

Cloud-native Network Controller

Department of Computer Science
OST – University of Applied Sciences
Campus Rapperswil-Jona

Autumn Term 2024

Authors	Vanessa Gyger Patrick Lenherr
Advisor	Urs Baumann
Co-Advisor	Jan Untersander
External Co-Examiner	Philip Schmid
Gegenleserin	Mitra Purandare
Project Partner	INS

Abstract

The philosophy in application operations has changed significantly over the last few years. Most applications no longer run on dedicated servers but have been migrated to virtual machines. For some time, they were used as best practices. Nowadays, applications are already developed in a cloud-native manner for easy deployment in container-based infrastructures. What happened in the meantime in the domain of network configuration management? The focus was set on automation enhancement with popular tools like Ansible and various proprietary solutions. They provide a one-shot way of deployment, where the configuration will be applied but still leave the possibility open to overwrite the changes manually. In this approach, there is no built-in self-healing of configuration drift. A group of network engineers is developing a tool stack around Kubernetes, allowing the user to leverage its cloud-native advantages for network configuration.

The objective of this bachelor thesis is to analyze the main component of the tool stack SDC (Schema Driven Configuration) in the use case of automating an EVPN fabric. The analysis shall examine the operating principle of SDC while also considering its place in the cloud-native ecosystem from data source to the device. Based on the analysis, a verdict should be given on SDC's fulfillment of requirements, multi-vendor compatibility, and readiness for use in production.

The findings prove that SDC follows great principles: It is designed to be vendor agnostic, working with virtually every vendor that supports Netconf or gNMI as a configuration interface and is customizable. This facilitates avoiding vendor lock-in. Being comprised of Kubernetes resources, SDC works completely declaratively and uses a mechanism for continuous reconciliation. The tool is open-source. Therefore, it is possible for anyone to use and improve it.

Since SDC is still in heavy development, the authors also found a few minor bugs - some of which have been fixed during this project. The more significant issue lies in the usage of YANG: SDC uses YANG schemas to validate the configuration before writing it to the devices. It was found that SDC often has trouble loading these schemas either because of bugs in SDCs loading mechanism or due to the fact, that some of the (vendor-supplied) schemas contain many deviations which lead to breaking the configuration tree. Therefore, using SDC in production requires a fair amount of effort to rework the available YANG schemas for the devices.

SDC integrates with Kuberenet as the configuration generation component. It first enables the creation of an abstract configuration, which will then be transformed into the device-specific configuration. Due to some disadvantages of running the configuration creation in Kubernetes, such as the lack of a good way to do a dry run, a new tool called Choreo exists. It replicates the API functionality of Kubernetes while providing additional functionality, such as running to completion with either a successful or failed result. These tools' customizability enables extreme flexibility but also increases their complexity, and many components for their functionality must still be created. This development is expected to be done by the user or the community as they have not (yet) been provided by the developers. As can be seen, SDC still needs some time for development to be entirely usable in productive environments. Nevertheless, it has many exceptional traits, making it inimitable in the current market.

Acknowledgments

The authors would like to thank our project advisors, Urs Baumann and Jan Untersander, for their continued support and guidance, as well as the expert Philip Schmid and “Gegenleserin” Mitra Purandare, for their time and effort in evaluating our work.

Our thanks also go to the developers of the tool SDC, especially Wim Henderickx and Hans Thienpondt, for their help in understanding the quirks and bugs.

Lastly, the authors are grateful for their patient friends, colleagues, family members and supporting employers.

Contents

- 1. Management Summary..... 8
- 2. Assignment..... 9
- 3. Methodology..... 10
- 4. Tool Stack / Technologies 11
 - 4.1 Big Picture 11
 - 4.2 EVPN-VXLAN..... 12
 - 4.3 Kubernetes 14
 - 4.4 YANG Models..... 15
 - 4.5 SDC..... 16
 - 4.6 Configuration Generators..... 18
 - 4.6.1 Kubenet 18
 - 4.6.2 Choreo 19
 - 4.6.3 Infracore 22
 - 4.7 Target Device Vendor 24
 - 4.7.1 Cisco..... 24
 - 4.7.2 Arista 24
 - 4.7.3 Nokia 24
 - 4.7.4 Juniper..... 24
- 5. Requirements..... 25
 - 5.1 Use Cases..... 25
 - 5.1.1 Actors 25
 - 5.1.2 Use Case Description..... 25
 - 5.2 Non-Functional Requirements..... 26
- 6. Challenges 27
 - 6.1 Knowledge Gap..... 28
 - 6.2 Cisco Netconf and YANG Models 30
 - 6.3 Arista and YANG Models 32
 - 6.4 Schema Validation Errors 35
 - 6.5 Broken K8s API in a test cluster..... 37
 - 6.6 Choreo Examples 37
 - 6.7 Debugging SDC 39

6.8	Arista new Release	43
7.	Applicability of the Tool Stack for Network Automation	48
7.1	Operational Lifecycle of an EVPN-VXLAN Spine-Leaf Fabric (L2).....	48
7.1.1	Initial Deployment.....	48
7.1.2	Add/remove L2 Endpoint	48
7.1.3	Add/remove L2 Service	49
7.1.4	Add/remove Leaf.....	49
7.1.5	Add/Remove a Spine	50
7.2	Proposed partitioning of the Configuration into Configuration Intents.....	51
7.3	Dependencies of Configuration Parts.....	53
7.4	Configuration Parts composing the Tasks.....	55
7.4.1	Removing Configuration via gNMI	55
7.5	Ecosystem around SDC (Requirements / Interfaces)	56
7.5.1	Configuration Generation.....	56
7.5.2	Configuration Source.....	57
7.5.3	Configuration Deployment (Kubenet, Choreo, Infrahub).....	58
7.5.4	Rollout Strategies	59
7.5.5	Operational Requirements	60
7.6	Fulfillment of Requirements.....	61
7.7	Verdict on SDCs readiness for production.....	63
8.	Outlook	66
8.1	Contributions.....	66
9.	Conclusion	67
10.	Risk Analysis.....	68
10.1	Overview	68
10.2	Details.....	68
10.3	Risk Development	69
10.3.1	16.10.2024.....	69
10.3.2	21.10.2024.....	69
	29.10.2024.....	70
10.3.3	18.11.2024.....	70
11.	Project Plan	71
11.1	Roles and obligations	71

11.2	Process Model	71
11.3	Time Management.....	71
11.3.1	Long-term planning.....	71
11.3.2	Short-Term Planning	72
11.4	Collaboration	72
11.4.1	Communication.....	72
11.4.2	Time Tracking.....	72
11.4.3	File Storage, Versioning and Review	74
11.5	Meetings	74
12.	Appendix	75
12.1	Glossary	75
12.2	List of Figures.....	78
12.3	Bibliography & References.....	80

1. Management Summary

Baseline

The field of IT infrastructure operations is subject to constant changes. Application operations have changed substantially in the last decade. Dedicated physical servers running applications were soon replaced by virtual machines. Nowadays, most applications are designed to run in containerized infrastructures. The network management side of the infrastructure is undergoing its own evolution. For some time now, network devices are no longer configured by manually issuing commands on the CLI but are managed by a variety of proprietary and open-source management tools. To take network device management one step further in the direction of the cloud-native paradigm, a group of network engineers has developed a tool stack which leverages the advantages of Kubernetes to manage network device configuration. The question is whether these tools fulfill the use case of a simple spine-leaf fabric using EVPN and whether they are already mature enough to be used in production.

Approach

The authors first acquired the necessary knowledge on EVPN by solving the corresponding lab of the bachelor courses CN2 module, which didn't exist when they attended it. As a further acquisition of knowledge, the available documentation resources and tutorials regarding the tool stack were consulted.

It was then decided to focus on SDC first. The authors investigated the functionality of SDC by performing simple configurations for a Cisco device. Due to the lack of sufficient YANG support from Cisco, another vendor, Arista, was chosen, and the same configuration was tested with this vendor. It was found that Arista provides YANG models, which create incompatibilities with SDC.

As a next step, an effort was made to analyze SDC runtime behavior, which was not pursued further due to time constraints. The focus was instead laid on SDCs environment and the analysis of its feasibility for the examined use case. In this respect, the tools Choreo and Infracore were visited and investigated.

Results

The results show that SDC follows good principles, such as designing for vendor neutrality and extensibility. The experience also shows that SDC is still in heavy development and contains several flaws and imperfections. These errors are also not easily debugged. This fact, along with the circumstance that creating suitable configuration input for SDC also needs extensive effort to be made by the user, leads to the verdict that SDC is not yet on the maturity level required for use in production.

Forecast

SDC holds a lot of future potential if first, the compatibility with YANG models from more vendors is improved either by the SDC developers or the vendors, and second, the effort needed to create the configuration input is lowered by making the generation tool Choreo more user-friendly or developing a new tool for the task altogether.

2. Assignment

Introduction

The landscape of IT infrastructure is characterized by constant evolution. Over the last decade, there has been a significant change in the philosophy of application operations. The development went from physical servers over virtualized ones to containerized applications. Today, applications are often developed in a cloud-native manner, meaning that they can use all the benefits a cloud environment provides. The infrastructure is not only comprised of server systems but also uses network devices as vital elements. The way they are configured has not changed as much, but now it is also picking up on the trend.

Status quo

Network automation has been done for quite some time now. There exist not only proprietary solutions sold by network device vendors but also a variety of open-source tools like Ansible. The cloud-native paradigm for applications doesn't just suggest automation but also scaling, an event-driven architecture, and continuous reconciliation (self-healing). The de-facto industry standard for cloud-native containerized applications is the orchestration tool Kubernetes. A group of network engineers has now developed a tool stack, comprised of SDC and Kubenet, around K8s with the aim of leveraging all of Kubernetes' advantages for network configuration.

Goals of the Project

The objective of this bachelor thesis is to analyze the main component of the tool stack SDC (Schema Driven Configuration) in the use case of automating an EVPN fabric. The analysis shall examine the operating principle of SDC while also considering its place in the cloud-native ecosystem from data source to the device. Based on the analysis, a verdict should be given on SDC's fulfillment of requirements, multi-vendor compatibility, and readiness for use in production.

3. Methodology

The authors took the following steps to analyze SD and its environment, and to give a verdict on its readiness for production.

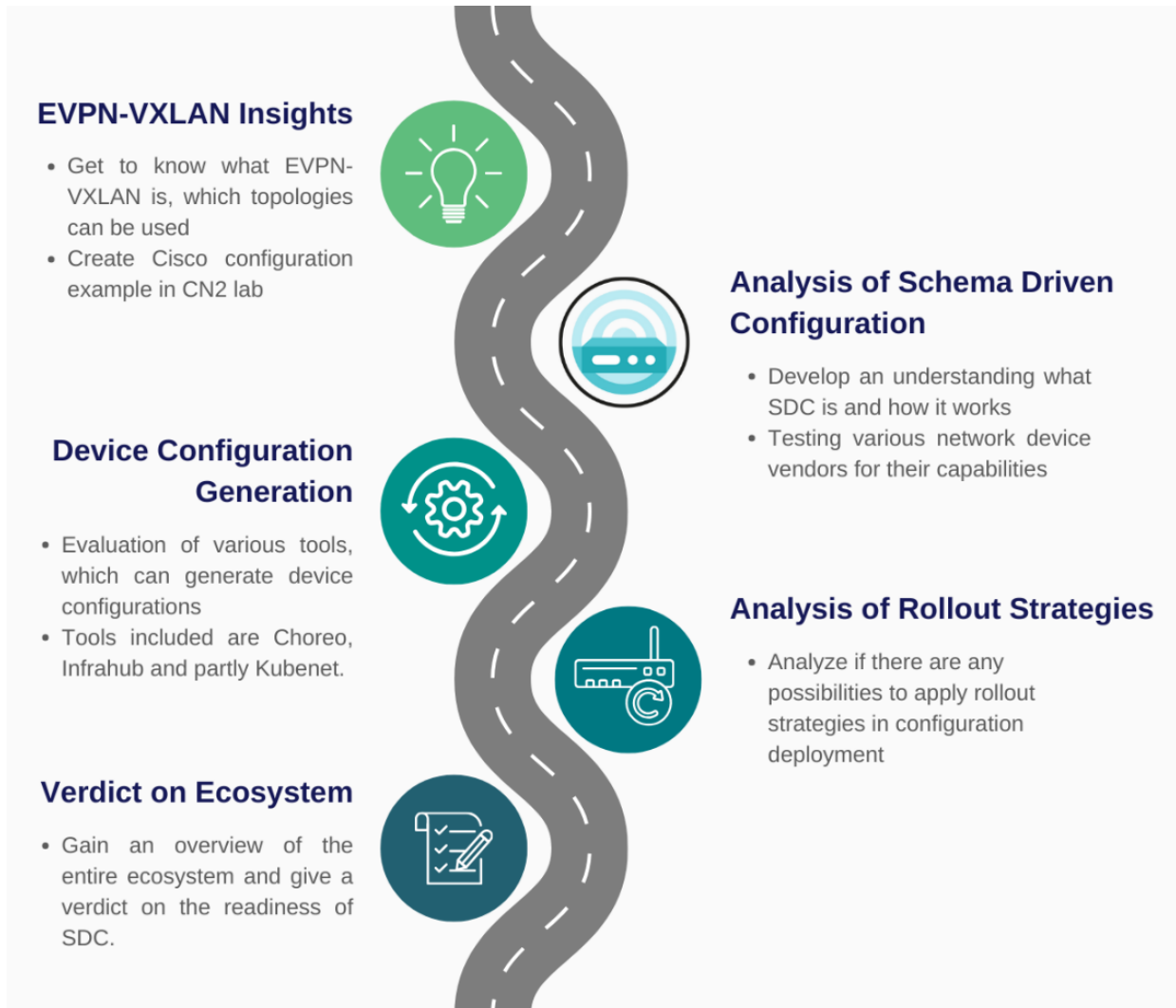


Figure 1 Overview of the project path [1]

4. Tool Stack / Technologies

The main tool, named SDC, was given by the advisors. Consequently, Kubernetes was also a prerequisite as it is needed by SDC. Tools used as possible data sources for SDC emerged during the project from the team's research and advisor's recommendations.

Different target device vendors were used during the project based on their compatibility with SDC.

4.1 Big Picture

To already receive an idea of how they could work together in the ecosystem, the components are presented here. The light blue colored ones generate configurations, the green one is used for strategic deployment, the red one, SDC, is the main part, and in dark blue are the network devices, called targets.

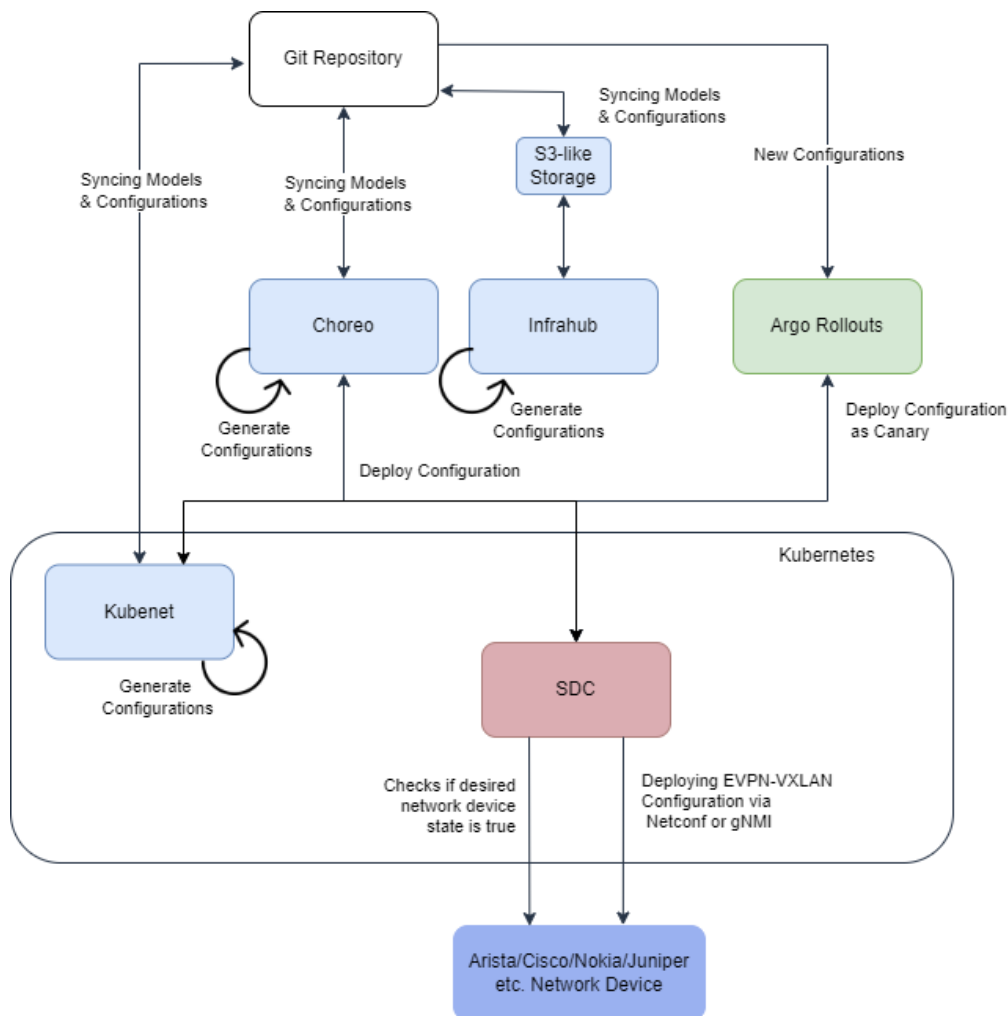


Figure 2 Proposed Ecosystem

For a better understanding, the following illustration shows the workflow and how reconciling should work in this use case. This is the optimal use case for using SDC.

