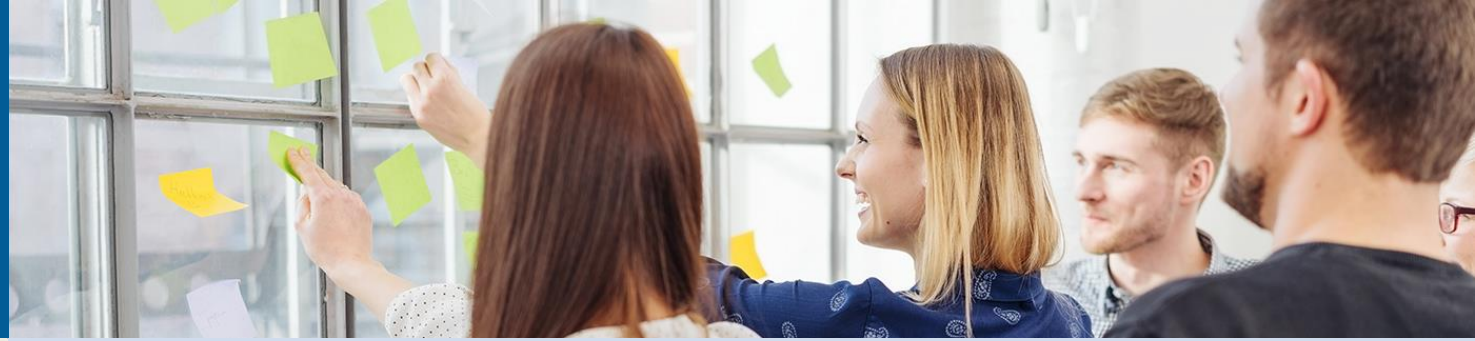
CSC

ICT Solutions for Brilliant Minds



Funet2020 IP/MPLS network automation

nog.fi workshop, Tampere 2019-05-17

Antti Ristimäki, CSC/Funet

Background

- Funet backbone network renewal ongoing on all layers
  - Fiber plant
  - Optical transport layer
  - IP/MPLS network
- Old IP/MPLS network very sparse, core routers only in 7 PoPs
- In the new network, IP/MPLS network will be the primary service layer
  - PE routers in all PoPs
  - Dedicated wavelengths only for heavy users, most services on top of packet network
- As the number of routers increases, old way of managing them just doesn't scale any more

Configuration management in the old network

- Mainly configured by hand via CLI
- Specific tooling for different tasks
  - Peering filters (as-path, prefix-lists) update and configuration
  - MPLS VPN services provisioning
  - Common configuration (loopback filters, prefix-lists etc.) centralized management
- Existing tools include self-made scripts (Perl, Expect, Bash..) and Ansible playbooks. Most tools only serve some specific purpose
- Daily configuration validation by a self-written script
  - Admins receive a daily error report by mail
  - Configuration clean-up and rewrite a manual job

Configuration management in the old network (cont.)

- JunOS apply-groups used to apply common configuration inheritance to relevant elements
  - e.g. customer facing interfaces specific configurations
  - BGP attributes for easily steering traffic during maintenance
  - etc.
- Ironically, router configurations have been used as a network "meta-data" or database and not vice versa

Funet2020 automation
Some goals

- Simple provisioning of new routers – in principle anyone in the network team should be able to do the job
- Consistent configurations across the network
- Standardized services
- Easy provisioning of new customer connections and services – no need to be a CLI jockey
- Less is more: no unnecessary configuration in routers
- Support for multi-vendor environment with reasonable effort

Partial vs. full automation

- Initial idea was to automate ”most” configuration and do the rest by hand
- However, having partially automated and partially manually maintained configuration is awkward
  - possible conflicts between automatically and manually generated configs
  - how to remove elements from the configuration, if the whole configuration (or at least a given configuration hierarchy) is not replaced?
  - if manual config was accepted, the configurations would deteriorate by time

→ Full automation

- We decided to always re-generate the **entire** configuration and then overwrite the entire running config for each router
  - as a result, the configuration includes only the elements we want or need
  - no need for separate garbage collection
  - manual hacks will simply get destroyed
- This is possible as we are building the new network from the scratch – no configuration is copied from the old network

Tool: Ansible and Jinja2 templates

- We have used Ansible for server automation for some years, thus familiar tool
- JunOS has good Ansible support, e.g. in form of existing Ansible modules
- For us, no other realistic choices at this point so we have chosen Ansible also for router automation
- Router configuration generated from Jinja2 template
- However, we use Ansible for IP/MPLS network primarily only as a template engine
  - The template-generated config could be loaded to routers also with some other tools, if needed
  - For now the config is also loaded to routers with Ansible, as it provides nice routines and error handling for that → no need to re-invent the wheel

