Infrastructure as Code at CBC/Radio-Canada's Media-over-IP Data Center

Sunday Nyamweno (CBC/Radio-Canada), Patrick Morin (CBC/Radio-Canada), Carl Buchmann (Arista), Alexandre Dugas (CBC/Radio-Canada), Felix Poulin (CBC/Radio-Canada)

Abstract. *Network infrastructure provisioning is a critical subset of the Automated Deployment of CBC/Radio-Canada's Media-Over-IP Data Center. As the On-Air date approached, it became clear that in order to respond to the fast-changing production needs, traditional network provisioning methods were inadequate. As we began to explore various automation solutions, we realized that a new NetDevOps culture needed to be cultivated within our organization.*

This paper will present the architecture and implementation of CBC's network automated deployment workflow in collaboration with Arista. We believe these tools and methods will be applicable as a way forward to many Media-over-IP projects at all scales.

Keywords. Media-over-IP, automation, configuration, NetDevOps, SMPTE ST 2110, network

1. Introduction

With our new green field production center in Montréal, we selected new Media-over-IP technology to provide the business agility and the density we needed to adapt to the fast-changing media business. And since we now must rely on Information Technology (IT) for the core live media systems, where before we were mastering audio-visual specialty infrastructure, this imposed a major shift on architecture, design, support, and expertise. As we explained in a previous paper[1], we adopted methodologies from the IT data center world to achieve the large-scale deployment of the overall infrastructure. In this paper, we get into more details on the approach we put in place for the deployment of our real-time media networks and our collaboration with network equipment provider Arista to make it reusable for other deployments.

1.1 Challenges of CBC's media networks deployment

Large scale. The new *Maison de Radio-Canada* production center comprises multiple real-time media networks made of around 500 switches connecting more than 6000 media endpoint devices. As we write this paper, this is one of the largest systems of its kind using the new Media-over-IP technology powered by SMPTE ST 2110 and AES67 standards. Those real-time networks are managed by fewer than 10 people.

High availability. We produce and broadcast real-time content 20 hours a day, 365 days a year with our continuous news channel (*Réseau de l'information*), linear radio and TV channels and online streaming. This gives us very limited maintenance windows - usually in the middle of the night - where we can afford risking network disturbance and then make sure that any modifications have not introduced any failure for when the content producers start their day in the early morning.

Complex. This is a complex system as we deal with media specific traffic patterns. We talk about thousands of high bandwidth (e.g., 1.1 Gbps for HD, 10.1 Gbps for UHD) multicast flows at very low latency with flawless real-time delivery. We use Layer 3 routing for scale and robustness purposes. Virtual Local Area Network

[1] A. Dugas, S. Nyamweno, F. Poulin and F. Legrand, "Automated Deployment of CBC/Radio-Canada's Media-Over-IP Data Center," in SMPTE Motion Imaging Journal, vol. 130, no. 6, pp. 20-29, July 2021, doi: 10.5594/JMI.2021.3083557.

(VLAN) and Layer 3 overlays are used for separation into different domains of a common physical layer. We use dual separate networks for redundancy. We are experimenting with Software-Defined Networking (SDN), access control using Access Lists (ACL), priority queuing with Quality of Service (QoS). Etc.

Evolutionary. In order to achieve the business agility, we strived with the choice of Media-over-IP, we need to support the ability to update and evolve the system throughout its lifecycle, much more than the previous "set and forget" approach we could use with former hardware-based and Serial Digital Interface (SDI). All the facilities now use the same backbone. One configuration mistake can bring several ports down, impacting studios, control rooms and so on.

Visible and tracked. With such requirements, it is essential to have a real-time view on the state of the network and a precise tracking of all the modifications to the network configuration to 1- minimize the number of human-introduced errors and 2- be able to quickly react when a problem is arising.

1.2 How Infrastructure as code address those challenges

In the beginning

It became clear during the initial deployment that the large-scale, complexity and high availability requirement of our real-time media networks would require a next level of operational maturity, moreover, considering the size of the teams which are responsible for supporting this infrastructure.

