules

```
---
- get_interfaces_ip:
    Ethernet1:
      ipv6:
        2001:db8:caf3:12:::
          prefix_length: 127
    Ethernet2:
      ipv6:
        2001:db8:caf3:13:::
          prefix_length: 127
    Loopback0:
      ipv6:
        2001:db8:b33f::1:
          prefix_length: 128
```

easy to write

easy to read

a failed test won't stop the rest

scales fine with complexity

static rules are good for low changing environments

Comparison (dynamic rules)

dynamic rules

```
---
- get_interfaces_ip:
{% for interface in interfaces %}
{{ interface.name }}:
      ipv6:
        {{ interface.ip_address.split("/")[0] }}:
          prefix_length: {{ interface.ip_address.split("/")[1] }}
{% endfor %}
```

easy to write

easy-ish to read

a failed test won't stop the rest

scales fine-ish with complexity (scales with configuration template)

dynamic rules are good for high changing environments

Summary

Automation is here to stay

If you are not validating changes you are flying blind

Immediate feedback is important, monitoring can be slow detecting issues

Validating changes by hand defeats the purpose of automation

`napalm_validate` allows you to write comprehensive tests for your network

Questions?

@dbarrosop

https://github.com/napalm-automation/napalm
https://napalm.readthedocs.io/en/latest/

Source for the slides and examples:

https://github.com/dravetech/network-validation-napalm