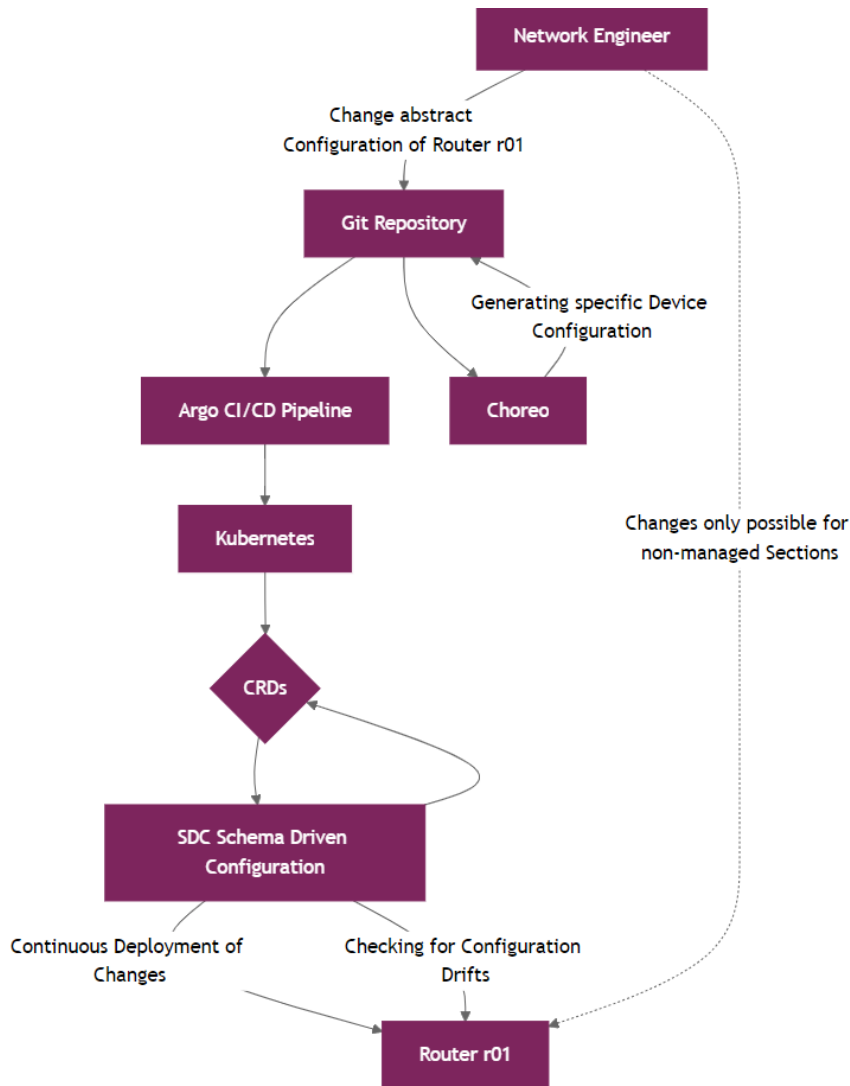


Figure 3 Ecosystem Workflow

4.2 EVPN-VXLAN

EVPN-VXLAN¹, is an overlay control/data plane used in networking. It enables organizations to operate their physical network infrastructure (underlay) in a simple routed configuration, which leads to small L2 (broadcast) domains. The virtual topology (overlay), orchestrated by EVPN, supports both bridged (L2) and routed (L3) segments that connect two or more endpoints. These endpoints are bridged to access ports towards client and server hosts.

VXLAN is an enhancement of VLAN that provides more logical networks than its predecessor by using 24 instead of 12 bits as an identifier, called a Virtual Network Identifier (VNI). VXLAN encapsulates

¹ Spelled out: Ethernet Virtual Private Network-Virtual Extensible LAN

L2 frames into L4 UDP datagrams, which creates a tunnel that is routable over L3 networks. The entities terminating the VXLAN tunnels are called VXLAN tunnel endpoints or VTEPs.

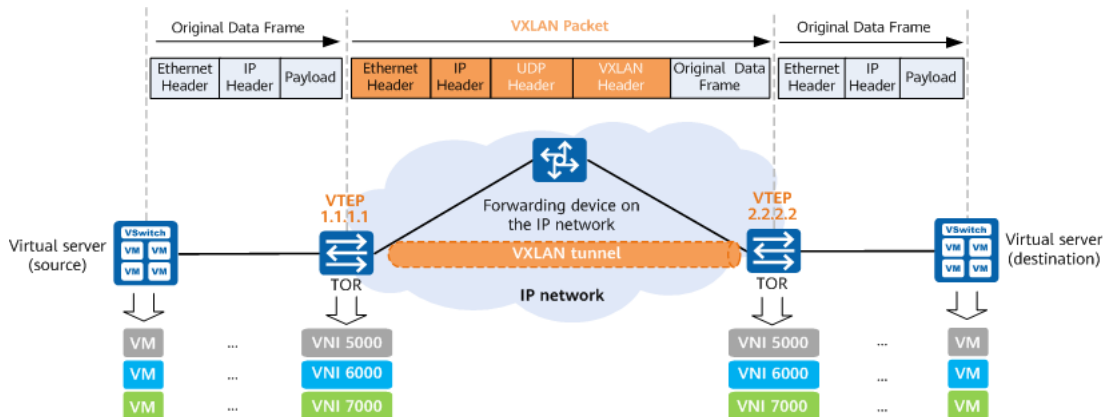


Figure 4 VXLAN tunneling[2]

EVPN itself is an extension of MP-BGP that is used to dynamically announce the MAC or IP of the hosts on the network. This is needed to route the traffic to the correct destination VTEP.

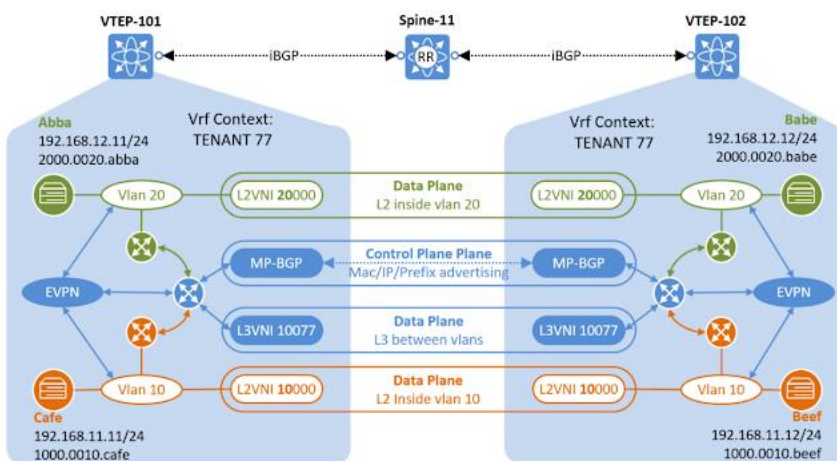


Figure 5 Overview of EVPN with MP-BGP and VNIs[3]

As an underlay topology, the Spine-Leaf architecture is often used. It features switches in a distinct role of either spine or leaf. Leaf switches are used to connect to hosts. There is no host connected to a spine switch. Spine switches are connected to an upstream gateway (Internet) or to a core for further concentration. In the standard configuration, a spine is connected to all leaves but not to other spines. Vice versa, a leaf is connected to all spines but to no other leaf. This topology provides various advantages, such as:

- Scalability: Easy addition of access ports by adding more leaf switches. Easy addition of bandwidth by adding more spine switches.
- Predictable latency between endpoints: The hop count between leaves is always 1.
- Maintainability of VXLAN configuration: New/changed VNIs do not need to be configured on the spines.

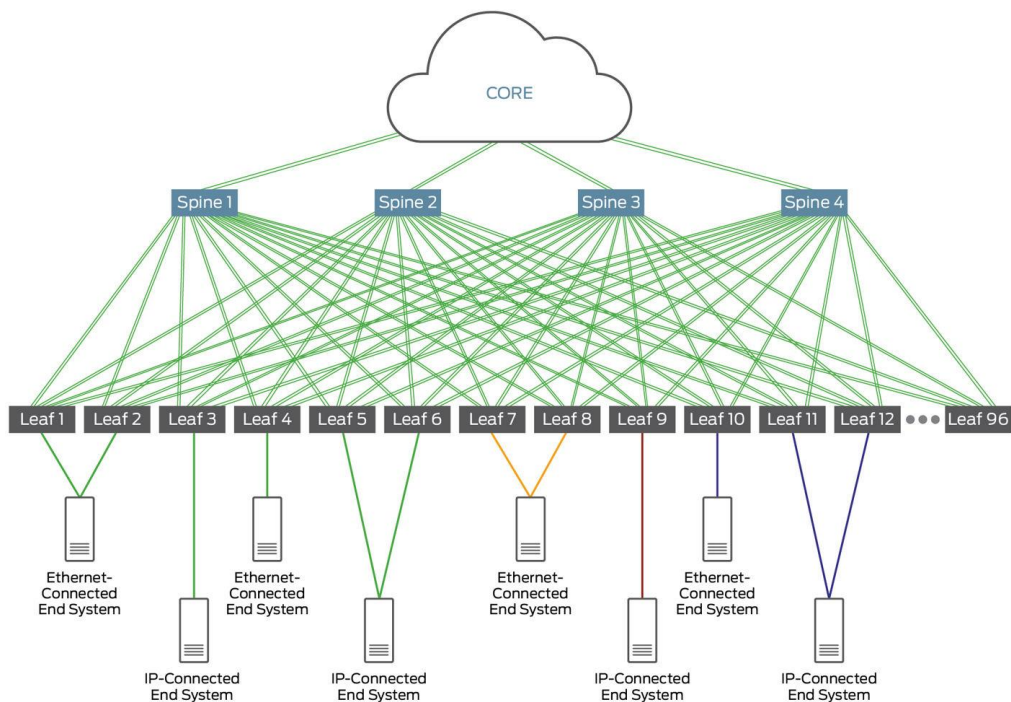


Figure 6 Spine-Leaf topology[4]

4.3 Kubernetes

“Kubernetes, also known as K8s, is an open-source system for managing containerized applications across multiple hosts. It provides basic mechanisms for the deployment, maintenance, and scaling of applications.”[5]

Because Kubernetes is already widely known, this section will just highlight some of its features that are relevant to this project. To readers not familiar with Kubernetes or interested in learning more details, the authors recommend the official documentation[6].

Declarative Model: Resources in Kubernetes are defined using so-called manifests written in YAML. The manifests do not contain actions but rather a declaration of the desired state. Kubernetes then interacts with the target systems by creating or configuring resources or components. This can, for example, be a container running an application or, in our case, push configuration to a network device.

Self-healing: Kubernetes continuously monitors its resources and automatically triggers the responsible controllers depending on events, like a failed container or, in our case, a changed device configuration. The controller then takes the steps needed to bring the actual state back to the desired one. This is called continuous reconciliation.

Extensibility: Kubernetes does not limit the types of resources that can be managed. In fact, an important part of it is the support of the so-called Custom Resource Definitions (CRDs) that allow developers to extend the API of Kubernetes to support more types. As the following chapters will prove, SDC and the accompanying tool Kubenet rely heavily on this extensibility.

4.4 YANG Models

YANG, short for Yet Another Next Generation, is a data modeling language developed specifically to define data concerning network management. It is maintained by a working group of the Internet Engineering Task Force (IETF) called NETMOD[7]. YANG can be used to model configuration data sent to a network device as well as state data which is retrieved from the device[8].

YANG is protocol-independent and can be encoded into any format understood by the employed network configuration protocol, e.g., XML or JSON. Popular protocols using YANG are gNMI and Netconf. YANG contains a collection of built-in data types which can be extended to meet the needs of a specific application.

YANG models follow a tree format with a syntax mainly comprised of containers, lists, and leaves.

Example excerpt of a Cisco model[9]:

```
module system {
  container system {
    leaf host-name {
      type string;
      description "Hostname for this system";
    }
  }
  container login {
    leaf message {
      type string;
      description "Message given at start of login session";
    }
    list user {
      key "name";
      leaf name {
        type string;
      }
      leaf full-name {
        type string;
      }
      leaf class {
        type string;
      }
    }
  }
}
```

YANG models can use namespaces, prefixes, and imports to enhance their functionality further. This allows them to be very flexible but can also quickly lead to complex data structures.

A project called OpenConfig aims to provide pre-made YANG data models which can be used by both network device vendors as well as network engineers. They are designed in a way that is vendor-neutral and covers most widely and commonly used features[10]. Many vendors use the OpenConfig models as a base and extend them to meet their own needs through a mechanism called deviation, which describes where the vendor “deviates” from the OpenConfig standard and augmentation, which contains data for additional features. While OpenConfig provides standardization, the extensive use of deviations, however, makes the YANG models even more complex.

4.5 SDC

Schema Driven Configuration embraces the new philosophy for network automation. It is a declarative way to deploy configuration as K8s resources to a network device.

SDC is currently in development, and no Alpha or Beta versions have yet been released. The open-source tool is spearheaded by Nokia, with Wim Henderickx as the leader of this project. One peculiarity is the vendor-agnostic nature of SDC; they want to support various vendors, not just Nokia. But they still need some time until they are at this point. Another peculiarity is the reconciliation loop: When SDC sees that there is another configuration deployed than intended, it will re-deploy the configuration again to the endpoint. The benefits of Kubernetes, its stateless architecture, and CRDs (Custom Resource Definitions) are used, which provide the aspect of declarative configuration and the reconciling loop.



Figure 7 SDC Logo[1]

The devices are targets, which are connected via Netconf or gNMI and must support YANG. The targets can either be physical devices providing Physical Network Functions (PNF), virtualized devices providing Virtual Network Functions (VNF), container-based networking functions (CNF), or no operations (NOOP) for testing. For some devices, working YANG model repositories already exist, maintained either by the vendor or the community.

SDC consists of four different components: Config-Server, Schema-Server, Data-Server, and Cache, which communicate with each other as described below:

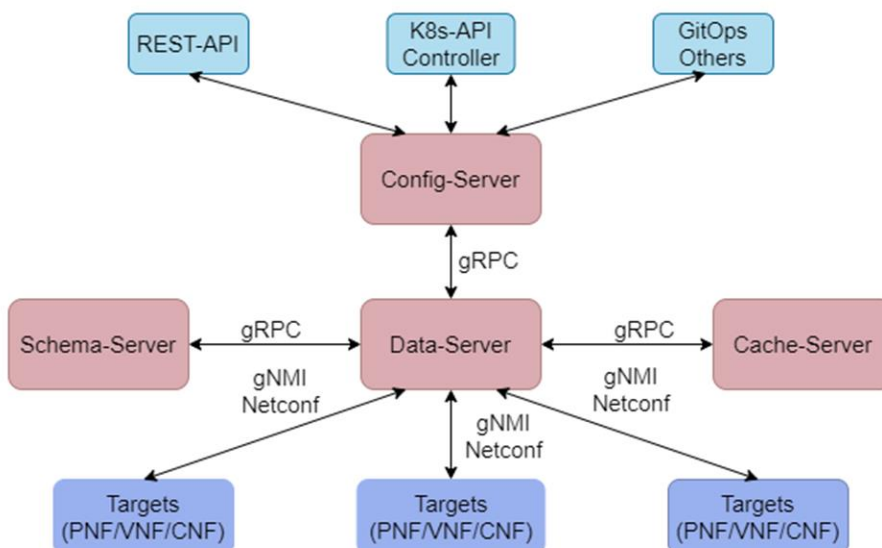


Figure 8 SDC components[11]

Config-Sever

The config-server is a Kubernetes-based operator. It is the brain of this tool and is responsible for different aspects[11]:

- It interfaces to the Kubernetes Cluster and other APIs, for example, by receiving the declarative configuration intents as a YAML and validating them against the schema.
- Schema-Controller: Manages the lifecycle of schemas as CRs.
- Discovery-Controller: Manages the lifecycle of targets through discovery.
- Target-Controller: Manages the lifecycle of endpoints as CRs and manages them in the data-server.

Data-Server

The data-server is a stateless service which cross connects the config-server, schema-server, cache and the targets. The connection among each other is realized via gRPC. In the current release, the data-server can use either gNMI or Netconf to interface with the targets (network devices).

Schema-Server

The Schema-Server is a repository for schemas, which can be from different vendors. The schemas are loaded from different Git repositories.

Cache

The cache manages datastores, config, and state of the targets and stores intent data. In addition, it can use persistent storage.

Workflow

SDC consists of different parts, which are described above. Now, the focus lies on how the user is interacting with SDC and which steps are required to run SDC and deploy a configuration at the end.

The first step is to apply on K8s the file[12] called colocated.yml. This will create Custom Resources Definitions CRDs in Kubernetes for SDC and deploy the Pods with the operators.

Now, the CRs have to be applied; the first one is the YANG schema. This file includes one or more repositories which hold the YANG models for one specific device. The YANG model repositories are usually provided by the device vendors themselves.