Early in the process we relied on the network provider's management tool (Arista CloudVision Portal (CVP)) for configuration management and network deployment changes. While we were able to proceed with initial deployment with this solution, we quickly realized the limitations of this method for our complex and large-scale system. Arista CVP strengths lie in managing device telemetry information and deployment changes, however, when it comes to network configuration the following challenges needed to be overcome:

- adding or moving switches
- updating network interface configuration
- documenting changes

Key Components of IaC

As defined on a popular website: "Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools."[2] IaC methodologies are now considered best practices in managing IT data center resources; we demonstrate in this work how we were able to adapt these IaC principles to the network infrastructure of a media facility.

We discussed earlier[1] how we started to move away from using spreadsheets to define the network infrastructure as we began to grow. Since then, we have been able to move our network infrastructure to a single source of truth (NetBox) and are currently in the process of moving our end-devices to it.

We put in place a continuous deployment pipeline (using Jenkins) that drives the automation generating of the "running configs" (using Red Hat Ansible and Python playbooks) from the information in the source of truth for all the required parameters such as the IP addresses and topology (NetBox).

[2] https://en.wikipedia.org/wiki/Infrastructure_as_code (accessed on 10/20/21)

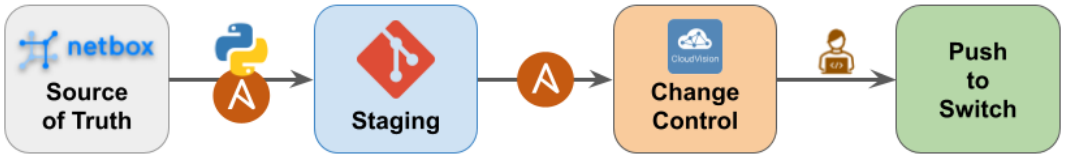


Figure 1: Key components of our IaC workflow

Configurations are managed using source control (Git) and change control is done using Arista CVP. Once a network administrator validates the changes, they are pushed to the network switches. Figure 1 shows the current workflow using those key components. We are actively planning further enhancements to include validation, monitoring and regression testing.

Internally, we have adopted some best practices which are shared by our industry partner, Arista:

- Every pull request includes test case, documentation, and the code
- Stable mainline development strategy: Must pass CI & code review to merge in the *devel* branch.
- Strong focus on documentation.

These best practices have allowed for easy adoption of the cultural shift that we are currently undergoing.

Benefits of IaC

A key concept of DevOps used in software development over the last 20 years is the idea of continuously testing and verifying code changes at multiple stages of the deployment process[3]. The IaC workflow depicted in Figure 1 allows us to define a Network DevOps (NetDevOps) tool chain that can protect our critical infrastructure from making changes directly in the production environment.

Rather than relying on manual configuration of the network switches via the Command Line Interface (CLI) or Graphical User Interface (GUI), both deployment and change workflows are now done using templates as will be discussed later to standardize the configuration across all real-time media networks. This allows us to deal with the large number of switches in a fraction of the time in a robust manner with reduced risk of human error.

All the configuration changes are stored in a Git[4] repository to precisely keep track of who made what changes and when. This way, we can quickly rollback the changes - or some of them - to a previous working configuration, if whenever they cause harm, we did not expect from prior lab tests.

