Evolving CERN's Network Configuration Management System

Stefan Nicolae Stancu[1,*], *Arkadiy* Shevrikuko[1], and *David* Gutierrez Rueda[1]

[1]CERN, Geneva Switzerland

Abstract. CERN's networks comprise several thousands network devices from multiple vendors and from different generations, fulfilling various purposes (campus network, data centre network, and dedicated networks for the LHC accelerator and experiments control). To ensure the reliability of the networks, the IT Communication Systems group has developed an in-house, Perl-based software called "cfmgr", capable of deriving and enforcing the appropriate configuration on all these network devices, based on information from a central network database. Due to the decrease in popularity of the technologies it relies upon, maintaining and expanding the current network configuration management system has become increasingly challenging. Hence, we have evaluated the functionality of various open-source network configuration tools, in view of leveraging them for evolving the cfmgr platform. We will present the result of this evaluation, as well as the plan for evolving CERN's network configuration management system by decoupling the configuration generation (CERN specific) from the configuration enforcement (generic problem, partially addressed by vendor or community Python based libraries).

1 Introduction

The CERN IT Communication Systems group is in charge of providing various wired and wireless based communication services across the laboratory. Among them, the group designs, installs and manages a large complex of networks (as illustrated in figure 1). Overall, these networks comprise approximately 400 routers and 4000 switches from multiple vendors and from different generations, with heterogeneous configurations depending on the network area they deserve. To ensure a consistent reproducible configuration across all these devices, an in-house software (denoted as "cfmgr", Perl based) has been developed and augmented over the past 20 years.

After detailing the architecture of the current network configuration management system, its features and limitations, we will describe the generic architecture of a network automation solution, and present our findings of how various open-source libraries and frameworks fit in it. Ultimately we will present our plan for evolving towards a more modular network configuration management system and make use of existing open-source tools.

*e-mail: stefan.stancu@cern.ch

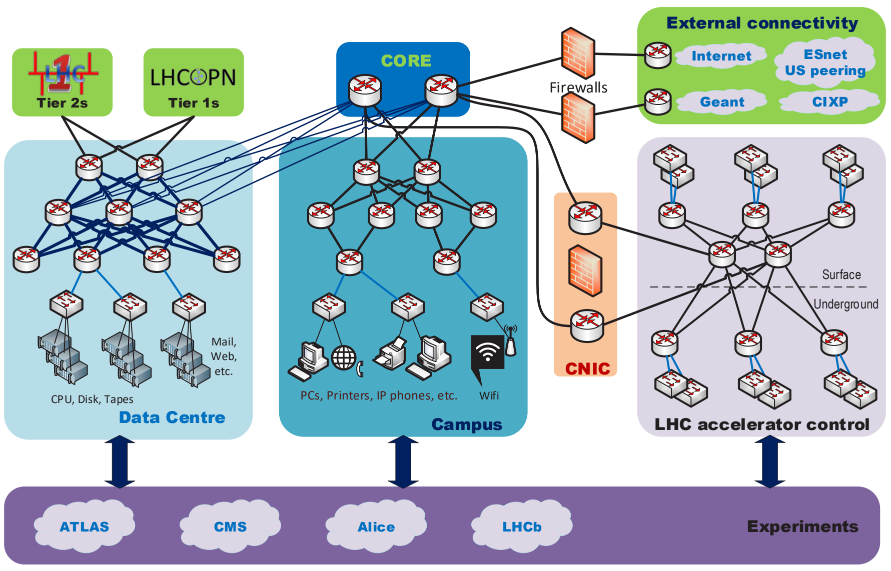

Figure 1: CERN networks complex.

2 Current network configuration solution

The architecture of the current Perl based network configuration management system (denoted as "cfmgr") is depicted in figure 2. Based on a central network database (LANDB), which stores the network model information for all connected devices at CERN, cfmgr is able to derive and enforce the desired configuration on all our network devices, by applying the following workflow:

Configuration generation : the full configuration of every device (with its particularities) is derived, using directly the device specific syntax.