Later, the Target Connection Profile can be applied. It is used to define the network protocol for the target and the port of the target. The Target Sync Profile defines the device's synchronization options. The Target Secret Profile holds the credentials of the target as a Kubernetes Secret. Secrets are



Figure 9 SDC Workflow

applied here in plaintext. But they aren't exposed during the workflow of creating Pods – so the exposition surface is minimal. The Discovery Addresses describe the IP addresses of the targets and the target connection profile used. It is possible to attach labels for easy selection of the targets later in the lifecycle.

In the last step, the device-specific configuration will be applied. An example of a configuration can be found in Figure 13 Specific Device Configuration. In the section about Choreo, the creation of the configuration will be illustrated.

Manifests Overview

Manifest	Use
Schema	References repository containing the YANG data models
TargetConnectionProfile	Specifies protocol, port, encoding and security options for device connection
TargetSyncProfile	Specifies protocol, port, encoding, path (scope) and frequency of state data retrieval from the devices.
Secret	Specifies the credentials used to connect to the devices
DiscoveryRule	Specifies target devices to find and manage and assigns schema and profiles to use
Config	Contains the configuration data for one or more devices in the structure defined by the schema

Figure 10 Overview of SDC Manifests

4.6 Configuration Generators

The following three tools for configuration generation were suggested by the advisors. They were chosen because they were all recently developed and provide promising functionality.

4.6.1 Kubenet

Kubenet is a framework consisting of multiple components: pkgserver, Kuid, kuidapps, and SDC. Together, they provide the capabilities to use Kubernetes for network automation.

Kuid: Kuid is used to manage inventory resources and identifiers used for network configuration in the Kubernetes environment. Those can be devices, IP prefixes, VLANs, Links, etc. They are represented as CRs and are used by kuidapps.

Kuidapps: Kuidapps are used to build the abstract configuration. They contain reconcilers for network topologies, devices, links and other elements. They also introduce more CRDs such as “networkconfigs.network.app.kuid.dev”, which defines IP version, encapsulation types, prefixes and protocols such as BGP-EVPN and OSPF.

Pkgserver: The pkgserver manages sets of Kubernetes Resource Model (KRM) resources in YAML files as packages. It interfaces with git repositories and enables reading resource files from git and writing the derived device specific configuration back to a repository for review instead of transacting it directly to the device.

SDC: see Chapter 4.5

4.6.2 Choreo

Choreo is the newer version of Kubenet, released by the team that is also developing SDC. Choreo does not have the same capabilities and is being created based on a different variant of Kubenet. Their goal is to simplify the usage, enhance automation and orchestration of systems [13]. SDC already had the declarative aspect of Kubernetes integrated, and they wanted to include this in Choreo as well.



Figure 11 Choreo KForm Logo[42]

Its main difference is that it runs without Kubernetes but has the same structure as the Kubernetes YAMLS; they reproduced parts of Kubernetes with Choreo. Kubernetes can be used for way more than running applications as micro-services. The idea of using Kubernetes to deploy configurations to network targets is quite a new philosophy in the cloud-network world, and maybe other tools used for the ecosystem aren't ready for this integration yet.

Choreo runs on Linux, WSL, and Darwin OS. The mindset is to be as simple as possible for the network engineer. Choreo introduces BYOS and BYOA - Bring your own Schema and API. In this thesis and with this tool, our goal is to render functional intent configuration into YAML. For example, the authors don't know how to connect two switches (Abstract Config), switch01 and switch02, and they are already physically connected together via Ethernet port1. The next step in Choreo is to generate an abstract device configuration. It has expanded topology data and is device-centric but still vendor-neutral.

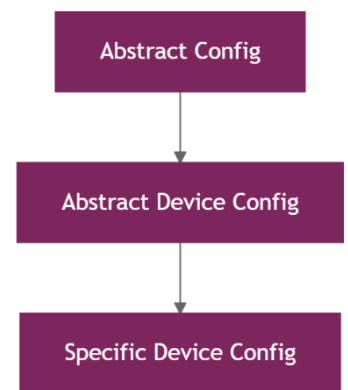


Figure 12 Diagram Configuration Rendering

In the end, SDC needs a YANG-based configuration intent in YAML. The Input config file for SDC could look like this:

```
test_arista_v051 > config > ! config_hostname.yaml
1  apiVersion: config.sdcio.dev/v1alpha1
2  kind: Config
3  metadata:
4    name: intent1-arista
5    namespace: default
6  labels:
7    config.sdcio.dev/targetName: eos1
8    config.sdcio.dev/targetNamespace: default
9  spec:
10  priority: 10
11  config:
12  - path: /system
13    value:
14      config:
15        hostname: arista-01
```

Figure 13 Specific Device Configuration

Here in Figure 13 Specific Device Configuration, the hostname of target eos1 will be changed. In this project's use case, the configuration file would need to set an IP address at the specific interface for each device. This configuration should not only be the input for SDC but also be generated by Choreo.

Reconciling Logic

Choreo implements the Kubernetes approach of event-based logic, also known as KEDA[14]. Kubernetes resources are declarative objects which describe the desired state. Controllers, sometimes called reconcilers, are also used. These provide control loops that watch resources and update these based on events. In this workflow, the desired state must be defined as a resource manifest, and the controller registers for events on this new resource. The controller waits until a resource is changed to initiate a trigger for the reconciler. Upon being called, the reconciler compares the actual state with the desired state and the subscribed resources will be created, updated, or deleted.

As an input Choreo would receive the two targets, the physical interface where they are connected and the IP pool for the interfaces. In addition, Choreo should manage resource-based handles like IPAM, VLAN, AS, and GENID. Those should be provisioned by reconcilers, who claim and register resource identifiers. With this information Choreo will generate the abstract configuration.

Interface to SDC

An interesting feature is the `--sdc` flag of the CLI, which enables SDC. According to the developers, the idea behind this flag would be to include the whole SDC into Choreo. They copied the code to Choreo because their intention was to run SDC without a Kubernetes cluster. They often received feedback from network engineers that Kubernetes would be too complicated for them. To minimize the entry threshold, they introduced the feature in Choreo to run SDC locally on the notebook.

So, after Choreo generates the device-specific config, it will be validated and later actuated by the SDC part. Within the validation, the running config from the node will be loaded and merged with the generated device-specific config. In addition, a built-in YAML-linter will also run to prevent failures.

Choreo Implementation

```
ins@ubuntu-L:~/test_arista_v051$ choreoctl --help
choreoctl is client tool to intercat with choreo

Usage:
  choreoctl [flags]
  choreoctl [command]

Available Commands:
  api-resources  get api resources from the resource
  apply          apply resource
  branch
  completion     Generate the autocompletion script for the specified shell
  delete        delete resource
  dev
  get           get resource
  help         Help about any command
  run
  server
  tui          tui resource
  version      show choreoctl version

Flags:
  --address string      address the server is listing on (default "0.0.0.0:51000")
  --apis string        the path where the api manifests are located (default "crds")
  -b, --branch string  branch from which the client wants to retrieve the info
  --cacheDir string   cache directory where the api resource information is stored (default "/home/ins/.config/choreoctl/cache")
  --config string     configuration where the client context is stored (default "/home/ins/.config/choreoctl/config.yaml")
  --db string         the path where the db manifests are located (default "db")
  -d, --debug         enable debug mode
  -h, --help         help for choreoctl
  --input string     the path where the input(s) are located (default "in")
  -r, --internalReconcilers  enable internal reconciler
  --libraries string  the path where the libraries are located (default "libs")
  --maxRecvMsg int   the maximum message size in bytes the client can receive (default 25165824)
  -n, --namespace string  If present, the namespace scope for this CLI request (default "default")
  --post string      the path where the postprocessing files are located (default "post")
  -p, --proxy string proxy context from which the client wants to retrieve the info
  --reconcilers string  the path where the reconciler manifests are located (default "reconcilers")
  --refs string      the path where the ref(s) are located (default "refs")
  --runningConfigs string  the path where the running config(s) are located (default "runningconfigs")
  --schemas string   the path where the schema(s) are located (default "schemas")
  -s, --sdc         enable sdc
  --servername string  the name of the server (default "choreo")
  --timeout int      gRPC dial timeout in seconds (default 10)

Use "choreoctl [command] --help" for more information about a command.
```

Figure 14 Choreoctl Help Command

In the Figure 14 Choreoctl Help Command, `choreoctl` is used, which is the CLI utility for Choreo.

What is revealed while following the workshop is that it takes a little bit more to work with Choreo fully. It isn't as easy as bringing your own schema or API, and everything else will work out. *In addition*, you are required to write your own reconciler in Starlark, a Python dialect. Starlark is a simple Python-like language designed to be embedded in other applications to provide configuration or scripting capabilities[15].



Figure 15 Starlark symbol[43]

Besides writing your own reconciler, you are required to write the configuration for the reconciler and the YAML file with the working resource. The aspect of BYOS looks simple, but it isn't that easy. It is required to develop all the templates and schemas in Jinja2 or Go. This must be done for every different OpenConfig device OS release. Previous work with OpenConfig has already proved that the repositories aren't well maintained and need additional support from the vendors or from a third party. As the experiments with OpenConfig continued, the authors also set up several Git repositories with different kinds of structures and files.

As seen, Choreo still takes a bit to be functional. Choreo is a powerful tool that can be used for way more than our use case. It is highly customizable, and almost everything can be done with Choreo. Because of this, it is more complex than Infracore, but allows you to also work with SDC and provides the reconciling functionality. However, the authors remain excited to see it when it's fully functional and the integration to SDC is provided. An option to automatically apply the newly generated device-specific configuration to SDC would have been preferred.

4.6.3 Infracore

Infracore is an opensource Infrastructure as Code for Everyone tool. It is an IoC implementation that is usable by everyone and is vendor-independent, like Choreo. Infracore already has one definitive version, but it is still in development. They provide good



Figure 16 Infracore Logo[44]

support if something is not clear or does not work, according to the experience of our colleague Simon Linder, who has already worked with Infracore. Infracore offers a clear and intuitive GUI, which is, however, still a little bit buggy, but the main interface to Infracore is GraphQL.

Behind the curtains, Infracore uses a NEO4J graph database to store schema-based data. It is also possible to connect a git repository to keep track of scripts, templates, and functions. Infracore supports the use of branches that are automatically reflected in the connected repository. It can, therefore, be used as the Single Source of Truth (SSOT). The SSOT also includes resource management and allocation of IP addresses, VLANs, and ASNs. SDC also needs the files for the target discovery, target secrets, sync profile, and so on. Speaking of Git, as it is integrated here, the CI/CD approach can be used to process the artifacts further. They could be deployed to SDC via a CI/CD pipeline.

In comparison to other tools like Netbox[16], Infracore does not contain any schemas out of the Box. In Netbox, a schema is already defined, describing many standard resource types, i.e., a device would already have relationships to an interface, module, and device type. All this must first be defined in Infracore – which has the benefit of a more clear and less overwhelming GUI but needs more effort.

Workflow

Some steps are required to work with Infracore. The first step is to analyze the target, the device which should be provisioned at the end. Which parts should be included in the configuration? Which resources need to be allocated? Should the management interface also be visible and manageable?

With the analysis, a logical concept, the target schema, is created. As you can see, the SDC schemas and schemas here are different. When the IP pool and other resources are known, the resource management can be mapped to Infracore. The next step is to create the transformations.

Transformations are generic plugins that convert a dataset into a different format. A transformation could be a Jinja2 template or a function written in Python. The Output of the Python function is a JSON file, and from Jinja2, a plain text file[17]. However, more research revealed that it is possible to use Python functions and have a plain text file. So, both transformations can be used to generate YAML. The output of the transformation should be a YAML file like in Figure 13 Specific Device Configuration.

The next step is to register a device in Infracore and fill out the forms, which were created with the Schema. Now, it has to be defined from which resource pools the target should obtain the IP addresses, VLANs, or ASNs and which transformations should be used. After starting the transformation (like changing some data in the repository), the output can be saved as artifacts to an S3-like store.

At last, the artifact must be applied to the K8s Cluster. This can be achieved using different means – it could be applied via CI/CD pipeline to Kubernetes or via the pkgserver[18]. It would also be possible to use other tools that provide rollouts with Custom Resources CRs.

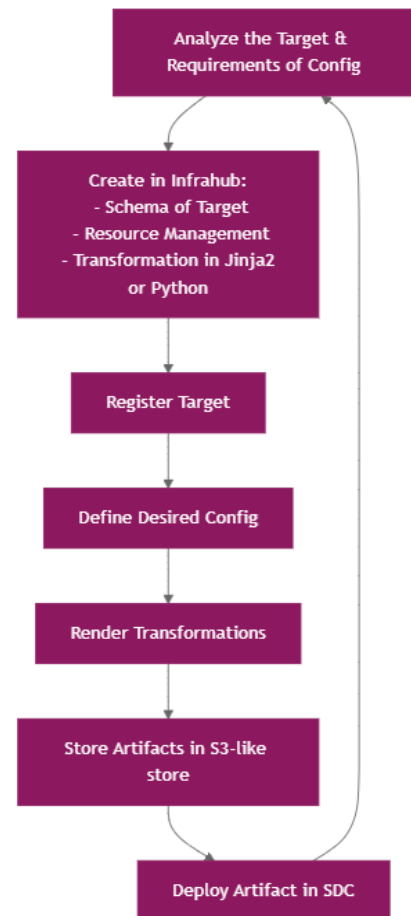


Figure 17 Workflow of using Infracore

4.7 Target Device Vendor

The target devices are specified as the vendors of routers and switches. The importance is here if the protocol is supported both by the vendor and the SDC.

4.7.1 Cisco

Cisco is widely used by the economy and is already known by both team members. In contradiction, Cisco does not have any free-to-use OS images for virtualized devices. The INS has some images to run as a container, but they aren't on the most recent version. In the lab, there are some physical switches that may also be used.

4.7.2 Arista

Urs already worked together with Arista, a vendor known for their better integration of Open-Config models. There are OS images for containers available to download – only a registration is needed. This is an amazing philosophy in network automation because no one else handles this transparently.

4.7.3 Nokia

Nokia SR Linux, the OS for virtualization, was utilized for pre-testing. However, as the authors have no prior experience with Nokia products, they only have limited insight into how well the protocols and tools are integrated.

4.7.4 Juniper

Juniper, the same as Nokia, is unfamiliar to the team members or the supervisors. Therefore, it is uncertain how effectively the integration performs together. In addition, the Juniper OS images aren't free to use.

5. Requirements

5.1 Use Cases

5.1.1 Actors

The following personas are used for the Functional and Non-Functional Requirements.

Network Admin: A person who knows how the cluster works. The network admin is a persona who uses the cloud-native network controller, deploys new EVPN VXLAN fabrics, and manages them.

User: A person who interacts with the cloud-native network controller. This could be a support engineer or a supervisor.

DevOps: A person who works in the field of software development and IT operations. The DevOps persona doesn't know how the underlying network works and only needs to connect two or more end systems.

5.1.2 Use Case Description

5.1.2.1 Required

UC 1 Manage Underlay-Network

As a Network Admin, I want to be able to manage the underlay network.

UC 2 Predefined Configuration

As a Network Admin, I want to deploy a predefined configuration to enable EVPN functionality on the fabric.

UC 3 Manage Overlay-Network

As a Network Admin, I want to be able to manage the overlay network.

UC 4 Golden State

As a Network Admin, I want to be able to have a golden state. If the configuration fails, I want to be able to revert to a working state.

UC 5 Vendor Independence

As a Network Admin, I want to be able to use network devices from different vendors.

5.1.2.2 Optional

UC 6 Automated Resource Deployment

As a Network Admin, I want the desired configuration state for EVPN-VXLAN to be deployed automatically.

UC 7 Manage Multitenancy

As a Network Admin, I want to manage multiple tenants for EVPN.

5.1.2.3 Out of scope

UC 8 Configuration Drift

As a User, I want to be able to detect configuration drift and revert them if necessary.

UC 9 GUI

As a user, I want to manage the controller using a GUI.

UC 10 Monitoring

As a User, I want to detect errors and hardware defects with a monitoring tool.

UC 11 Management Access

As a Network Admin, I want to onboard network devices to enable management access.

5.2 Non-Functional Requirements

NFR 1 Cloud-native

The Controller should be Cloud-native. In addition, the controller should be Kubernetes-based.

NFR 2 Provisioning

The EVPN Fabric with the Underlay and the Overlay network should be provisioned within 5 minutes.

NFR 3 Usability/Learnability

A DevOps Engineer with basic Git and YAML skills should be able to deploy an EVPN-VXLAN Fabric.

6. Challenges

While trying to analyze the fulfillment of the use cases, the authors encountered many challenges. A few were possible to be solved by themselves or with the help of other people. In summary, the team had at least one issue nearly every week. For a better overview of the challenges in the depiction below, a timeline is illustrated with a short description.

The span of the challenges is the time spent by the team members. As seen in Figure 18 Timeline of encountered Challenges, four challenges were at work at one time.