2. The CBC Media-over-IP Network

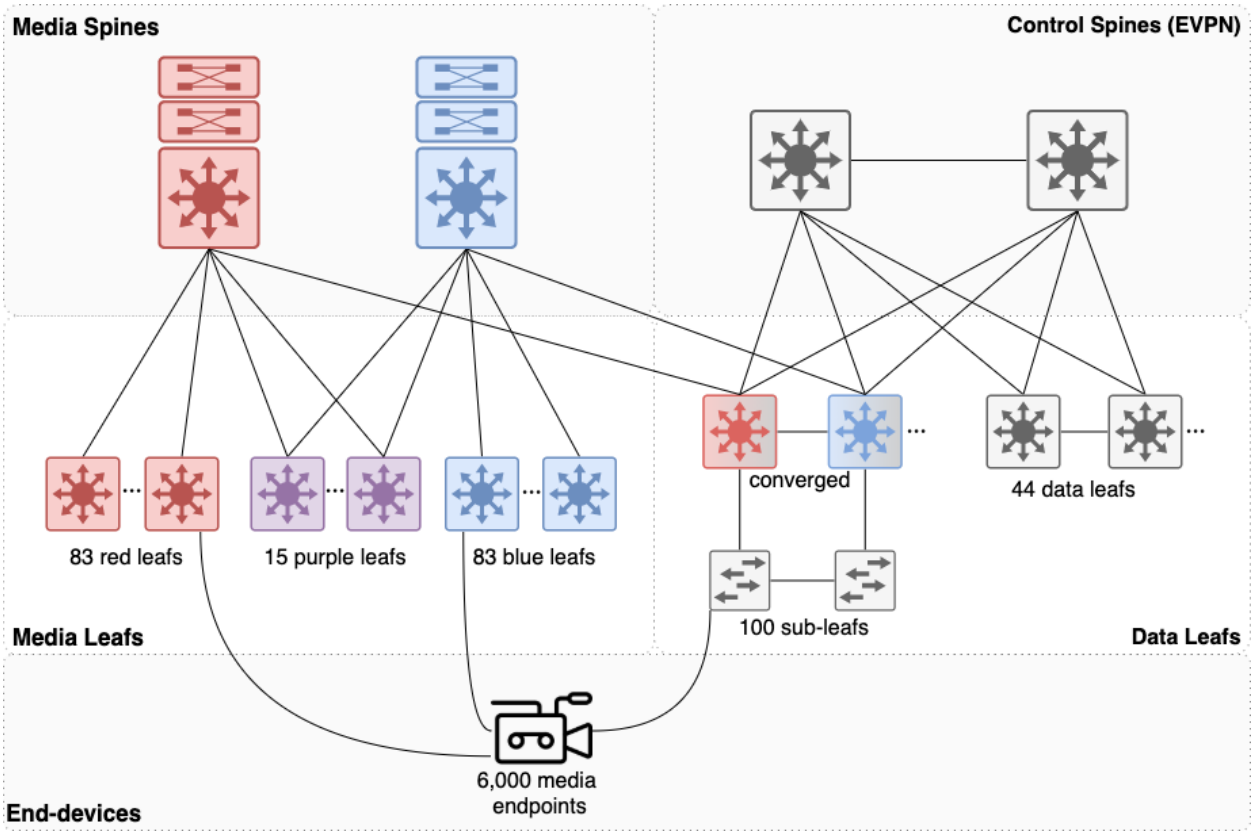
Each independent Media-over-IP network is composed of three networks, for example

1. GREY Control Network – 2 spines, 44 leafs, 100 sub-leafs; Ethernet VPN (EVPN) network
2. RED Media Network – 1 Spine, 83 leaf switches; routed networks
3. BLUE Media Network – 1 Spine, 83 leaf switches; routed networks

The RED and BLUE media networks carry almost exclusively multicast traffic. Each color carries traffic from each side of the SMPTE ST 2022-7 standard for redundancy.

---

[3] https://www.ibm.com/cloud/learn/devops-a-complete-guide (accessed on 10/23/21)
[4] https://en.wikipedia.org/wiki/Git (accessed on 10/20/21)



The network depicted in Figure 2 can support some special features such as

- Purple leaf switches: usually for SMPTE ST-2022-7 devices with a single port like the Riedel Mediornet MuON SDI Gateways
- Moveable devices: devices that can move physical location, but must maintain the same IP address
- Converged switches: capable of both media and control traffic

2.1 BGP and Multicast

On the routing side, the Media networks are regular Exterior Border Gateway Protocol (eBGP) networks where each leaf and spine has its own Autonomous System Number (ASN). Routes are tagged at the source with BGP communities as RED or BLUE and filtered at the spine, because there is some mixing of routes in the “purple” leaf switches. A “purple” leaf is a leaf that has RED and BLUE devices at the same time and that has uplinks to both the RED and BLUE spines as presented in Figure 2.

Converged switches have the full Border Gateway Protocol (BGP) configuration of a media leaf and a modified BGP configuration of a Control leaf. The main modifications are for peering with the Control spines and the Interior BGP (iBGP) peering with the multi-chassis link aggregation group (MLAG) partner switch. The converged switch helped reduce our device footprint while preserving both the media and control functionality in our studios.