Data model

- Own data-model, formed by (a lot of) iteration
- Most variables have a default values in template and can be overridden in Ansible variables, when needed
  - e.g. interface MTUs
- Use of e.g. OpenConfig data model would be cool, but in practice:
  - no time to learn it
  - it still supports only a very limited subset of features, so own models would be needed anyway

```
interfaces:

 - name: xe-0/0/0:3
    description: funet2020_testlab-a
    units:
      - number: 1
        description: funet2020_testlab_internet-a
        ip_mtu: 9170
        ipv4_addresses:
          - address: 193.167.244.98/31
        ipv6_addresses:
          - address: 2001:708:0:f001:0:fe08::2/127
      - number: 100
        vrf: funet-mgmt
        description: funet2020_testlab_mgmt-a
        ipv4_addresses:
          - address: 192.168.255.0/31
```

```
routing_instances:
 - name: FUNET-MGMT
   import_communities:
      - name: MANAGEMENT
        accept_prefixes: [ FUNET-MANAGEMENT-NETWORKS ]
   bgp_groups_v4:
      - name: FUNET2020-TESTLAB
        peer_as: 65032
        role: primary
        accept_prefixes: [ TESTLAB1-SW1 ]
        export_prefixes: [ FUNET-MANAGEMENT-NETWORKS ]
        neighbors:
          - address: 192.168.255.1
            description: testlab1.ip.funet.fi
```

More data model examples

```
bgp_groups_v4:

 - name: FUNET2020-TESTLAB
    peer_as: 65032
    role: primary
    export: [ internet-out ]
    accept_prefixes: [ FUNET2020-TESTLAB ]
    bfd: yes
    neighbors: [ { address: 193.167.244.99, description: testlab1.ip.funet.fi } ]
```

```
policers:

  - name: POLICER-2M
    bandwidth_limit: 2m
    burst_size: 256k
    action: [ discard ]
```

```
- name: FW6-BORDER-IN
   terms:
     - name: DISCARD-BOGON-SOURCEADDR
       from:
         source-prefix-list:
           - IPV6-BOGONS
       then:
         - action: count
           param: bogon-source
         - action: discard
     - name: DISCARD-MCAST-SOURCEADDR
       from:
         source-address:
           - ff00::/8
       then:
         - action: count
           param: mcast-source
         - action: discard
     - name: DISCARD-SPOOFED-SOURCE
       from:
         source-prefix-list:
           - IPV6-FUNET-PA-AGGREGATES
           - IPV6-FUNET-PI-PREFIXES
       then:
         - action: count
           param: spoofed-source
         - action: discard
     - name: ACCEPT-BY-DEFAULT
       then:
         - action: accept
```

Minimal ~~Zero~~ Touch Provisioning

```
[funet2020-core-routers]
espoo1.ip.funet.fi ansible_host=espoo1_re0_fxp0
```

- Each PoP is equipped with a serial console server anyway for OOB access, so we use them also for initial commissioning
- Only a few configuration commands to make a newly installed router reachable to Ansible, and then the playbook does the rest
- During the initial commissioning the Ansible playbook is run via "backdoor"
  - SSH is tunneled through serial console server to router mgmt interface
  - In Ansible inventory an alternative host is defined so that it knows to use the OOB access instead of trying in-band SSH
- Remote hands only needed for physical installation

"Database"

- The entire network configuration is now in YAML files
- Maybe one day the data will be pulled from some real database (e.g. Service Now)
- Router specific configuration is defined in the given router's host variables
- Common elements defined in shared vars files
  - prefix-lists
  - firewall filters and policers
  - BGP communities and route-targets
  - Customer AS numbers
- A router configuration is composed from those different data sources