Timeline of encountered Challenges

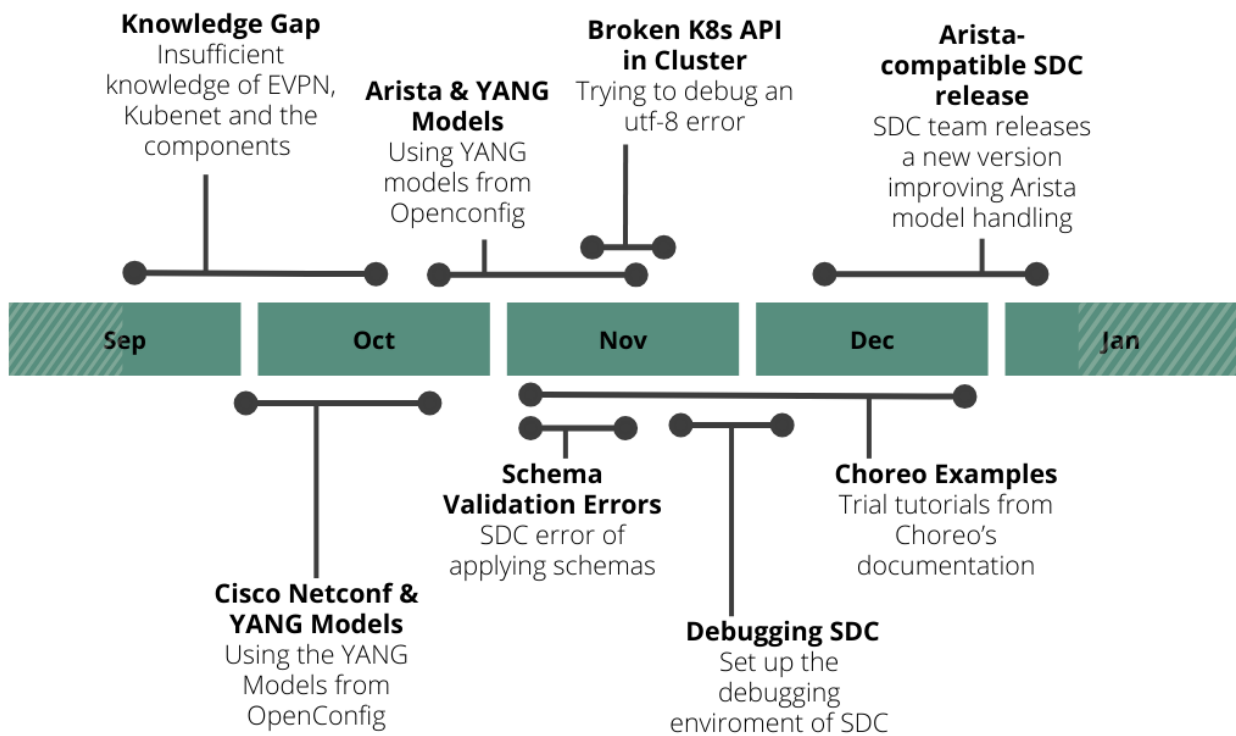


Figure 18 Timeline of encountered Challenges

6.1 Knowledge Gap

Situation at Hand

In the first few weeks, the authors had to gain knowledge on what exactly EVPN-VXLAN is, as this topic wasn't covered in class for them because the classes were restructured, and they fell in the gap between them.

Tried Remedies

To gain practical competence with EVPN-VXLAN, the team completed the EVPN-VXLAN labs, but it took longer than expected to complete. The authors learned that the labs were shortened in the meantime for the regular courses. For them it was the perfect opportunity to get to know EVPN-VXLAN and have a better comprehension about it, providing a fundamental understanding for the thesis. At this point, the authors were able to configure this structure on Cisco routers and manage it for daily usage with Cisco. Later, they became familiar with SDC and its components. At this point, they tried to analyze how SDC and other tools could work together. The diagram below shows their understanding of the basic workflow within the ecosystem at the time.

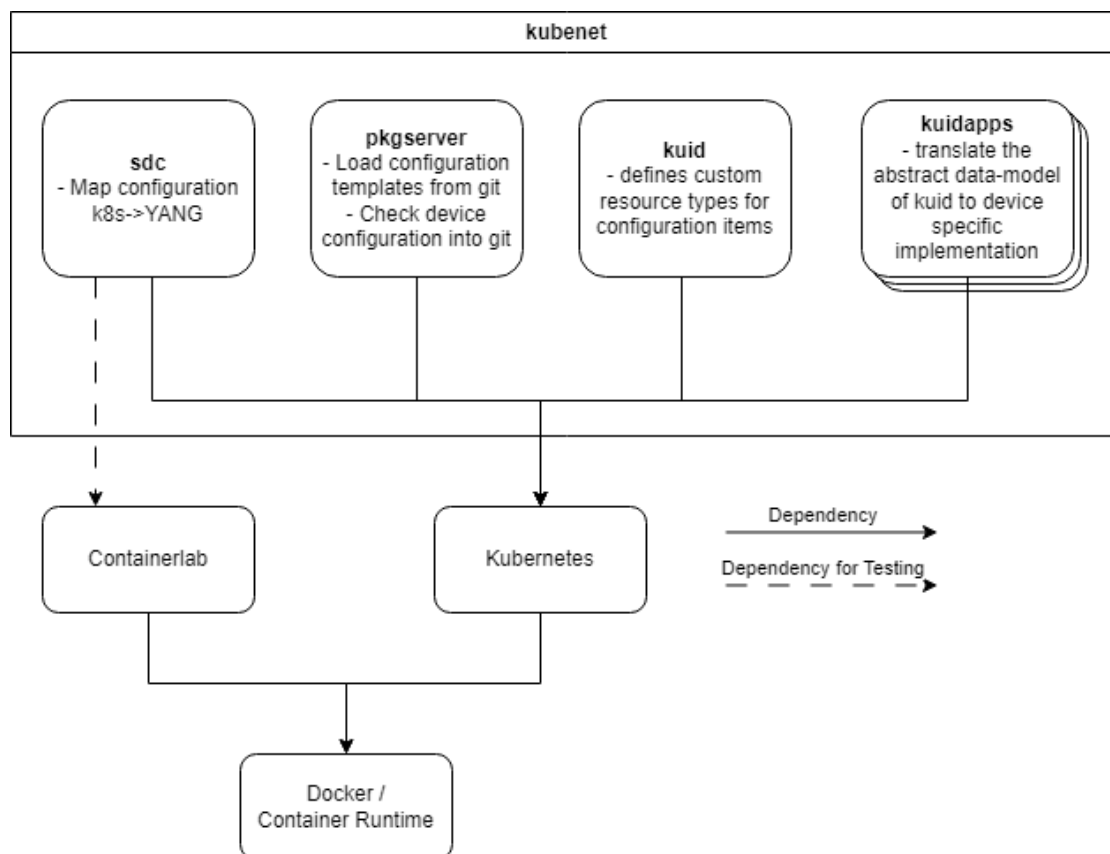


Figure 19 SDC and Kubenet

The components are decoupled from each other and have distinct responsibilities. Later in the architecture analysis phase, the authors became more familiar with SDC, but they still couldn't completely comprehend the responsibility of each service in the interaction of the components in the overall system. The examples on the SDC's website were tested, and the amount of resources required

in the K8s cluster, depending on its working state, became apparent. Therefore, an Ubuntu instance was created as a VM in the network lab infrastructure of the INS. Otherwise, it wouldn't be possible to work on the same notebook and run the kind cluster as well as virtualized network device targets in VMs, as there was too much lag. At this point, the authors received a glimpse of how schema-driven configuration works. Now, the question is how the business logic could be implemented: How can a configuration be created? At this time, the authors found out how the OpenConfig model structure is defined for this model and built a YAML configuration intent file according to the OpenConfig structure. The next step would be how to create the YAML configuration file without writing it by hand each time. A tool is needed that takes the affected routers and the desired configuration as input and provides the output as a YAML file, which can be further processed by SDC. The developers from SDC are also developing this tool. On their website[19] which is why the authors can't make sense of it or have any clue about how the components interact with each other. Their understanding of the whole ecosystem looks like the diagram below. The Kuid is a part of the Kubenet.

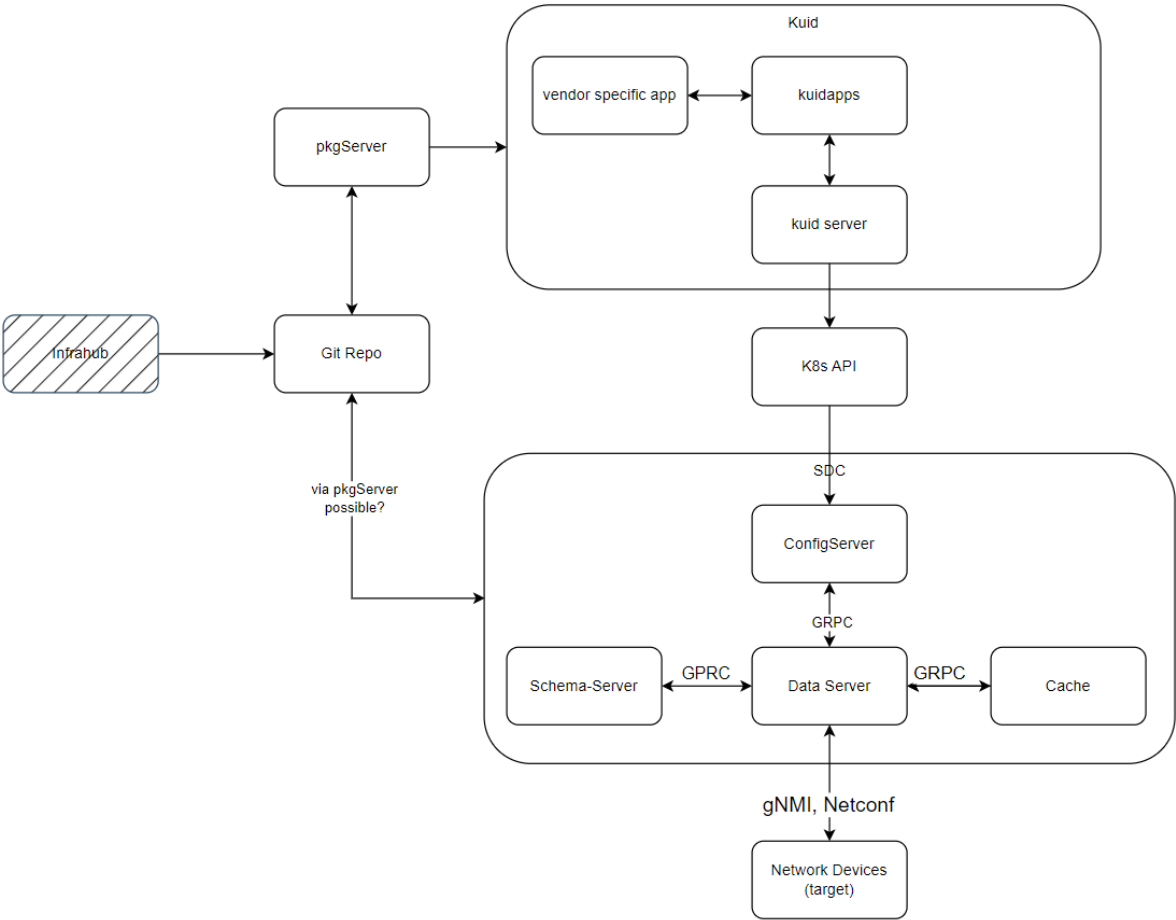


Figure 20 Components of SDC and Kuid

As a summary of this diagram, the pkgserver is also a part of Kubenet, which should manage the Git access. Our question here was if the pkgserver should truly replace tools like Argo Rollouts. On the other hand, there are the Kuid components. All that is known to the authors is that they exist, but not how they interact with each other. It isn't visible how Kubenet communicates with the pkgserver and to the K8s API or if Kubenet directly communicates with the Config-Server.

Later, the Kubenet Git repository[20] was inspected. It contains one file, which renders the abstract configuration into YAML. It has more than 1500 lines and looks like it would already render to a Nokia-specific configuration, something which would be done by SDC with the schemas before being sent to the endpoint – according to the authors' assumption at this moment. Because of this black box, Kubenet, it was decided to arrange a meeting with the main developer of SDC and Kubenet, Wim Henderickx. During the meeting, he explained the current development and pointed out that Kubenet would be replaced by Choreo. There is, however, no information on any platform about this change. Only two weeks later, the authors received a link to a preview Git repository for trying Choreo. Also, during the meeting, he explained how SDC and Kubenet fundamentally work together. The 1500-line file was discussed, and the explanation was provided that this is only used for validation. For us, this is a little bit obscure because this tool should also be usable for other vendors, not only Nokia.

At this point, the authors had to wait until there would be a more comprehensive release for Choreo. Choreo was put aside for now, and the focus was placed on SDC and its capabilities instead.

Solution

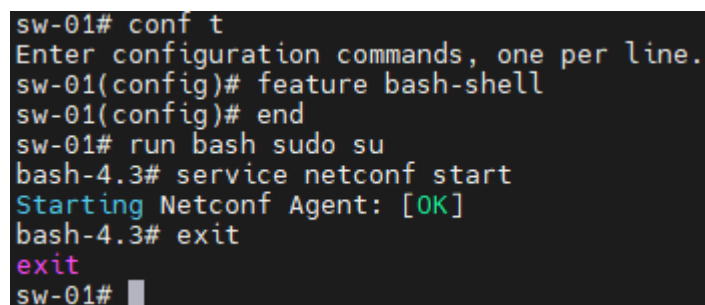
Many resources were used to close the knowledge gap. They completed the EVPN-VXLAN lab to gain expertise. A meeting was set up with the developer from the pkgserver, SDC, and the Kuid Apps to receive insight into the current developments and architecture design approach. Choreo will replace the Kuid Apps.

6.2 Cisco Netconf and YANG Models

Situation at Hand

The virtualized NX-OS in version 9.2 from Cisco was first used for the PoC. In the default configuration, the Netconf service is disabled and can only be enabled by first enabling the bash-shell in the CLI and then enabling the service via bash. Using the feature command as described in Cisco's guide[21], does not work:

```
sw-01(config)# feature netconf
^
% Invalid command at '^' marker.
sw-01(config)# feature net?
  netflow  Enable/Disable NetFlow
```



```
sw-01# conf t
Enter configuration commands, one per line.
sw-01(config)# feature bash-shell
sw-01(config)# end
sw-01# run bash sudo su
bash-4.3# service netconf start
Starting Netconf Agent: [OK]
bash-4.3# exit
exit
sw-01#
```

Figure 21 Enable the Netconf Service through Bash.

Tried Remedies

With the Netconf service enabled, loading the YANG model from this repository[22] through SDC was tried. According to the SDC User Guide[23], the schema repository can be referenced either by tag or by branch. As the used repository contains no tags, the branch option was tried. The schema load, however, failed because the tag reference was broken in SDC at that time. (It was fixed in a later release.) To continue testing, the authors forked the repository and created a tag of their own. At this point, the native device Cisco model found in “Cisco-NX-OS-device.yang” on the repository was used.

After successfully changing the hostname of the device, the authors started to try other sections to achieve the goal of configuring EVPN-VXLAN. Enabling the features EVPN and PIM worked without any problems. The BGP instance could be created, and the peers added. The PIM instance with the Anycast rendezvous point peers did not result in the expected configuration on the device.

Explanation

For completeness, the OpenConfig models in the Cisco YANG repository were also tried. With these, however, not even the hostname could be changed. A look at the corresponding deviations Cisco makes from the OpenConfig System model[24], reveals the answer why: It is just not supported.

```
31     deviation /oc-sys:system/oc-sys:config/oc-sys:hostname {
32     deviate "not-supported";
```

Figure 22 Cisco hostname Deviation from OpenConfig in NX-OS 9.2-3

Cisco did not support the hostname leaf of OpenConfig until NX-OS version 10.4-1. It was then deviated with a format constraint.

```
43     deviation /oc-sys:system/oc-sys:config/oc-sys:hostname {
44     deviate replace {
45     type string {
46     length "1..64";
47     pattern
48     '(((([a-zA-Z0-9_]([a-zA-Z0-9\_-]){0,61})?[a-zA-Z0-9]\.)*' +
49     '([a-zA-Z0-9_]([a-zA-Z0-9\_-]){0,61})?[a-zA-Z0-9]\.?)' +
50     '|\.);
51     oc-ext:posix-pattern
52     '^(((([a-zA-Z0-9_]([a-zA-Z0-9\_-]){0,61})?[a-zA-Z0-9]\.)*' +
53     '([a-zA-Z0-9_]([a-zA-Z0-9\_-]){0,61})?[a-zA-Z0-9]\.?)' +
54     '|\.)$';
55     }
56     }
57     }
```

Figure 23 Cisco hostname Deviation from OpenConfig in NX-OS 10.4-1

6.3 Arista and YANG Models

Situation at Hand

The authors switched to Arista as a vendor because it is supposed to have better support for OpenConfig. Right off the start, there appear to be problems loading the models into SDC. OpenConfig requires standard models from IETF and IANA. However, the Arista YANG repository[25] does not contain them. Therefore, an own repository was again created to include all models, the ones from IETF and IANA, the OpenConfig ones as well as the Arista-specific augments and deviations – which still exist despite Arista’s support for OpenConfig. But loading the schemas separately to deploy a configuration to change the hostname didn’t work either:

```
Spec:
  Config:
    Value:
      System:
        Config:
          Hostname: arista-01
  Lifecycle:
    Priority: 10
  Status:
    Conditions:
      Last Transition Time: 2024-10-31T06:38:06Z
    Message: rpc error: code = Unknown desc = unknown object "system" under container "__root__"
    Reason: Failed
    Status: False
    Type: Ready
  Events:
    Type      Reason      Age          From          Message
    ----      -
    Warning   config      17m (x1087 over 14h)  ConfigController  setIntent failed, err: rpc error: code = Unknown desc = elem:{name:"standard"}; key not found
    Warning   config      2m1s (x27 over 14h)  ConfigController  setIntent failed, err: rpc error: code = Unknown desc = unknown object "system" under container "__root__"
```

Figure 24 Error when configuring Hostname on Arista

Tried Remedies

Since SDC’s documentation on how the models need to be structured is not comprehensive enough, the authors had to resort to trial and error, for example, placing all models in the same directory. The main developer, Wim Henderickx, was also contacted, and he informed us of the problem of having the IETF and IANA models in a second repository. The discovery of this missing feature led to an update of the CRD for the schema, which now allows the supply of multiple repositories. However, even with the exact schema manifest proposed by Wim, schema loading was still error-prone and inconsistent and often resulted in an error similar to the following, preventing the schema from being loaded and getting ready:

```
Events:
  Type      Reason      Age          From          Message
  ----      -
  Warning   schema      115s (x531 over 5d22h)  SchemaController  cannot create schema, err: rpc error: code = Unknown desc = schema entry "routing-policy/defined-sets/neighbor-sets/neighbor-set/name" not found
```

Figure 25 Schema load Error using proposed Schema Manifest

In the end, the most promising results stemmed from using a custom repository consisting of two folders while placing all models containing import statements in one folder and those with no such statements in the second one.