Configuration comparison : the text based configuration data is structured, and the structured generated data is compared with the structured existing configuration. The result of the comparison is a sequence of commands that should bring the configuration on the device inline with the generated one (i.e. a "configuration patch").

Device driver : an abstract interface is implemented for each type of device, typically using the telnet or ssh command line-interface. This allows performing various operations on the device (apply configuration commands, read configuration data or operational state). The device driver is used to apply the "configuration patch" on the network device.

Cfmgr is an essential tool for managing CERN's networks, as it ensures a consistent, reproducible configuration. However, due to its underlying technologies (Perl, with various coding styles that have evolved over a 20 years period), we need to develop in-house the entire workflow and cannot capitalize on emerging open-source tools, which are mostly written in Python. Furthermore, cfmgr was designed for devices with text-based configuration, while modern network devices support structured configuration APIs (e.g. NETCONF [1] and YANG models [2]), and these new technologies have better libraries support in Python (e.g. the ncclient library [3] is a significantly more active github project than netconf-perl [4]).

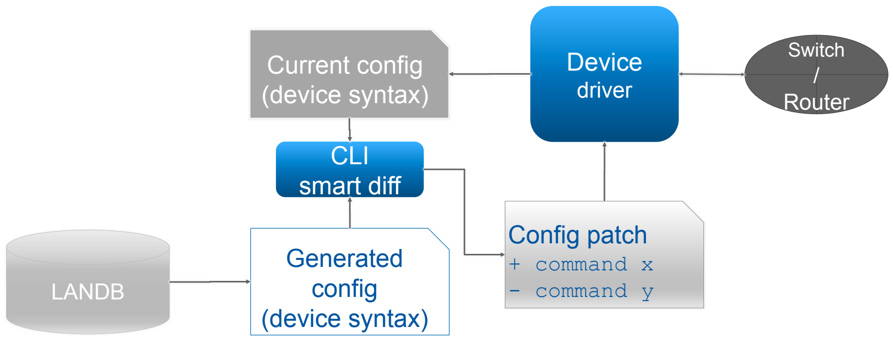

Figure 2: Current network configuration management architecture.

3 Network automation ecosystem

The demand for increased agility in network provisioning has led to an active network configuration ecosystem with contributions from both network vendors and the open-source community. A network automation stack comprises multiple layers, as depicted in figure 3.

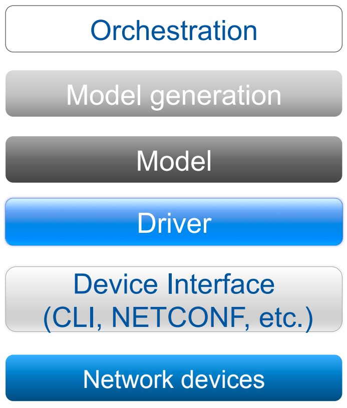

Figure 3. Network automation system stack.

Orchestration From the generic frameworks that deal with IT automation, the following tools have support for network automation orchestration:

- Ansible [5] is a centralized orchestration platform that can sequentially configure parameters on network devices and possibly run conformance reports.
- SaltStack [6] can be used to enforce a desired configuration state on network devices. It is more difficult to use than Ansible, but has more advanced features (state enforcement, database with parameters collected from devices, good scaling capabilities for parallel task execution).
- StackStorm [7] is a generic IFTTT (if-this-then-that) platform, suitable for triggering actions (e.g. maintenance or corrective measures) on network devices when some conditions are met.

Configuration Model Although there are multiple vendor neutral configuration models (e.g. IEEE, IETF and OpenConfig [8]), some configuration parameters are only covered by vendor specific models. The generation of the network configuration model is specific to each environment. At CERN we rely on the LANDB central database for deriving the network configuration parameters.