2.2 Moveable Devices

Moveable devices can connect in different physical locations while retaining their IP address. For example, a camera may be placed on the east end of the Studio for the morning show and moved to the west end of the

studio floor for the evening show. The studio is equipped with patch panels that allow for device connectivity at different physical locations.

Typically, all network interfaces in a Layer 3 routed leaf-spine media network are configured in a small /30 subnet[5]. This means that the IP address of devices that connect to a specific media interface must change whenever a device moves to a different location. In order to allow for device mobility, while maintaining the same IP address, we implemented the same VLAN subnet with an anycast gateway on multiple switches, but without Layer 2 reachability across these same switches. This has been made functional by announcing a /32 host route for each and every moveable device with an ARP entry on the VLAN.

Unfortunately, there are some quiet devices. We have convinced some vendors to implement gratuitous ARP on their devices and others have configured LLDP to announce the IP of the device in the *mgmtAddress* field or the *sysName* field of the LLDP announces. An event handler is then configured to ping the device when it comes online[6]. If none of those are possible, we are thinking, in a last resort, of implementing a ping sweep locally on each switch with moveable devices.

2.3 Implementing Multicast Determinism and Security

All our Media leaf switches have more than one uplink. Often a single uplink can transport all the theoretical traffic that the leaf can generate or request, sometimes not. We put in place a system that will manage the bandwidth used on the uplinks by statically mapping multicast routes on specific uplinks, making sure not to oversubscribe the links. The system is bandwidth aware and knows how much bandwidth is used by each flow.

We also want to make sure that a rogue end device can't flood the network with more data than it is supposed to (example: sending out a UHD signal instead of a 1080i signal) The system will configure for each authorized source an admission control policy to permit the specific flows and their allowed bandwidth.

Arista Media Control Services (MCS) has been able to meet some of these design objectives. In parallel we have been exploring how multicast determinism can be added to IGMP and this presents an interesting area of further research.

3. An Extensible Automation Framework

Although many network users share common design patterns, such as leaf and spine fabrics, each of them brings unique sets of requirements. Even in the case of CBC, the control network (EVPN) and the media networks share some commonalities, but also have some unique sets of requirements. So, let's explore how this can be accomplished with an extensible automation framework.

When designing the automation of a network fabric, it is important to provide an abstraction layer to configure fabric as a whole and treat it as a distributed system, as opposed to translating what we do when managing devices independently via CLI into a script or playbook. For example, adding a new leaf in a fabric constitutes creating a new configuration for the leaf, as well as updating the configuration of the spine switches, so it is important to consider the fabric as one system. To achieve this, an extensible automation framework provides various abstraction layers to effectively help build, document and test the fabric.

---

[5] https://thebroadcastknowledge.com/2020/12/14/video-proper-network-designs-and-considerations-for-smpte-st-2110/ (accessed on 10/23/21)
[6] https://github.com/philippebureau/EOS-scripts-public/tree/main/Ping%20LLDP%20neighbor (accessed on 10/23/21)

It starts with the network operating system. It needs to provide a robust programmable interface that is simple to use and that supports a config replace strategy using the device CLI syntax. This allows us to create the full configuration of the device, streamlining the process of deterministically applying configuration to the device based on our intended state. This is a very important construct because network engineers are very familiar with this network operating system CLI syntax, and it greatly simplifies the templating required to generate our configuration.