Solution

In the meantime, there was an updated release of SDC, which now allows the user to also use a branch as a reference for model files and define two repositories for a schema. Configuring the hostname on Arista devices is now possible.

This schema could now be loaded to the ready state with the following manifest:

```
apiVersion: inv.sdcio.dev/v1alpha1
kind: Schema
metadata:
  name: two-folder.openconfig.test
  namespace: default
spec:
  provider: two-folder.openconfig.test
  version: 0.0.1
  repositories:
  - repoURL: https://github.com/patrick-lenherr-ost/openconfig-2-folder
    kind: branch
    ref: main
    dirs:
    - src: .
      dst: .
  schema:
    models:
    - models
    includes:
    - include
```

To verify the correctness of the data loaded into the schema, a small tool provided by SDC is started in a temporary pod in K8s and used to query the paths.

```
$ kubectl run -ti --rm sdctl --image=ghcr.io/sdcio/sdctl:v0.0.8 --restart=Never
--command -- /bin/bash
```

```
sdctl:/app$ /app/sdctl -a data-server.network-system.svc.cluster.local:56000
schema get --vendor two-folder.openconfig.test --version 0.0.1 \
--path /system/config/hostname
<request removed>
```

```
response:
schema: {
  field: {
    name: "hostname"
    description: "The hostname of the device -- should be a single
domain\nlabel, without the domain."
    namespace: "http://openconfig.net/yang/system"
    prefix: "oc-sys"
    type: {
      type: "string"
      length: {
        min: {
          value: 1
        }
        max: {
          value: 253
        }
      }
      type_name: "domain-name"
      patterns: {
        pattern: "((((([a-zA-Z0-9_]([a-zA-Z0-9\\-_]){0,61})?[a-zA-Z0-9]
9]\\.)*([a-zA-Z0-9_]([a-zA-Z0-9\\-_]){0,61})?[a-zA-Z0-9]\\.?)|\\.)"
      }
    }
    module_name: "openconfig-system"
  }
}
```

6.4 Schema Validation Errors

Situation at Hand

After successfully changing the hostname of the device via SDC, it was time to make larger configuration changes, i.e., interface settings. Whenever the authors tried to set an interface leaf, however, SDC reported a schema validation error – meaning the configuration was not valid against the loaded schema.

Tried Remedies

However, it can clearly be seen that config *should* be a container under interface (Using an unnamed list for the interface was also tried, as OpenConfig defines interface as a list rather than a node.[26]):

```
9 spec:
10   priority: 10
11   config:
12     - path: /
13       value:
14         interfaces:
15           - interface:
16             name: Ethernet1
17             config:
18             description: "S1 -> L1"
```

Figure 26 Interface Test Configuration #1

```
9 spec:
10   priority: 10
11   config:
12     - path: /
13       value:
14         interfaces:
15           - name: Ethernet1
16             config:
17             description: "S1 -> L1"
```

Figure 27 Interface Test Configuration #2

```
Status:
Conditions:
  Last Transition Time: 2024-11-11T15:38:45Z
  Message: rpc error: code = Unknown desc = unknown object "config" under container "interface"
  Reason: Failed
  Status: False
  Type: Ready
```

Figure 28 Schema Validation Error

```
$ .pyang -p ./../models/openconfig-interfaces.yang -f tree
module: openconfig-interfaces
  +--rw interfaces
    +--rw interface* [name]
      +--rw name -> ../config/name
      +--rw config
        +--rw name? string
        +--rw type identityref
        +--rw mtu? uint16
        +--rw loopback-mode? oc-opt-types:loopback-mode-type
        +--rw description? string
        +--rw enabled? boolean
```

Figure 29 The Interfaces Model of the Schema.

Next, the config node was left out of the hierarchy because SDC may have set the config node implicitly. With this attempt, SDC complained about missing required leaves, which are, however, part

of the “status” node. The leaves of the status node are for reading the current state of the device and are read-only.

It appeared that Ethernet Interfaces might be more complex than, for example, Loopback interfaces because they could be managed as subinterfaces. Therefore, the authors tried to disable the Loopback0 interface, which unfortunately led to the same result.

As another test for the correctness of our understanding of the schema path, the gNMI client gnmic was used to manually set the Loopback0 interface to disabled and back to enabled. This manual test worked flawlessly.

```
$ gnmic -a 10.8.33.250:6030 -u admin -p admin --insecure set --update-path
'/interfaces/interface[name=Loopback0]/config/ --update-value 'false'
{
  "source": "10.8.33.250:6030",
  "timestamp": 1731488703732116376,
  "time": "2024-11-13T10:05:03.732116376+01:00",
  "results": [
    {
      "operation": "UPDATE",
      "path": "interfaces/interface[name=Loopback0]/config/enabled"
    }
  ]
}

arista-01#sh run
! device: arista-01 (cEOSLab, EOS-4.33.0F-39050855.4330F (engineering build))
<excerpt>
interface Loopback0
  shutdown
!

$ gnmic -a 10.8.33.250:6030 -u admin -p admin --insecure set --update-path
'/interfaces/interface[name=Loopback0]/config/enabled' --update-value 'true'
{
  "source": "10.8.33.250:6030",
  "timestamp": 1731488714545226922,
  "time": "2024-11-13T10:05:14.545226922+01:00",
  "results": [
    {
      "operation": "UPDATE",
      "path": "interfaces/interface[name=Loopback0]/config/enabled"
    }
  ]
}

arista-01#sh run
! device: arista-01 (cEOSLab, EOS-4.33.0F-39050855.4330F (engineering build))
<excerpt>
interface Loopback0
!
```

Explanation

Because the manual test succeeded, the problem must be rooted in the verification of the configuration intent against the schema done by SDC or the loaded schema structure itself. Further investigations will include the need to debug the running SDC code and the way it processes the YANG models.

6.5 Broken K8s API in a test cluster

Situation at Hand

After some time of testing SDC, it became clear that the schema loading – even though it had worked now – was behaving inconsistently. Sometimes it worked, sometimes it didn't, and sometimes it worked after letting the K8s cluster sit for 8 to 12 hours. Another, as soon to be suspected, related, problem was that now not even the hostname change was working anymore. Instead, an error claiming a string would contain invalid UTF-8 was presented.

```
Last Transition Time: 2024-11-05T08:32:46Z
Message:             rpc error: code = Internal desc = grpc: error unmarshalling request: string field contains invalid UTF-8
Reason:              Failed
Status:              False
Type:                DatastoreReady
```

Figure 30 Invalid UTF-8 Characters

Moreover, a strange phenomenon was discovered: Sometimes, deleting a resource took a very long time or blocked the shell prompt altogether. Even after waiting an extended time, the resource would still be in the “deleting” state.

Tried Remedies

After a session of analyzing the cluster with Jan, the problem was discovered: During the tests, a new version of the config-server and data-server image was released. In the hope of solving the schema loading problems, the authors updated the deployment on the cluster with the new images. However, it was missed that some of the Custom Resource Definitions (CRDs) had changed as well, and these hadn't been applied. Through this mistake, the K8s API on the cluster was broken.

Solution

Since setting up SDC with its resources in the cluster doesn't require that much time, provided it works without errors, it was decided to scrap the cluster and instead set up a new one. On the fresh cluster, SDC was deployed with the newly released images and the latest version of CRDs. As a result, the UTF-8 error could be resolved, and the hostname could be set again.

6.6 Choreo Examples

First, as the team got in touch with Choreo, they read the documentation[13] for Choreo. The documentation was now more written as a selling webpage, not as a technical and detailed one. Because of this, it was time to have a look at the examples.

Situation at Hand

While working through the examples, the reconcilers couldn't be started and didn't produce the expected output from the examples[27]. Also, when executing the `choreoctl api-resources` command, not all resources are getting loaded. After running the reconciler with `choreoctl run once`, no reconcilers were found. As expected, the HelloWorld manifest wasn't generated. After

running `choreoctl get helloworlds.example.com test -o yaml`, to get the resources, no messages were printed out.

```
● ins@ubuntu-L:~/choreo-examples$ choreoctl api-resources
resource:"customresourcedefinitions" group:"apiextensions.k8s.io" version:"v1" kind:"CustomResourceDefinition" choreoAPI:tr
ue
resource:"libraries" group:"choreo.kform.dev" version:"v1alpha1" kind:"Library" categories:"choreo" choreoAPI:true
resource:"reconcilers" group:"choreo.kform.dev" version:"v1alpha1" kind:"Reconciler" categories:"choreo" choreoAPI:true
resource:"upstreamrefs" group:"choreo.kform.dev" version:"v1alpha1" kind:"UpstreamRef" categories:"pkg" categories:"knet" c
horeoAPI:true
● ins@ubuntu-L:~/choreo-examples$ choreoctl run once
loading ...
loading done
running reconcilers ...
completed
● ins@ubuntu-L:~/choreo-examples$ choreoctl get helloworlds.example.com test -o yaml
apiVersion: example.com/v1alpha1
kind: HelloWorld
metadata:
  annotations:
    api.choreo.kform.dev/origin: '{"kind":"File"}'
  creationTimestamp: "2024-12-12T09:03:48Z"
  name: test
  namespace: default
  uid: aa00be3f-30e8-4e2f-8345-9090af742f64
spec:
  greeting: hello choreo
○ ins@ubuntu-L:~/choreo-examples$
```

Figure 31 Choreo HelloWorld Reconciler Example not working

Tried Remedies

A week later, a commit was made to the Git repository, where the Choreo examples are located for a workshop at AutoCon. Additionally, a meeting was planned to show the functionalities of Choreo on Thursday, 12.12.2024. The developers updated the examples, and the reconcilers are now working! Running Choreo again, the examples are working and can be tested. The meeting with Wim Hendrickx was interesting for us as it gave us a deeper understanding of Choreo. Now, the greeting example works.

```
ins@ubuntu-L:~/choreo-examples/choreo-workshop$ choreoctl run once
loading ...
loading done
running reconcilers ...
running root reconciler hello-world
Run root summary
execution success, time(msec) 8.802503ms
Reconciler          Start Stop Requeue Error
helloworlds.example.com.hello-world 2    2    0    0
completed
ins@ubuntu-L:~/choreo-examples/choreo-workshop$ choreoctl get helloworlds.example.com test -o yaml
apiVersion: example.com/v1alpha1
kind: HelloWorld
metadata:
  annotations:
    api.choreo.kform.dev/origin: '{"kind":"File"}'
  creationTimestamp: "2024-12-12T15:39:08Z"
  finalizers:
  - helloworlds.example.com.hello-world
  name: test
  namespace: default
  uid: 0af9ae40-ce13-4239-922d-5a3415e8649c
spec:
  greeting: hello choreo
status:
  conditions:
  - lastTransitionTime: "2024-12-12T15:40:56Z"
    message: ""
    reason: Ready
    status: "True"
    type: Ready
```

Figure 32 Choreo HelloWorld Reconciler Example works now

Solution

This challenge could be solved by using the updates from developers. The examples aren't up to date with the workshop examples at this time.

6.7 Debugging SDC

Situation at Hand

Because of the previous problem, the schema validation errors, the consensus was that further investigation was needed. Based on this it was decided to debug the SDC code. It was obvious from the authors' trial-and-error efforts, that code adjustments were needed, which couldn't be solved by investigating the traffic on the pods itself.

Tried Remedies

The authors had a look at the chapter on development from SDC, which said that a tool called Telepresence is needed to connect to the namespace and intercept the traffic. Then, it would be possible to run the config-server locally in the IDE. Telepresence is used to impersonate other components which aren't run locally. It can intercept the traffic and directly forward it to the developer's machine or can place an init-container in the namespace, which would investigate the traffic for us and later send it to the developer's machine. It became clear only later that it would also be possible to run it without an init-container.

The development page from SDC only had a few commands and nearly no description at all. The commands on the SDC development page[28] were followed. However, there wasn't much success. It was possible to install telepresence and connect to the namespace, but interception of traffic was not possible at all.

```
vani@nbvg01-1040g8:~$ telepresence intercept config-server-api --workload config-server --service config-server --port 6443:api-service
telepresence intercept: error: connector.CreateIntercept: Back-off restarting failed container tel-agent-init in pod config-server-7f87c494bb-2md2f_network-system
The logs of Pod config-server-7f87c494bb-2md2f might provide more details
```

Figure 33 SDC Debugging Telepresence Intercept Error

Since the authors didn't have much debugging experience in these constructs before and didn't know if it was a problem or lack of experience they reached out to the supervisors. Together with Jan, they first cloned the config-server repo and tried to run the make file as it was described in the readme file[29] from their Git repository. An entire afternoon was dedicated to this problem and the question of how the make file could run without errors. This was frustrating for all involved because why should there be a Makefile, which doesn't run properly? After the four-hour session, it was decided to reach out to developers of SDC. It does look like no one outside the development team had been trying to debug the code before.

Wim Hendrickx helped us and created a private channel on Discord. First, the same versions of telepresence that were used by the developers v.2.20.2 were installed, but there was still no luck. After asking if they did something in addition, they informed the authors about the command:

```
telepresence helm upgrade --set
client.routing.allowConflictingSubnets="{10.0.0.0/8}"
```

This command allows the routing of this specific subnet, which could conflict with other routes on the local machine. But still, a look at the logs showed that the error persisted. It looked like the pod didn't know the iptables command. The thought was that the authors might have a different Linux Kernel, where the K8s Cluster is running, than the others.

So, replication of the error on a different system was tried. In the INS Network Garden, the newest Ubuntu 22.04 LTS was deployed, kind installed, and SDC was set up on a kind cluster. Some SDC resources were also applied, and telepresence was installed on the cluster. Unfortunately, the error persisted. Therefore, one last attempt was made before reaching out to Telepresence, Kind, SDC, and Telepresence were installed on an Ubuntu VM on the authors' notebook. Here, the same error appeared – it was time to reach out to Telepresence.