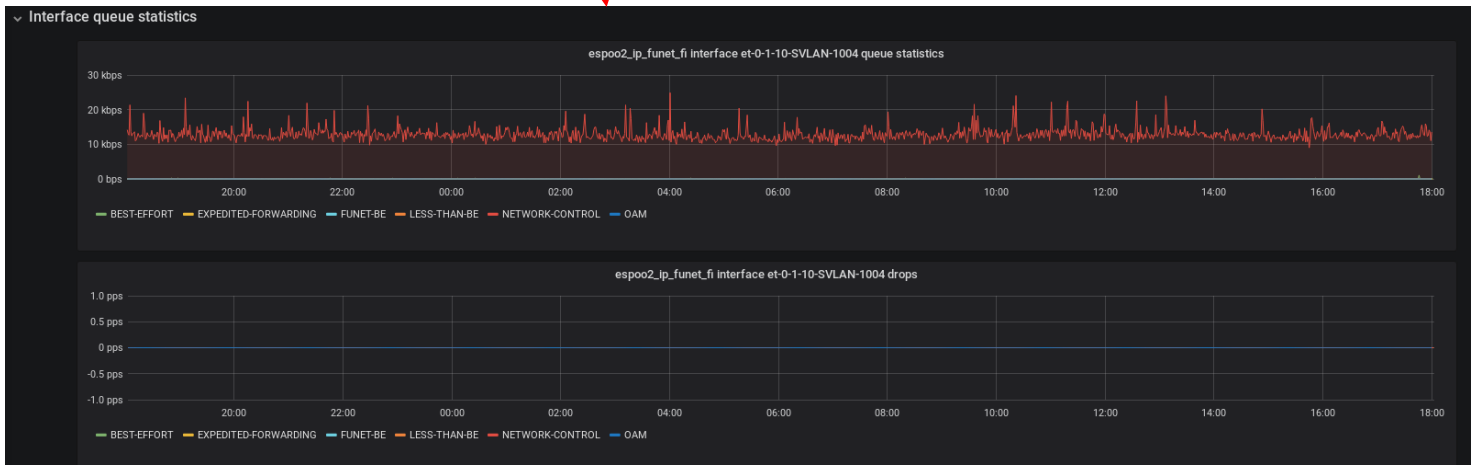
Automation as an enabler

- As configuration is fully automated, configuration complexity is no more a relevant factor when considering which technologies, topologies etc. to use
  - e.g. iBGP full-mesh vs. route-reflectors
  - MPLS LSP provisioning
- Automation makes it easier to use the routers to their full potential
  - Without automation, a lot of things would be too complex or configuration intensive to be deployed for us
  - QoS configuration is a good example – more on that later
- Easy to template another tools using the same existing meta-data, for example:
  - Nagios configuration automatically generated always when routers are configured → always up-to-date with the production network
  - Interface statistics view – links to respective Grafana dashboards
  - DoS filter view for our CERT team

Funet interface descriptions

testlab

Description	Router	Interface	Type	Interface-set	Comment
funet2020_testlab-a	espoo1	xe-0/0/0:3	customer		
funet2020_testlab_internet-a	espoo1	xe-0/0/0:3.1	customer		
funet2020_testlab_mgmt-a	espoo1	xe-0/0/0:3.100	customer		
funet2020_testlab-b	espoo2-sw2	xe-0/0/4	customer		
funet2020_testlab_internet-b	espoo2	et-0/1/10.4	customer	et-0/1/10-SVLAN-1004	



Aggregation switches

- 1/10G connections aggregated and QinQ-tunneled via L2 switches
- Ansible automation is extended also to aggregation switches
  - In most cases enough to only define the customer facing port
  - SVLAN defined and configured automagically (outer VID calculated by switch port number)
- QoS is configured at router using the aggregation switch variables within the router template
  - Outer VLAN is shaped to the switch port speed at router interface
    → the switch doesn't need to buffer
  - Works magically also for LAG ports, e.g. 2 x 10GE switch port is shaped to 20 Gbit/s at router
- All this would be too configuration intensive and error-prone to do by hand

```
interfaces:

  - name: ge-0/0/0
    description: foo
```

For most cases, this is enough to configure the switch port and relevant Q-in-Q tunneling

Customer migrations to new network

- Customer services need to be defined in YAML when migrating to new network
- Mechanical part of converting existing configurations to YAML definitions is the easy part
  - a helper script to reduce manual work, nice especially for prefix-lists and firewall filters
- Existing service standardization/normalization requires a lot of effort
  - different routing policies etc. especially within VPN instances → normalization needed
  - some things might be difficult to generalize
- But, the result really is worth the effort!
  - Beautiful, consistent configurations without any junk
  - No more case-by-case or tailor-made solutions

Custom configurations and exceptions

- In principle, we now try to avoid any custom or tailored configurations
  - Standardized services are much more manageable
- However, in an NREN environment there might emerge needs for some special solutions
- It is also very likely that a service needs to be deployed before it has been incorporated into the Ansible template
- To overcome this limitation, custom configuration can be added as a normal JunOS configuration snippet that will be automatically read into the template
- Still, even if custom needs arise, one should do his/her best to find a generalized way of configuring it and try to incorporate it into the automation template
  - Custom snippet is meant only for a temporary solution

Things to consider

- Learning curve is initially steep
  - wrt using Ansible and running playbooks
  - wrt defining configurations in the new YAML data model
  - wrt fixing things in the playbook and/or Jinja2 template
- Ansible is only a tool, the admin still MUST know what he/she is doing
- One cannot outsource the responsibility to the Ansible playbook
  - Recommended to run in "check mode" first and validate the diff
- Lack of software developer kind of skills might become a bottleneck
- YAML data model documentation – currently worked around by an example file
- Version control conflicts – e.g. change committed to routers not committed in Git repository

Antti Ristimäki

antti.ristimaki@csc.fi

facebook.com/CSCfi

twitter.com/CSCfi

youtube.com/CSCfi

linkedin.com/company/csc---it-center-for-science

github.com/CSCfi