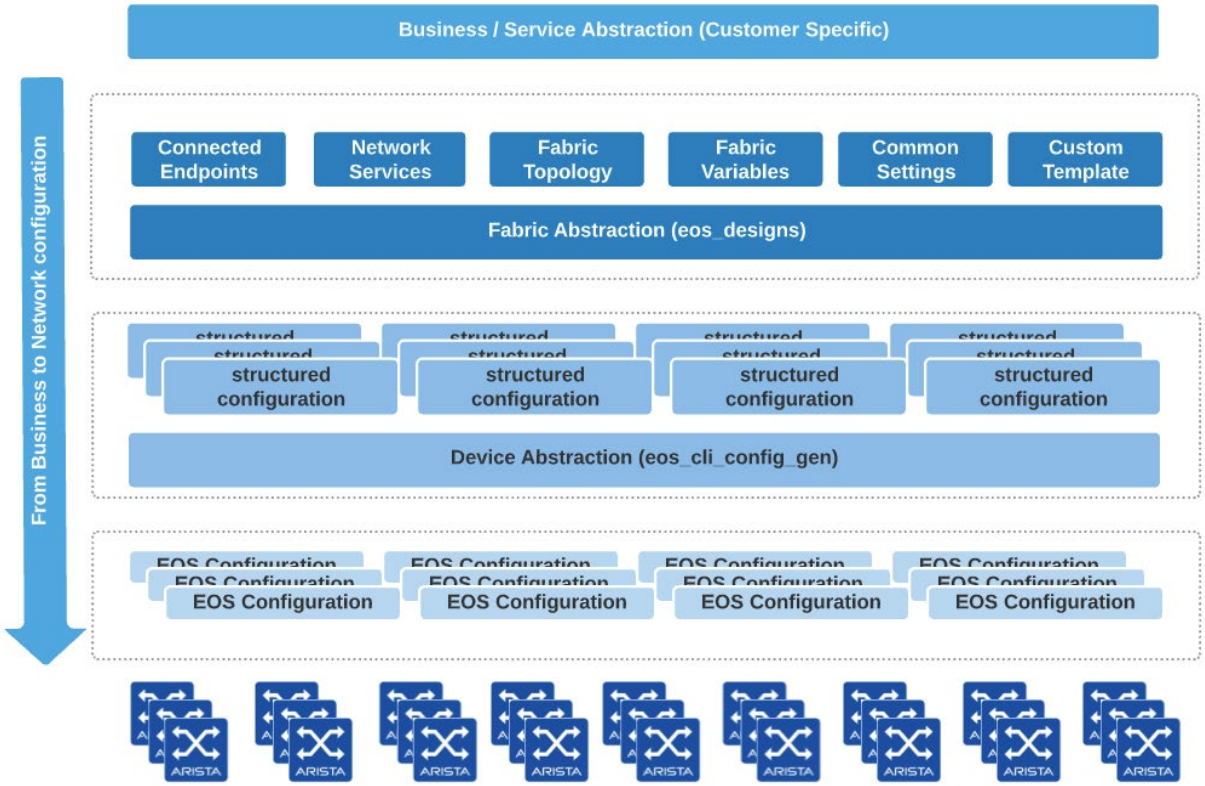


Figure 3: Arista Validated Design Configuration Abstraction Model

Next, the device abstraction layer provides machine-readable configuration referred to as the “structure configuration.” The data model for this layer is simply a YAML or JSON representation of the configuration, which is then leveraged to generate the EOS CLI configuration, the documentation, and auto-derive network ready for use (NRFU) test cases.

Finally, the fabric abstraction layer provides various building blocks to configure an end-to-end fabric: Fabric Topology, Network Services (EVPN), Connected Endpoint, Fabric Variables, and Common settings. The data model for this layer provides an abstraction layer that simplifies the configuration of a network fabric and enforces network design best practices. The focus of the layer is to provide a simplified interface and minimize the user inputs required and abstract. The framework also allows the user to add their own custom template to address specific use cases.

When users interact with the framework, they can either interact with it natively with YAML data models stored in a git repository or develop custom plugins to pull data from one or many external sources, i.e., NetBox. This is referred to as the “business / service abstraction” layer which is always user specific.

3.2 Arista Validated Design - Ansible Collection

CBC is using the Arista Validated Design (AVD) Ansible collection (*arista.avd)* that provides opinionated roles and modules to help users automate the complete lifecycle of managing using Infrastructure as Code an end-to-end Arista fabric.

The Ansible collection follows these guiding principles:

- Design to be easily extended, and customizable to users' needs, by the end user.
- Has a well-established coding style and guidelines speed up onboarding new contributors.
- Focus on community (mostly network engineers) participation and feedback.
- Works with or without Arista CloudVision

The arista.avd collection provides roles that achieve the following:

- Builds Arista Extensible Operating System (EOS) configuration for each device.
- Documents fabric and device configuration in Markdown and format
- Configures deployment via Arista CloudVision Portal (CVP)[7] natively via Arista EOS API (eAPI)
- Validates fabric configuration and post-deployment based on intent.