The authors reached out to Telepresence via Slack and posted their question[30] with the following errors:


```

patrick@TITAN-BOOK:~/kind/images/base$ k -n network-system logs config-server-5cf9686c9-mrbxv
Defaulted container "config-server" out of: config-server, data-server, traffic-agent, tel-agent-init (init)
Error from server (BadRequest): container "config-server" in pod "config-server-5cf9686c9-mrbxv" is waiting to start: PodInitializing
patrick@TITAN-BOOK:~/kind/images/base$ k -n network-system logs traffic-manager-7578869448-zlkw
2024-11-13 14:04:19.762z info OSS Traffic Manager v2.18.0 [uid:1080,gid:0]
2024-11-13 14:04:19.763z info Using traffic-agent image "docker.io/datawire/tel2:v2.18.0"
2024-11-13 14:04:19.785z info Extracting service subnet 10.96.0.0/16 from create service error message
2024-11-13 14:04:19.785z info Using podCIDRstrategy: auto
2024-11-13 14:04:19.785z info Using AlsoProxy: []
2024-11-13 14:04:19.786z info Using NeverProxy: []
2024-11-13 14:04:19.786z info Using AlsoConflicting: []
2024-11-13 14:04:19.788z info Cluster domain derived from agent-injector reverse lookup "agent-injector.network-system.svc.cluster.local."
2024-11-13 14:04:19.788z info Using cluster domain "cluster.local."
2024-11-13 14:04:19.788z info ExcludeSuffixes: [.com .io .net .org .ru]
2024-11-13 14:04:19.788z info IncludeSuffixes: []
2024-11-13 14:04:19.789z info cli-config : Started watcher for ConfigMap traffic-manager
2024-11-13 14:04:19.789z info agent-injector : Loading ConfigMaps from []
2024-11-13 14:04:19.790z info Prometheus : Prometheus metrics server not started
2024-11-13 14:04:19.891z info Scanning 1 nodes
2024-11-13 14:04:19.891z info Found 1 subnets
2024-11-13 14:04:19.891z info Deriving subnets from podCIDRs of nodes
2024-11-13 14:04:20.791z info agent-configs : Started watcher for ConfigMap telepresence-agents cluster wide
2024-11-13 14:04:20.792z info agent-configs : Started watcher for Services cluster wide
2024-11-13 14:10:29.775z error httpd/conn-127.0.0.1:8081 : LookupIP failed, trying LookupIP "tel2-recursion-check.kube-system." : session_id="10c4e558-b3c4-4cb2-8c1c-277173db30a8"
2024-11-13 14:11:15.761z error httpd/conn-127.0.0.1:8081 : LookupIP failed, trying LookupIP "tel2-recursion-check.kube-system." : session_id="ce387df-e56a-48ef-b8f5-71e5214984d2"
2024-11-13 14:12:42.075z error httpd/conn-127.0.0.1:8081 : LookupIP failed, trying LookupIP "tel2-recursion-check.kube-system." : session_id="846309c-04a2-4394-922d-06e886e2f162"
2024-11-13 14:13:01.802z info agent-configs : Successfully rolled out config-server.network-system
2024-11-13 14:13:02.776z info agent-injector : Injecting traffic-agent into pod config-server-86cb59d5b8-.network-system
2024-11-13 14:13:02.777z info agent-injector : Injecting 9 patches into pod config-server-86cb59d5b8-.network-system
2024-11-13 14:13:02.795z info agent-injector : Injecting traffic-agent into pod config-server-86cb59d5b8-.network-system
2024-11-13 14:13:02.796z info agent-injector : Pod config-server-86cb59d5b8-.network-system already has container traffic-agent and it isn't modified
2024-11-13 14:13:04.666z info httpd/conn-127.0.0.1:8081 : Warning Backoff Back-off restarting failed container tel-agent-init in pod config-server-86cb59d5b8-khknk.network-system(dbe09c56-122c-466d-bed7-e0212e9c9cf)
2024-11-13 14:13:03.830z info Traffic Agent Init v2.18.0
2024-11-13 14:13:03.833z error failed to clear chain TEL_INBOUND_TCP: running [/sbin/iptables -t nat -N TEL_INBOUND_TCP --wait]: exit status 4: Fatal: can't open lock file /run/xtables.lock: Permission denied
2024-11-13 14:13:03.833z error quit: failed to clear chain TEL_INBOUND_TCP: running [/sbin/iptables -t nat -N TEL_INBOUND_TCP --wait]: exit status 4: Fatal: can't open lock file /run/xtables.lock: Permission denied
: session_id="846309c-04a2-4394-922d-06e886e2f162"
2024-11-13 14:22:59.369z error httpd/conn-127.0.0.1:8081 : LookupIP failed, trying LookupIP "tel2-recursion-check.kube-system." : session_id="df205c26-2a63-49f2-a663-947a8d3161e3"
2024-11-13 14:24:21.707z error httpd/conn-127.0.0.1:8081 : LookupIP failed, trying LookupIP "tel2-recursion-check.kube-system." : session_id="1ab21c79-02d5-401e-9132-984b988286ba"
2024-11-13 14:26:43.483z info agent-configs : Successfully rolled out config-server.network-system
2024-11-13 14:26:44.046z info agent-injector : Injecting traffic-agent into pod config-server-7f87c494bb-.network-system
2024-11-13 14:26:44.047z info agent-injector : Injecting 9 patches into pod config-server-7f87c494bb-.network-system
2024-11-13 14:26:44.092z info agent-injector : Injecting traffic-agent into pod config-server-7f87c494bb-.network-system
2024-11-13 14:26:44.093z info agent-injector : Pod config-server-7f87c494bb-.network-system already has container traffic-agent and it isn't modified
2024-11-13 14:26:45.079z info httpd/conn-127.0.0.1:8081 : Warning Backoff Back-off restarting failed container tel-agent-init in pod config-server-7f87c494bb-2md2f.network-system(414827a8-47b8-4aa9-938c-8c488d31def)
2024-11-13 14:26:45.096z info httpd/conn-127.0.0.1:8081 : Log from failing pod "config-server-7f87c494bb-2md2f", container tel-agent-init
2024-11-13 14:26:45.120z info Traffic Agent Init v2.18.0
2024-11-13 14:26:45.123z error failed to clear chain TEL_INBOUND_TCP: running [/sbin/iptables -t nat -N TEL_INBOUND_TCP --wait]: exit status 4: Fatal: can't open lock file /run/xtables.lock: Permission denied
2024-11-13 14:26:45.123z error quit: failed to clear chain TEL_INBOUND_TCP: running [/sbin/iptables -t nat -N TEL_INBOUND_TCP --wait]: exit status 4: Fatal: can't open lock file /run/xtables.lock: Permission denied
: session_id="1ab21c79-02d5-401e-9132-984b988286ba"
2024-11-13 14:31:08.528z info agent-configs : Successfully rolled out config-server.network-system
2024-11-13 14:31:01.1789z info agent-injector : Injecting traffic-agent into pod config-server-5cf9686c9-.network-system
2024-11-13 14:31:01.1791z info agent-injector : Injecting 9 patches into pod config-server-5cf9686c9-.network-system
2024-11-13 14:31:01.4971z info agent-injector : Injecting traffic-agent into pod config-server-5cf9686c9-.network-system
2024-11-13 14:31:01.4982z info agent-injector : Pod config-server-5cf9686c9-.network-system already has container traffic-agent and it isn't modified
2024-11-13 14:31:04.1230z info httpd/conn-127.0.0.1:8081 : Warning Backoff Back-off restarting failed container tel-agent-init in pod config-server-5cf9686c9-mrbxv.network-system(fa36791e-a036-4608-a746-bd1ec4d6eb)
2024-11-13 14:31:04.1303z info httpd/conn-127.0.0.1:8081 : Log from failing pod "config-server-5cf9686c9-mrbxv", container tel-agent-init
2024-11-13 14:31:03.2832z info Traffic Agent Init v2.18.0
2024-11-13 14:31:03.2868z error failed to clear chain TEL_INBOUND_TCP: running [/sbin/iptables -t nat -N TEL_INBOUND_TCP --wait]: exit status 4: Fatal: can't open lock file /run/xtables.lock: Permission denied
2024-11-13 14:31:03.2868z error quit: failed to clear chain TEL_INBOUND_TCP: running [/sbin/iptables -t nat -N TEL_INBOUND_TCP --wait]: exit status 4: Fatal: can't open lock file /run/xtables.lock: Permission denied
: session_id="1ab21c79-02d5-401e-9132-984b988286ba"
patrick@TITAN-BOOK:~/kind/images/base$

```

Figure 34 Logs from Traffic Manager Pod

Error:

quit: failed to clear chain TEL_PREROUTING_TCP: running [/sbin/iptables -t nat -N TEL_PREROUTING_TCP --wait]: exit status 4: iptables v1.8.10 (nf_tables): Could not fetch rule set generation id: Permission denied (you must be root)

And here is to mention that the K8s Security Context enforce permissions were already set to "privileged."

Solution

A few days later, the authors received a response from Thomas Hallgren, a developer at Telepresence, stating that traffic could also be forwarded without needing an init-container (which produced the error). This can be done in the deployment specification in the colocated.yaml file. The ports of the services do also have the parameter targetPort, a description of the port. Otherwise, if such a named targetPort can't be found, a container will be injected into the namespace. From our view, this isn't practical because the tools should be separated, and a configuration shouldn't be somewhere you wouldn't expect it to be. In our view, this looks more like a workaround, which could have been solved better.

Nevertheless, the named port was tried out, and with the allow conflicting routes command, it did finally work out! Now, Telepresence can be used.

```
1091 spec:
1092   template:
1104     containers:
1105     - args:
1160       - args:
1161         - --config=/config/data-server.yaml
1162         command:
1163         - /app/data-server
1164         image: ghcr.io/sdcio/data-server:v0.0.48
1165         imagePullPolicy: Always
1166         name: data-server
1167         ports:
1168         - containerPort: 56000
1169         name: data-service
```

Figure 35 Adding containerPort name to data-service

```
1508 apiVersion: v1
1509 kind: Service
1510 metadata:
1511   labels:
1512     sdcio.dev/config-server: "true"
1513   name: data-server
1514   namespace: network-system
1515 spec:
1516   ports:
1517   - name: data-service
1518     port: 56000
1519     protocol: TCP
1520     targetPort: data-service
1521   selector:
1522     sdcio.dev/data-server: "true"
```

Figure 36 Adding "targetPort: data-service" to data-server

As a first step of the setup to debug SDC and configure Telepresence, the config-server could now be started. However, shortly after, the application crashed while creating a Data-Server instance. After some debugging, it became apparent that the config-server wants to connect to the config-server via `localhost:5774`. Assuming the config-server and the data-server could be deployed in separate pods, the authors didn't expect this. But since they were together in one pod, cloning the data-server and starting it locally alongside the config-server enabled running it without errors! After many hours of debugging the debugging environment, the authors are now able to debug SDC. After solving those issues, the authors created two pull requests to address the benefit of the named targetPort (see 8.1).

6.8 Arista new Release

Situation at Hand

At that moment, the Arista schemas still couldn't be loaded. At the end of October, the AutoCon2 was held in Denver. Our advisor, Urs Baumann, went to the convention, met one of the SDC developers, and spoke with one of them - Hans Thienpondt. Hans promised that after they were back from the convention, they would devote their time to looking after the known Arista problems.

Tried Remedies

One or two weeks later, the authors received a message that the developers had addressed the issue and were planning to release a new version of SDC, which should be out before Christmas. Hans asked for some configuration intent samples and wanted to know which Arista cEOS version the authors used to verify the progress.

Later, the authors were notified that a new release could be tested. While Hans tested it with Arista 4.31.1F, the newest version of cEOS, 4.33.0F was used by the team.

Hans also provided the files, which he used for testing.

Schema file:

```
apiVersion: inv.sdcio.dev/v1alpha1
kind: Schema
metadata:
  name: eos.arista.4.31.1f
  namespace: default
spec:
  provider: eos.arista.sdcio.dev
  version: 4.31.1F
  repositories:
  - repoURL: https://github.com/aristanetworks/yang
    kind: branch
    ref: master
    dirs:
    - src: EOS-4.31.1F
      dst: .
    schema:
      models:
      - release/openconfig/models
      - experimental/eos/models
      - openconfig/public
      includes:
      - ietf
      excludes:
      - arista-acl-notsupported-deviations.yang
      - arista-bfd-notsupported-deviations.yang
      - arista-defined-sets-notsupported-deviations.yang
      - arista-interfaces-notsupported-deviations.yang
      - arista-keychain-notsupported-deviations.yang
      - arista-lacp-notsupported-deviations.yang
      - arista-lldp-notsupported-deviations.yang
      - arista-macsec-notsupported-deviations.yang
      - arista-network-instance-notsupported-deviations.yang
      - arista-platform-notsupported-deviations.yang
      - arista-qos-notsupported-deviations.yang
      - arista-sampling-notsupported-deviations.yang
      - arista-spanning-tree-notsupported-deviations.yang
      - arista-system-notsupported-deviations.yang
      - arista-telemetry-notsupported-deviations.yang
      - arista-terminal-device-notsupported-deviations.yang
  - repoURL: https://github.com/YangModels/yang
    kind: branch
    ref: main
    dirs:
    - src: standard/ietf/RFC
      dst: ietf
    schema:
      models:
      - ietf/ietf-inet-types.yang
      - ietf/ietf-yang-types.yang
  - repoURL: https://github.com/sdcio/arista-yang-patch
    kind: branch
    ref: main
    dirs:
    - src: .
      dst: sdcio-deviations
    schema:
      models:
      - sdcio-deviations/EOS-4.31.1F/release/openconfig/models
```

From the Schema YAML file above, it can be indicated that two repositories are used, the official OpenConfig schema and one for patching the dependencies, which they hosted by themselves. Hans wrote a YANG with the patches himself, which are needed because OpenConfig didn't update the new capabilities of the devices. They have changed because of new firmware updates, and therefore, the configuration structures are different. The deviations invalidate paths that OpenConfig still has active references to. Because of this, they've applied patches to their deviations. As deployment of other

configurations is necessary for the use case, it could be possible that more similar issues will occur. Hans sent the example configurations, which he used.

Loopback config:

```
apiVersion: config.sdcio.dev/v1alpha1
kind: Config
metadata:
  name: arista_loopback
  namespace: default
  labels:
    config.sdcio.dev/targetName: eos1
    config.sdcio.dev/targetNamespace: default
spec:
  priority: 10
  config:
    - path: /
      value:
        interfaces:
          - interface:
              name: Loopback0
              config:
                description: "my custom created loopback"
                name: Loopback0
                type: ianaift:softwareLoopback
```

Hostname config:

```
apiVersion: config.sdcio.dev/v1alpha1
kind: Config
metadata:
  name: intent1-arista
  namespace: default
  labels:
    config.sdcio.dev/targetName: eos1
    config.sdcio.dev/targetNamespace: default
spec:
  priority: 10
  config:
    - path: /system
      value:
        config:
          hostname: arista-01
```

Indeed, it worked out with the two example configurations and cEOS version 4.31.1F! But testing it with 4.33.0F, only changing the loopback description worked. Changing the hostname wasn't successful.

The error of the hostname change on 4.33.0F:

```

ins@ubuntu-L:~/test_arista_v051$ k get sdc
NAME          READY   REASON   TARGET   SCHEMA
arista_loopback True    Ready    default/eos2 eos.arista.sdcio.dev/4.33.0F
intent1-arista False   Failed    default/eos2

NAME
eos2
eos1

NAME   DEVIATIONS
eos1   917
eos2   0

NAME                                     READY
discoveryrule.inv.sdcio.dev/dr-arista-gnmi-4.31.1f True
discoveryrule.inv.sdcio.dev/dr-arista-gnmi-prefix-4.33.0f True

NAME          READY   PROVIDER   VERSION   URL                                     REF
schema.inv.sdcio.dev/eos.arista.4.31.1f True    eos.arista.sdcio.dev 4.31.1F   https://github.com/aristanetworks/yang master
schema.inv.sdcio.dev/eos.arista.4.33.0f True    eos.arista.sdcio.dev 4.33.0F   https://github.com/aristanetworks/yang master

NAME          PROTOCOL   PORT   ENCODING   INSECURE   SKIPVERIFY
targetconnectionprofile.inv.sdcio.dev/test-arista-gnmi gnmi    6030   JSON_IETF true        true

NAME          READY   REASON   PROVIDER   ADDRESS   PLATFORM   SERIALNUMBER   MACADDRESS
target.inv.sdcio.dev/eos1 True     True     eos.arista.sdcio.dev 172.21.0.200
target.inv.sdcio.dev/eos2 True     True     eos.arista.sdcio.dev 172.21.0.201

NAME          PROTOCOL   PORT   ENCODING   MODE   INTERVAL
targetsyncprofile.inv.sdcio.dev/test-arista-gnmi-get gnmi    57400   JSON_IETF get      30s
ins@ubuntu-L:~/test_arista_v051$ k describe configs.config.sdcio.dev
arista_loopback intent1-arista
ins@ubuntu-L:~/test_arista_v051$ k describe configs.config.sdcio.dev intent1-arista
Name:          intent1-arista
Namespace:     default
Labels:        config.sdcio.dev/targetName=eos2
               config.sdcio.dev/targetNamespace=default
Annotations:   <none>
API Version:   config.sdcio.dev/v1alpha1
Kind:          Config
Metadata:
  CreationTimestamp: 2024-12-11T14:45:24Z
  Finalizers:
    config.config.sdcio.dev/finalizer
  Generation:      2
  Resource Version: 2
  UID:             29c447b6-32db-4465-911d-967e7c1c51c6
Spec:
  Config:
    Path: /system
    Value:
      Config:
        Hostname: arista-01
  Lifecycle:
    Priority: 10
Status:
  Conditions:
    Last Transition Time: 2024-12-11T14:45:24Z
    Message:             set intent failed err rpc error: code = Unknown desc = unknown object "config" under container "system"
    Reason:              Failed
    Status:              False
    Type:                Ready
Events:
  Type    Reason   Age          From          Message
  ----    -
Warning  Config  63s (x2 over 63s) ConfigController set intent failed err rpc error: code = Unknown desc = unknown object "config" under container "system"
ins@ubuntu-L:~/test_arista_v051$

```

Figure 37 Arista 4.33.0F Error Config change Hostname

While testing version 4.31.1F, the reconciliation feature was tested as well. The hostname was changed to another value; it needed one or two minutes, and it was already changed back to the defined hostname. Reconciling's magic works now with this example.

However, further tests with the configuration revealed that YANG and the OpenConfig schemas aren't reliable and easy to work with. It is complex to find out in YANG which objects of the tree are required to build the desired configuration. In addition, it needs a lot of text for a single change.

Explanation

Shortly before Christmas, Hans gave an update to the self-constructed repository with the OpenConfig patches: If all of the configuration parts should be used, they have to test it one by one and possibly add a patch if the path isn't valid. It is a lot of work to have a running network automation system, and from our point of view, the practice of vendors making extensive deviations will undoubtedly lead to hacked-together solutions such as the one provided by Hans. This makes the authors question the reliability and efficiency of YANG in this tool and other tools for daily usage.

7. Applicability of the Tool Stack for Network Automation

7.1 Operational Lifecycle of an EVPN-VXLAN Spine-Leaf Fabric (L2)

In this chapter, the configuration deployments/changes that need to be made during the lifetime of such a network infrastructure are analyzed.

7.1.1 Initial Deployment

The initial deployment is the most straight forward one: All configuration values can be collected and written to the devices. In theory there could be one single configuration intent per device.