Driver and device interface The configuration model is translated to the appropriate device specific format and the resulting configuration is enforced on the network device. NAPALM[1] [9] is the leading open-source framework that provides a unified API for interacting with different network device operating systems. It has been evaluated and deemed appropriate for use, as it contains required core functionality (e.g. saving and enforcing configuration), and provides means for extending the device drivers [10].

4 Evolved network configuration system

While the benefits of migrating "cfmgr" to more modern technologies (Python and open-source libraries) are clear, the size of the current project (approximately 200,000 lines of code) and the requirement to keep operating the existing infrastructure make this task challenging.

As illustrated in figure 4, the evolved network configuration management platform aims at decoupling the configuration generation from the configuration enforcement, enabling thus changing and developing each of these components independently. We plan a two-phase evolution:

- In the first phase, rely on the existing Perl code to generate a vendor neutral configuration, implement the translation to a vendor specific configuration in Python, and rely on NAPALM for enforcing the generated configuration on network devices.
- In the second phase, gradually port the generation of the vendor neutral configuration from Perl to Python, making sure that all corner cases are covered and the new implementation generates exactly the same configuration as the old one.

Currently, cfmgr is a standalone Linux program with support for multi-processing. Once its evolution to a Python version with the same functionality will have been completed, we will consider using one of the orchestration tools described in Sect. 3 for implementing more advanced features (e.g. event based reconfigurations).

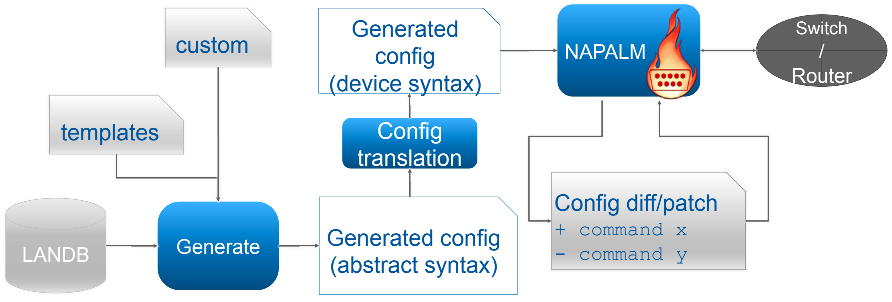

Figure 4: Evolved network configuration architecture.

[1]Network Automation and Programmability Abstraction Layer with Multivendor support

5 Conclusion

Over the past 20 years CERN has developed a tool (called "cfmgr") that ensures a consistent reproducible configuration over several thousand network devices. The recent evolution of network automation tools and the need to support new generation devices, have determined us to look at options for evolving the tool. After assessing the capabilities of open-source network-automation platforms, the benefit of evolving "cfmgr" to make use of them (e.g. NAPALM) and of modern Python libraries became clear.

The new architecture of the network configuration management systems foresees separate modules for dealing with the following three aspects:

- the generation of an abstract network device configuration;
- the translation of the abstract model to a device specific configuration, and
- enforcing the configuration on the network devices.

The decoupling of the configuration generation from the configuration enforcement is key for being able to gradually evolve the cfmgr network configuration system whilst keeping it functional and backwards compatible.

References

[1] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, *Network Configuration Protocol (NETCONF)*, RFC 6241

[2] M. Bjorklund, *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, RFC 6020

[3] *ncclient: Python library for NETCONF clients*, `https://github.com/ncclient/ncclient`

[4] *NETCONF Perl client*, `https://github.com/Juniper/netconf-perl`

[5] *Ansible is Simple IT Automation*, `https://www.ansible.com/`

[6] *SaltStack – Intelligent IT automation and orchestration for your entire team*, `https://www.saltstack.com`

[7] *StackStorm – Robust Automation Engine*, `https://stackstorm.com/`

[8] *Openconfig - Vendor-neutral, model-driven network management designed by users*, `http://www.openconfig.net/`

[9] *NAPALM - Network Automation for the people, by the people*, `https://napalm-automation.net`

[10] *NAPALM - Extending a Driver*, `https://napalm.readthedocs.io/en/latest/tutorials/extend_driver.html#`