CSV

or

pre

As of arista.avd 3.0.0, the collection supports the deployment of any type of topology with various stages, partial mesh or fully meshed topologies.

3.2 Arista AVD Provisioning Building Blocks

The Arista Ansible AVD collection provides the following configuration management building blocks:

- Ansible provides an orchestration layer between Source of Truth and Arista CloudVision
    - User input for various source of truth in YAML, NetBox[8], CSV files, etc.
    - Ansible aggregates all data in a Git repository.
    - Template engine (Jinja2) and custom plugins (Python) provides data transformation
    - Configuration is pushed to Arista CloudVision
    - Arista CloudVision provides Change Control and Provisioning Engine
    - When the changes are completed, Arista CloudVision provision provides a rich set of telemetry to monitor the active state of the network.

[7] https://www.arista.com/en/cg-cv/cv-introduction-to-cloudvision
[8] https://netbox.readthedocs.io/en/stable/

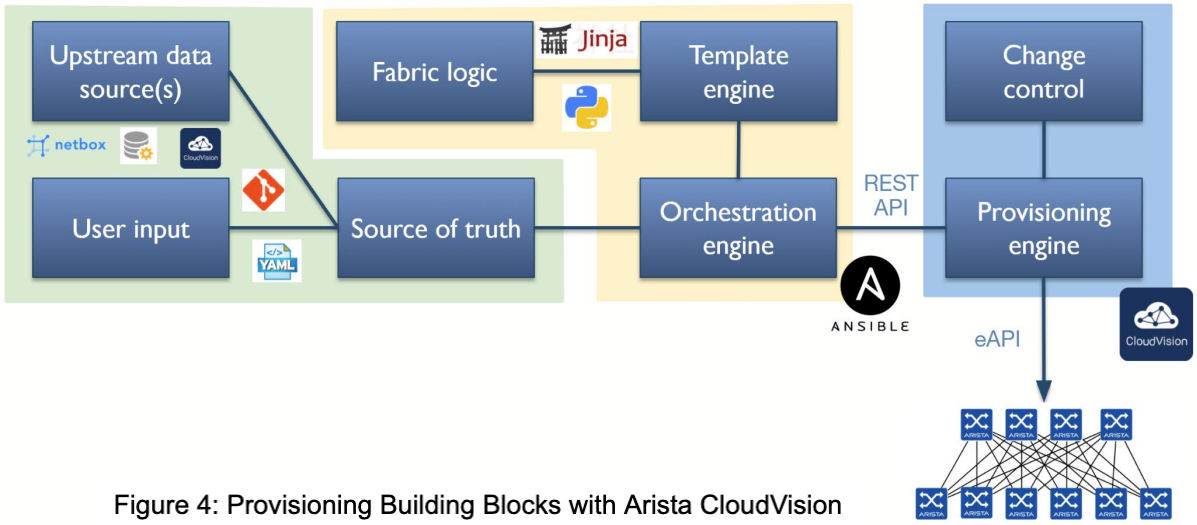


Figure 4: Provisioning Building Blocks with Arista CloudVision

4. Implementing our Media-over-IP network

Adding ***arista.avd*** to our deployment pipeline was motivated by the need to increase visibility and tracking of changes. This change in practice is taking us through a cultural transformation where we are learning new skills which have been established in the IT industry. Our current NetDevOps based workflow obtains switch information from NetBox and end device data from a spreadsheet. There is an ongoing effort to consolidate end-device information into a single source of truth (SSoT).

Ansible and Python are used to create Ansible variables from NetBox and spreadsheets which then rely on ***arista.avd*** to generate switch configurations. All configuration files are stored in version control. In order to meet our requirements, we needed to extend the capabilities of arista.avd for a Real Time Media network.