7.1.1.1 Underlay

Configuration	All Spines	All Leaves
Port configuration (routed mode, static IP address)	X	X
OSPF (enable feature, router instance, interface configuration)	X	X

Figure 38 Configuration Parts Underlay

7.1.1.2 Overlay

Configuration	All Spines	All Leaves
BGP (enable feature, router instance, neighbors)	X	X
Multicast (enable feature, rendezvous point, interface configuration)	X	X
EVPN (enable feature, nve interface, evpn instance)		X
VLANs (VN segment mapping, interface configuration)		X

Figure 39 Configuration Parts Overlay

7.1.2 Add/remove L2 Endpoint

This is expected to be the most common use case: A new endpoint, e.g. server or client is connected to the fabric. The access-port needs to be configured for the necessary VLAN.

7.1.2.1 Overlay

Configuration	Leaf of endpoint
VLAN (interface configuration)	X

Figure 40 Configuration Parts VLAN

7.1.3 Add/remove L2 Service

A new L2 service, i.e. a VLAN is rolled out in the fabric. Because of VXLAN, the spine switches need no configuration update for this change.

7.1.3.1 Overlay

Configuration	All Leaves
EVPN (instance VNIs, nve members)	X
VLANs (VN segment mapping, VLAN interface)	X

Figure 41 Configuration Parts VLAN mapping

7.1.4 Add/remove Leaf

A new leaf is integrated into the fabric for more access ports.

7.1.4.1 Underlay

Configuration	All Spines	New leaf
Port to new leaf: Port configuration (routed mode, static IP address, OSPF)	X	
Port configuration (routed mode, static IP address)		X
OSPF (enable feature, router instance, interface configuration)		X

Figure 42 Configuration Parts Underlay

7.1.4.2 Overlay

Configuration	All Spines	New leaf
BGP (new neighbor)	X	
BGP (enable feature, router instance, neighbors)		X
Multicast (enable feature, rendezvous point, interface configuration)		X
EVPN (enable feature, nve interface, evpn instance)		X
VLANs (VN segment mapping, interface configuration)		X

Figure 43 Configuration Parts Overlay

7.1.5 Add/Remove a Spine

An additional spine provides more bandwidth and more redundancy.

7.1.5.1 Underlay

Configuration	New Spine	All Leaves
Port configuration (routed mode, static IP address)	X	
OSPF (enable feature, router instance, interface configuration)	X	
Port to new spine: Port configuration (routed mode, static IP address, OSPF)		X

Figure 44 Configuration Parts Underlay

7.1.5.2 Overlay

Configuration	New Spine	All Leaves
BGP (enable feature, router instance, neighbors)	X	
Multicast (enable feature, rendezvous point, interface configuration)	X	
BGP (new neighbor)		X
Port to new spine: Port configuration (PIM)		X

Figure 45 Configuration Parts Overlay

7.2 Proposed partitioning of the Configuration into Configuration Intents

In this chapter, the partitioning of the configuration in parts to make the changes for the above tasks possible is analyzed.

Coloring:

- Green: Configuration is quasi-static. It doesn't change during the lifecycle.
- Purple: Configuration is dependent on the current number of spines/leaves.
- Orange: Configuration changes regularly in operation. (L2 services / VLANs, Endpoints)

#	Config Name	Targets	Config Content
1	Loopback0 (BGP, OSPF)	All switches	<pre>interface loopback0 ip address <lo0-ip>/32 ip router ospf 1 area 0.0.0.0 ip pim sparse-mode</pre>
2	Loopback1 (EVPN)	All leaves	<pre>interface loopback1 ip address <lo1-ip>/32 ip router ospf 1 area 0.0.0.0 ip pim sparse-mode</pre>
3	Loopback1 (RP Anycast)	All spines	<pre>interface loopback1 ip address 100.1.1.1/32 ip router ospf 1 area 0.0.0.0 ip pim sparse-mode</pre>
4	OSPF base	All switches	<pre>feature ospf router ospf 1</pre>
5	Interface to spine	All leaves (one per spine)	<pre>interface Ethernet<number> no switchport medium p2p ip unnumbered loopback0 ip ospf network point-to-point ip router ospf 1 area 0.0.0.0 ip pim sparse-mode no shutdown</pre>
6	Interface to leaf	All spines (one per leaf)	<pre>interface Ethernet<number> no switchport medium p2p ip unnumbered loopback0 ip ospf network point-to-point ip router ospf 1 area 0.0.0.0 ip pim sparse-mode no shutdown</pre>
7	BGP base	All switches	<pre>feature bgp router bgp 65000 address-family l2vpn evpn retain route-target all</pre>

8	BGP neighbor spine	All leaves (one per spine)	router bgp 65000 neighbor <u><spine-lo0-ip></u> remote-as 65000 address-family l2vpn evpn send-community send-community extended
9	BGP neighbor leaf	All spines (one per leaf)	router bgp 65000 neighbor <u><leaf-lo0-ip></u> remote-as 65000 address-family l2vpn evpn send-community send-community extended route-reflector-client
10	PIM base	All switches	feature pim ip pim rp-address 100.1.1.1 group-list 224.0.0.0/4 ip pim ssm range 232.0.0.0/8
11	PIM anycast	All spines (one per spine)	ip pim anycast-rp 100.1.1.1 <u><spine-lo0-ip></u>
12	EVPN base	All leaves	nv overlay evpn feature vn-segment-vlan-based feature nv overlay interface nve1 no shutdown host-reachability protocol bgp source-interface loopback1
13	VLAN base	All leaves	feature interface-vlan
14	VLAN ids	All leaves	vlan <u><vlan-id0></u> , <u><vlan-id1></u> , <u><vlan-id2></u> ,...
15	VLAN service	All leaves (one per vlan)	vlan <u><vlan-id></u> vn-segment 300 <u><vlan-id></u> interface nve1 member vni 300 <u><vlan-id></u> mcast-group 239.0.0. <u><vlan-id></u> evpn vni 300 <u><vlan-id></u> l2 rd auto route-target import 1: <u><vlan-id></u> route-target export 1: <u><vlan-id></u>
16	Endpoint	Leaf of endpoint	interface Ethernet <u><number></u> switchport access vlan <u><vlan-id></u> spanning-tree port type edge

Figure 46 Partitioning of the Configuration Parts

7.3 Dependencies of Configuration Parts

The following graph shows how the configuration parts could depend on each other. For example, the OSPF feature (#4) might need to be enabled before OSPF settings can be applied to interfaces (#1).

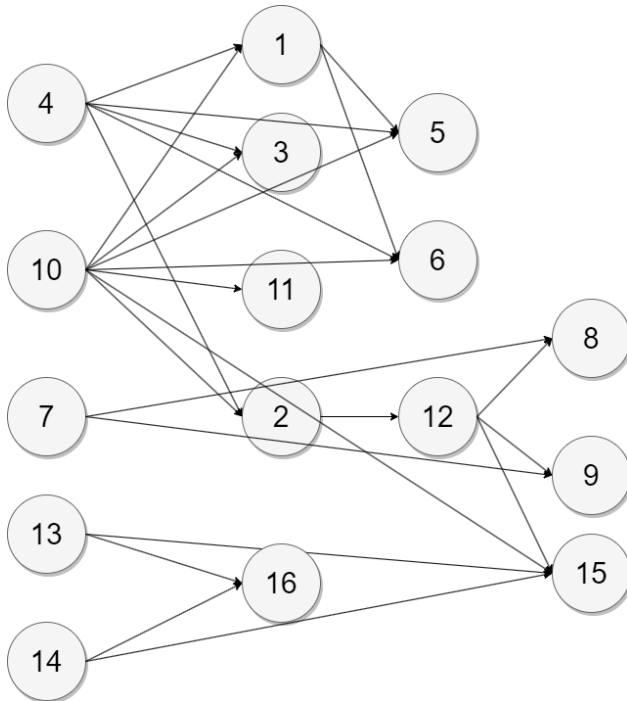


Figure 47 Dependencies of the different Configuration Parts

The authors suspect however, that by using YANG based declarative configuration, the device can accept the dependent configuration part before the needed one and it will be active as soon as the latter is sent as well. As it is suspected that it even works in the CLI, the authors test this by configuring one device in the correct order and one in the reverse order.

Configuration on dev1 :

```
dev1#conf t
ip routing

router ospf 1
  router-id 10.0.0.10
  interface unnumbered hello mask tx 0.0.0.0
  max-lsa 12000
  timers spf delay initial 100 200 500
  timers lsa rx min interval 100
  timers lsa tx delay initial 100 200 500

interface Ethernet1
  no switchport
  ip address 10.1.0.9/30
  ip ospf network point-to-point
  ip ospf area 0.0.0.0
```

Configuration on dev2:

```
dev2#conf t
interface Ethernet1
  no switchport
  ip address 10.1.0.10/30
  ip ospf network point-to-point
  ip ospf area 0.0.0.0

router ospf 1
  router-id 10.0.0.20
  interface unnumbered hello mask tx 0.0.0.0
  max-lsa 12000
  timers spf delay initial 100 200 500
  timers lsa rx min interval 100
  timers lsa tx delay initial 100 200 500

ip routing
```

While the CLI warns that IP routing is not yet enabled, the configuration is still activated as soon as all needed parts are applied:

```
dev2(config-if-Et1)#router ospf 1
! IP routing not enabled
```

Figure 48 Warning IP Routing

```
dev2#show ip ospf neighbor
Neighbor ID      Instance VRF      Pri State      Dead Time  Address      Interface
10.0.0.10        1      default 0  FULL      00:00:38  10.1.0.9    Ethernet1
```

Figure 49 OSPF successfully activated.

Therefore, it is concluded that if it works on CLI, it will surely also work via YANG and gNMI.

7.4 Configuration Parts composing the Tasks

With the necessary tasks during the life cycle listed and the configuration divided into parts, it can be determined which part is needed on which device for each task.

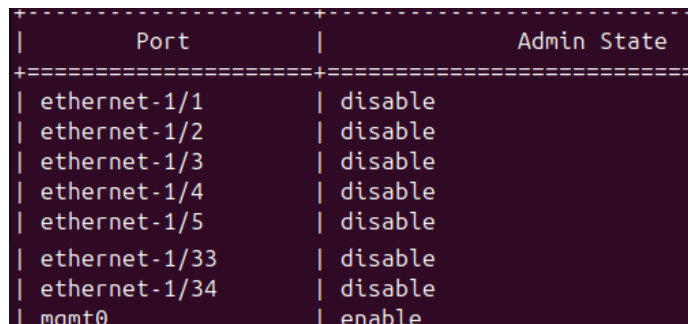
Task	Add(+) or Remove(-) parts
Initial deployment	All spines: +1,3,4,6,7,9,10,11 All leaves: +2,4,5,7,8,10,12,13
Add spine	New spine: +1,3,4,6,7,9,10 All spines: +11 All leaves: +5,8
Remove spine	All leaves: -5,8
Add leaf	New leaf: +1,2,4,5,7,8,10,11,12,13 All spines: +6,9
Remove leaf	All spines: -6,9
Add L2 service	All leaves: +14,15
Remove L2 service	All leaves: -14,15
Add endpoint	Endpoint leaf: +16
Remove endpoint	Endpoint leaf: -16

Figure 50 Configuration Parts composing the Tasks

7.4.1 Removing Configuration via gNMI

In the above partitioning of the needed configuration, it is assumed that a configuration snippet can be deleted from the device in the same manner as it was applied. This assumption is tested manually by using the gNMI client, `gnmic`[31].

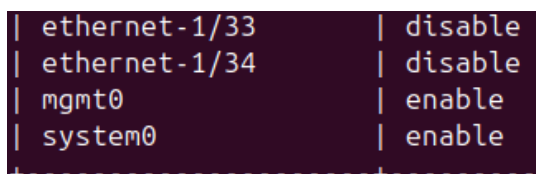
First, an interface called "system0" was added and enabled with the following command:



Port	Admin State
ethernet-1/1	disable
ethernet-1/2	disable
ethernet-1/3	disable
ethernet-1/4	disable
ethernet-1/5	disable
ethernet-1/33	disable
ethernet-1/34	disable
mgmt0	enable

Figure 51 Interfaces on Device before adding system0

```
nmic -a 172.21.0.200:57400 -u admin -p NokiaSrl1! --skip-verify --gzip set \
--update-path '/interface[name=system0]/admin-state' \
--update-value 'enable'
```



ethernet-1/33	disable
ethernet-1/34	disable
mgmt0	enable
system0	enable

Figure 52 Interfaces after adding the Configuration

The "--delete" flag of gnmic is then used to remove it again:

```
gnmic -a 172.21.0.200:57400 -u admin -p NokiaSrl1! --skip-verify --gzip set \
--delete '/interface[name=system0]'
```

```
| ethernet-1/33 | disable
| ethernet-1/34 | disable
| mgmt0         | enable
+-----+-----+
```

Figure 53 The Interface is gone again.

With this test, the authors could confirm their assumption. Nokia SR Linux was used for the test as it was available at this moment. The functionality is expected to be the same for other vendors, which supporting gnmic.

7.5 Ecosystem around SDC (Requirements / Interfaces)

Now that it is known which configuration snippet is needed at which point and for which device, a closer look is needed at how these configuration intents are generated (i.e., rendered from templates), where the source data comes from, and how they make their way to SDC.

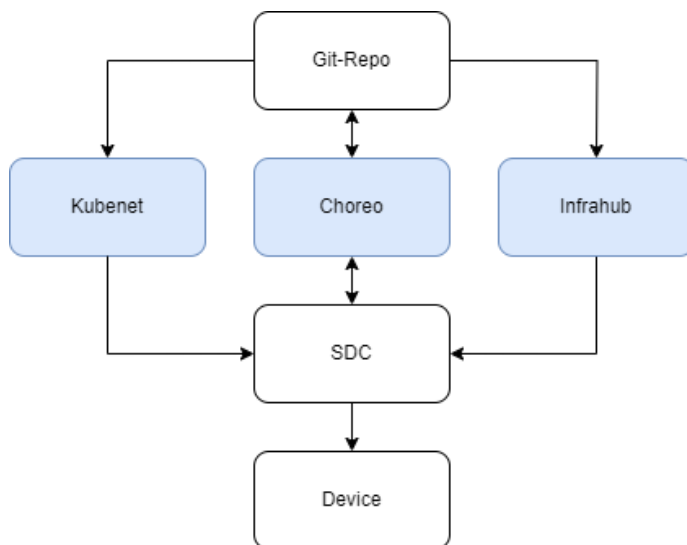


Figure 54 Configuration Sources for SDC

7.5.1 Configuration Generation

Most configuration snippets such as loopback interfaces contain parameterized data i.e. IP addresses. This means that the configuration parts can't be static but need to be generated from dynamic values. For this purpose, different tools can be used. The SDC development team proposes the Kubenet tools and its successor, Choreo, for this task.

7.5.1.1 SDC's original partner: Kubenet

Input:

- Network Topology
- Device Model
- IP Index (Prefix)
- Network Configuration (Interface addressing pool, routing protocols, overlays, encapsulation, ...)
- Overlay Networks (L2, L3)
- Kuid app (vendor-specific app that contains all definitions and templates of the device)

Output:

- Configuration intents are K8s CRs that are picked up by SDC and transacted to the devices.

With the help of a component called pkgserver, a Git repository can be used as the source for the input artifacts. The component also allows the export of the device configuration to a Git repository for checking and validation by a network engineer or performing automated tests instead of transacting it straight to the device. Thus, together with Git and SDC, Kubenet offers the whole ecosystem for device configuration.

Kubenet is however not without disadvantages. The most apparent of which is the kuid app. This app is the source of all logic and templates behind the whole configuration. This means that it's a big monolithic artifact. And one must be written for every vendor.

7.5.1.2 Kubenet's Successor: Choreo

Choreo can basically use the same CRs as Kubenet. The main difference with respect to configuration generation is that there are more capabilities due to its decoupling of K8s. Examples are built-in version control, (better) dry-run capabilities, can do "run to completion" i.e. run once and even more extendibility with Python (Starlark). Additional Choreo provides the --sdc flag, which adds the logic from SDC, it will validate the generated config against the schema.

7.5.1.3 The Alternative: Infracore

Infracore uses so-called transformations to create objects like K8s manifests from its data. It can use Jinja2 templates or Python functions to do so. The data is input using a GraphQL query. This would mean that the business logic for creating the configuration would have to be written in Python since Jinja2 likely doesn't provide enough programmatic flow control for the task.

7.5.2 Configuration Source

This chapter describes the source for the configuration parts. This includes the templates as well as the parameterized data, like IP addresses.

7.5.2.1 Static Configuration and Templates

All the above-mentioned configuration generators can use a git repository as their source of data. In the philosophy of Infrastructure as code (IaC), the version-controlled storage of the configuration is of vital importance. Since Git provides the de facto standard for version control, it is the ideal source for any static configuration and templates.

7.5.2.2 Parameterized Data

The parameterized data encompasses all kinds of parameters unique to a specific device or set of devices, like IP address pools, IP addresses, AS numbers, VLAN IDs, and VNIs. In the use case of an EVPN-VXLAN spine-leaf fabric, loopback IPs and VLAN IDs are needed at a minimum.

If Kubenet is used, IP address pools and VLAN IDs can be set directly in the input files. IP addresses will be reserved automatically out of the specified address pool. That means if the use case is to be implemented with Kubenet, the Git repository can also be used for this parameterized data (while IP reservations are made dynamically within K8s). Using an external IPAM tool for address management, such as Netbox, is surely possible by modifying the corresponding CRDs. However, this is outside of the scope of this project.