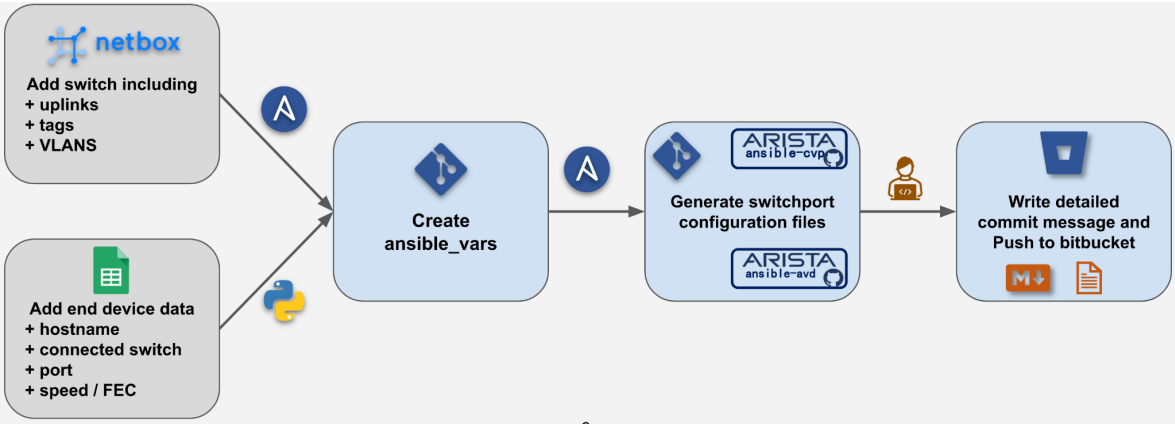


Figure 5: CBC's Media-over-IP deployment pipeline

4.1 Extending *arista.avd*

Ansible-AVD has support for L3 EVPN networks but does not yet support some of the multicast and AMCS functionality required by our network. In order to add the functionality our network requires, we started a GitHub project to collaborate with Arista. In parallel, we created a fork of ansible-avd where we could internally test the tools we need by

1. adding knobs to *eos_cli_config_gen* for required CLI commands
2. Creating a *rt-media* design that matches the description in Section 3

4.1.1 Adding knobs to eos_cli_config_gen

An example of this is to implement the static multicast routing required by AMCS, we needed to enable multicast routing on all ports in the network. Since ansible-avd does not support this feature, we added it in our fork for internal deployment, and created an issue upstream with ansible-avd. Other broadcasters using AMCS will be able to easily deploy deterministic routing with this addition.

4.1.2 Creating rt-media design abstraction

The first implementation of the red and blue network described in Figure 2 relied on classifying our inventory according to color and type. For the media portion of our network, our devices were grouped by: Red Spine, Blue Spine, Red Leaf, Blue Leaf, Purple Leaf.

Special variables related to these classifications were stored using ansible's group_vars structure and variables specific to each host were stored in a host_vars file. Ansible-AVD was then leveraged to generate the switch configurations. Adding ansible-avd to our workflow improved response time to change requests and added flexibility required for special events like the Olympics.

However, another layer of abstraction is required to further increase flexibility and add the converged switch which is the special class of network services described in section 3. We modified the l3ls_evpn eos_design, by removing the features that were not required such as MLAG and adding the BGP and multicast settings described earlier. We also added customized templates for the required BGP and Multicast required by our media network. Converged switch abstraction will be added soon.

5. Summary

By transforming our deployment workflow using NetDevOps models, we have been able to add tracking and flexibility that has resulted in significant process efficiency. For example, change requests happen faster and increased visibility allows network experts to better collaborate on design strategies. Furthermore, major changes like switching from IGMP to AMCS can be done much faster. Using a traditional method to configure the network would require wide maintenance windows and downtime. The NetDevOps model allows preparing the changes offline and push the latter during a short maintenance window.

Acknowledgements

The authors would like to thank the colleagues at CBC who contributed to this project, in particular Selva Subramaniam, Yann Lefebvre, Alexandre Cormier, Guillaume Lorrain-Bélanger, Emanuel Mateus, Alexis Ouellet Patenaude, Alexandre Hébert, Alexandre Pellerin, Eric Waltz, Dave Laroche, David Marangoni, Ryan Fathi, Cyriel Ricout, Alexandre Clemenceau, Jules Dumontier, Mario Dageneais, Marlene Al-Kazzi and all the colleagues who provided requirements and feedback.

Arista lead maintainers for arista.avd and arista.cvp collection: Claus Holbech and Thomas Grimonet.