Choreo which understands itself as successor to Kubenet, uses Git by default as a version-controlled storage. As it is compatible with the K8s resource model (KRM) it can use the same CRDs as Kubenet.

Infrahub uses its NEO4J graph database to store parameterized data as properties of objects.

7.5.3 Configuration Deployment (Kubenet, Choreo, Infrahub)

Using Kubenet, the deployment is straightforward and direct: Normally, Kubenet and SDC run in the same cluster. As soon as Kubenet creates the configuration intents as CRs, they are picked up by SDC and “reconciled” to the devices automatically. There is the option to instead send the configuration to a Git repository and then use whichever deployment tool is preferred.

Choreo creates the configuration intents as files in the folder structure. They can be applied directly with the -s (SDC) switch or picked up from the file system for further processing.

Infrahub can be integrated with automation tools such as Nornir or Ansible to deploy the configuration to the devices[32].

7.5.4 Rollout Strategies

Making changes to running networks always poses the risk of errors. Reviewing the configuration before it is transacted to the devices as described in 0 can help reduce the risks on deployment but never completely prevent it. In application operations, there exists a strategy called Canary Deployment[33] for this exact reason.

```
apiVersion: config.sdcio.dev/v1alpha1
kind: ConfigSet
metadata:
  name: intent1
  namespace: default
spec:
  target:
    targetSelector:
      matchLabels:
        sdcio.dev/region: us-east
  priority: 10
  config:
  - path: /
    value:
      interface:
      - name: ethernet-1/1
        admin-state: "enable"
        description: "intent1"
```

Figure 55 Configuration Intent with Label targetSelector; taken from [45]

Since the CRD for configurations in SDC selects the target based on K8s labels, the mechanism for staggered updates would ideally reside inside SDC. This is, however, not the case at the moment. The authors will make a feature recommendation for this functionality to the maintainers of SDC. Up until this feature will be possibly implemented, another way is needed.

A well-known tool implementing Canary Deployment in K8s is Argo Rollouts[34]. Unfortunately, Argo Rollouts is explicitly targeted at K8s workloads by managing Deployment resources[35]. Our resources in question are, however, configs, which are CRs, meaning that they can't be managed with Argo Rollouts.

Another Deployment tool is Flux CD[36], which, in combination with Flagger, can implement various deployment strategies like Canary Deployment, A/B Testing, etc[37]. The documentation also only provides ways to deploy Deployment resources in a canary manner.

What else can be done? Two intents for each change could be generated, one that matches only a single device via name (i.e., only one of the leaves) and the other that matches the purpose (i.e., all leaves). Only the first intent would be applied, and the network would be monitored for problems. If everything works well, the second one will be applied, which will transact the configuration to all devices. In the current state, there is the need to have the configuration generator push the intents into a Git repository instead of the device directly, manually create the "canary" intent, and apply both either by hand or through the help of a workflow engine of choice. This engine must support deploying non-standard Kubernetes resources.

7.5.5 Operational Requirements

Up until now, the authors analyzed SDC's features, followed examples, and tried to create a PoC for the specified use cases. In this chapter, the assumption is made that an organization wants to use SDC in production. The requirements for productive use are, of course, more extensive than in testing environments.

Integration: "No system is an island." The growing complexity of networked systems often leads to several systems working together in network operation. For effective and efficient network automation, they must interact with each other.

Stability: In a productive environment, minimal downtime is always a crucial requirement. The tools used thus need to provide a required level of stability in operation.

Knowledge management: The engineers operating the network need the knowledge to extend functionality, monitor the operational status, and, if needed, perform corrective maintenance. If the organization cannot cover the requirement themselves, support is usually bought from an external supplier.

7.6 Fulfillment of Requirements

Use Case	Responsible Component(s)	Degree of fulfillment	Comment
UC 1 Manage Underlay-Network UC 3 Manage Overlay-Network	Kubenet / Choreo	Showcase only Nokia only	Kubenet creates the configuration with kuidapps, along with various CRs of the types Topology, Network, NetworkDevice, and NetworkConfig, according to the CRDs. The network designs and settings available in the CRs published by developers are currently limited to their showcase. The result of the generation is a set of new CRs representing the abstract configuration, which is then transformed into a device-specific configuration (YANG) by a vendor-specific kuid app using templates. Currently, the only vendor-specific kuid app available is the one for Nokia SR Linux. Both CRs and kuid apps are extendable by experienced users.
UC 2 Predefined Configuration	SDC	With compatible YANG models	Connection to the devices and discovery work well. Loading of YANG models and subsequent validation of configuration intents depends highly on the structure of the models, as success depends on the order in which the models are loaded by SDC. Patching/Reworking of the vendor models is needed.
UC 4 Golden State	SDC	Partly	While SDC offers no versioning, it validates configuration against the schema before deployment, thus reducing the risk of failures. Reverting to a previous state would mean deleting the configuration intent from the K8s cluster either manually or by some external automated means.
UC 5 Vendor Independence	Kubenet / Choreo/ SDC	Yes, by design	All three tools were designed with multivendor capabilities. Their capabilities depend on the resources provided by the user, such as the kuid app for transformation abstract -> vendor specific and YANG models.

UC 6 Automated Resource Deployment	Kubenet / SDC	Partly	Resources created by Kubenet are automatically picked up by SDC, but applying the source manifests for Kubenet, i.e., from Git, has to be done manually or by an external tool.
UC 7 Manage Multitenancy	Kubenet / Choreo	Nokia only	The scope of available configuration features depends on the CRs and kuid app provided by the user. The example CRDs and the Nokia SR Linux kuid app support multiple VRFs for Nokia SR Linux. Both CRs and kuid apps are extendable by experienced users.
NFR 1 Cloud-native	Kubenet / Choreo/ SDC	Mostly	Kubenet and SDC run in K8s and heavily rely on its API functionality. They both provide Git repository integration and are, so far, only intended to run collocated. Choreo breaks with the need for K8s and instead provides its own API server. Since it is in early development, it is unclear how far the integration will go in the future.
NFR 2 Provisioning	SDC	Assumed	Unfortunately, only limited testing of the actual deployment could be done, but provided the necessary CRs and schemas are loaded correctly(!), deployment seems to be rather quick.
NFR 3 Usability/Learnability	(see comment)	Poor	The authors see the Kubenet/Choreo – SDC in its current state more as a rough framework showcasing ideas and paradigms than a mature tool stack. It is constructed on good principles and provides extensibility (especially Choreo) but lacks functionality other than the showcase with Nokia so far. Developing the functionality for a specific vendor is left to the user. Also, troubleshooting is not very user-friendly.

Figure 56 Fulfillment of Requirements

7.7 Verdict on SDCs readiness for production

Principles

SDC is built to be vendor-agnostic. Devices from multiple vendors can be managed by adding specific connection profiles and, if needed, the appropriate YANG schemas for each vendor. This follows best practices to avoid vendor lock-in and provides a future-proof solution. Its declarative nature is an enabler for modern network automation and adheres to the current cloud-native philosophy. The idea of using YANG schemas to validate data has been around for some time. See for example[38]. However, SDC is the first tool the authors have come across that bundles it with K8s-based network device configuration and declares it as a high priority (schema-driven). Validating the configuration data before deploying it to the devices is seen as an important aspect, and SDC's intentions for this are good.

Need and Benefit

One of SDC's core functions is that it uses K8s' continuous reconciliation feature. This is a valid operation principle for cloud-native application operations. It integrates with monitoring and scaling. Failed pods are redeployed automatically, and the horizontal scaling is adjusted to the current load without manual intervention. However, it begs the question of the necessity and usefulness of continuous reconciliation in network configuration.

First, in network automation there are no pods (application workloads) that fail. Failures in network devices most often happen in hardware that necessitates replacement. Failures in network devices operating systems typically don't require a form of redeployment but rather a reboot, or they result in a hangup that results in the device being unreachable for management purposes.

Second, networks typically don't possess any "live" scaling capabilities. The installation and connection of physical hardware are needed. In the case of virtual network devices, scaling computing power might be the exception here, but this topic is out of the scope of this project. And even with virtual devices, at some point they still need physical connections that require manual intervention to "scale".

Third, one would not be completely astray to assume even with IaC principles, network configuration experiences far fewer changes than software applications with CI/CD applied.

Lastly, the authors strongly believe that a defined state and a terminating execution are far more important in the domain of network management than automatic remediation of potential configuration drift (continuous reconciliation).

Stability

The first commits to the SDC repositories date back about two years. It is still in heavy development, with the latest changes made only one day ago in release v0.0.44 of the config-server (13.12.2024). This fact, along with our own experiences over the last weeks of trying to deploy basic configurations and some problems in even playing through the examples, brings us to the conclusion that in its current

state, SDC does not provide stable operation with a reasonable degree of error-freeness for vendors other than Nokia without significant effort in tweaking official configuration models.

Some of these functionality problems are, however, not caused by SDC itself but are instead rooted in the complexity of YANG and the practice of device vendors to make an extensive number of deviations from them.

Integration

SDC runs in K8s and is thus well integrated with it. It makes use of K8s' components and principles where possible. For communicating with targets, the well-known protocols Netconf and gNMI are supported, which is good for compatibility.

On the data input side, SDC follows IaC principles by using Git repositories for device YANG models. The control of SDC's Config-Server is done via the K8s API. There is also a REST API for which no documentation is available at this moment.

Interfacing with an external system for IP address management or other dynamic values is not needed for SDC because the configuration creation and template rendering to K8s manifests must be done by an external tool. The manifests are then applied using the K8s API, i.e., kubectl.

The authors would have liked to see an integration option with an external tool that handles canary deployment or rolling updates because SDC doesn't support the former, and the actual procedure of the latter is unknown to us. Known rollout tools like Argo Rollouts that support, for example, canary deployment don't work in this case because they were designed for K8s' original use case, where work units in the form of Pods are rolled out, which isn't the case with SDC.

Knowledge

With the documentation page^[11] SDC seems to be reasonably well documented. By now, the documentation also includes a troubleshooting section and even a small guide for development. The troubles for us began when errors appeared that went beyond what is documented. Some error messages contain no useful or even inaccurate information, which increases confusion. For example, there was a version where referencing a model repository by branch name did not work despite being documented as a feature. When trying to do so, the error message reads something like "Repository not found", while, in fact, the repository was found, but the branch reference was not.

Because SDC is still a relatively new tool without a very broad user base, it's no surprise that the variety of additional resources that can be found online is very small to almost non-existent.

To account for that, to spread the news on the tool and to get feedback from the community, the developers not only hold presentations and workshops at conferences but also create regular online talks/meetings. Furthermore, they provide support on their Discord server so everyone can ask questions and get help with any problems. The authors find that quite admirable.

Nevertheless, since there is expected to be no commercial support offers for SDC to be sold by anyone anytime soon, organizations that want to use SDC in production environments should expect extensive

efforts needed to not only adapt the tool and especially its resources (YANG models) to their environment but also to build up operational know-how and troubleshooting knowledge.

8. Outlook

This project provides an analysis of a novel approach to network device configuration. While the ideas behind the approach are valid and provide a benefit to network operation, the tools in their current state are still too raw of a framework. Transitioning from an implementation project to an analysis, it might be beneficial to investigate some additional aspects of product quality requirements according to a standardized framework[39].

As a next step, the authors would see the value in a more detailed documentation of all the functions and internal processes that make up the application. Furthermore, network engineers trying the tool out and evaluating it would benefit from more concise error messages in the event of failures.

Lastly, there are many components providing the logic for not only SDC but also its accompanying tool Choreo. These components provide specific functionality, either for building abstract network architecture designs and configuration or for rendering the abstract configuration into device specific artifacts. So far, the developers provide a collection of these necessary to demonstrate their approach. To make these tools useful in productive environments, a broader variety of the components should be made available by either the developers or the community. This would also lower the initial effort for users.

8.1 Contributions

Based on the experiences made during the testing of SDC, the authors found some improvements to both the deployment configuration of SDC as well as its documentation for developers. These improvements led to two pull-requests made by us:

- Documentation: Explain the use of a named port for Telepresence.
(<https://github.com/sdcio/docs/pull/84>)
- Deployment configuration: Use a named targetPort for the data-service to enable easier debugging with Telepresence without the need for an init-container.
(<https://github.com/sdcio/config-server/pull/280>)

At the time of writing this chapter, both pull requests are still to be reviewed by the project's maintainers.

9. Conclusion

In the beginning of the project stood the task of knowledge acquisition. Both the use case EVPN as well as the given tool stack were new territory for the authors. Learning about EVPN was greatly facilitated through the available CN2 lab documentation. The research on SDC on the other hand proved to be more difficult because the available documentation is not extensive enough.

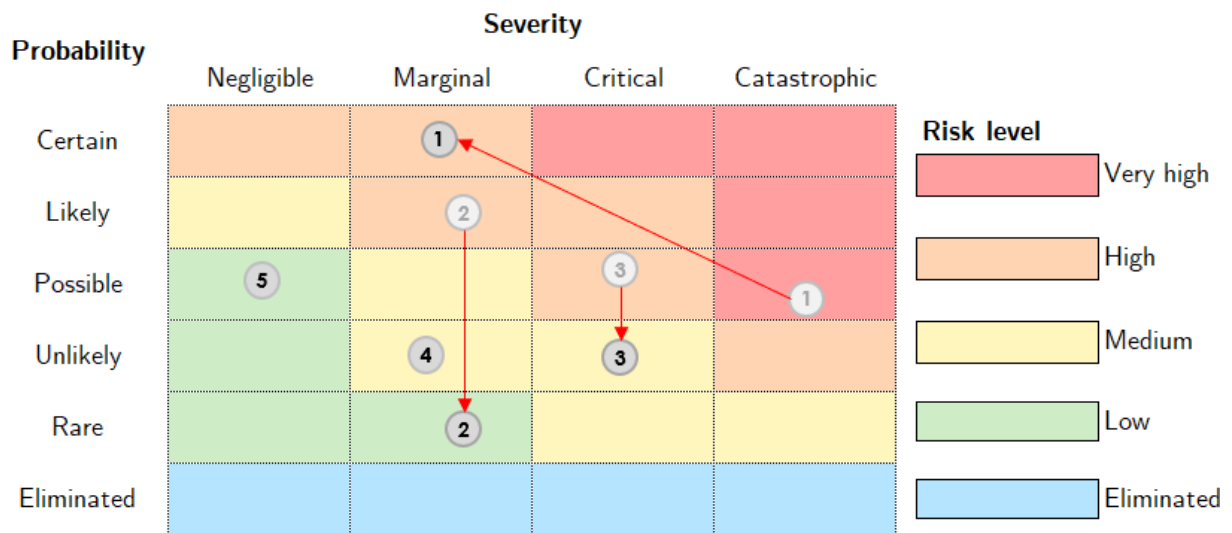
The proposed approach at network configuration with SDC using Kubernetes is a novel and interesting one. The concrete implementation being still in early development led the authors to face a variety of difficult to non-solvable problems, given the time schedule. This necessitated changes in the direction planned for the project.

Overall, this work provides an introduction to the topic along with an analysis of the environment and a verdict on the readiness of the currently available implementation.

10. Risk Analysis

Issues can always arise, when working on a project. In preparation potential risks were identified that could arise along the way. The risks have been assessed according to their possibility and impact. Mitigation strategies have been developed in order to reduce or eliminate them.

10.1 Overview



Risks

- 1 Maturity of available tools
- 2 Documentation / community support
- 3 Vendor compatibility
- 4 Adaptions to the vendor
- 5 Illness of a team member

Figure 57 Risk Overview

10.2 Details

1. Maturity of the tool stack

The given tool stack for Cloud-native network management is still a very new one. Furthermore, tools using Kubernetes to do so are very rare. They are also still in heavy development, so stable operation, vendor compatibility and desired functionality scope are highly unknown.

Mitigation: The authors will keep the big picture in mind and focus on concepts as well in order not to get lost in small operational problems or functional bugs.

2. Documentation/community support

Because of a small developer/user community, extensive documentation and timely and competent support in case of problems are likely not to be found.

Mitigation: The authors will keep the big picture in mind and focus on concepts as well, in order to not get lost in small operational problems or functional bugs.

3. Vendor compatibility

Because the tools are still a novelty, the developers are expected to start out ensuring compatibility with one vendor first and to adapt to different vendors slowly.

Mitigation: The authors keep multiple available vendors in mind so they can switch in case of unsolvable incompatibilities or highly extensive workarounds.

4. Adaptions to the vendor

The team will need to adapt some components to the used vendor. With the limited availability of information resources, this could set the timeline back.

Mitigation: The authors will keep multiple vendors in mind to be able to use one that doesn't require many adaptions.

5. Illness of a team member

Due to the fact that this project has a tight schedule and fixed deadline, unexpected absences pose a risk to the project's timeline.

Mitigation: The authors set weekly meetings to get everyone on the same page and discuss any problems and open questions, ensuring that in the event of a team member's absence, work can continue.

10.3 Risk Development

10.3.1 16.10.2024

Risk 2

In an effort to better understand SDC and especially KubeNet, the authors met with Wim Henderickx, the main developer behind these two tools. Wim could help us understand what function the different components of KubeNet have, how they interact, and which logic they provide. This explanation helped us understand the process from a topology idea to the configured network devices. This risk is reduced to Unlikely.

10.3.2 21.10.2024

Risk 3

The vendor for the PoC was switched from Cisco to Arista due to better OpenConfig compatibility. This risk is reduced to Unlikely.

29.10.2024

Risk 2

The authors learned about Discord channels run by the developers and read by users as well as developers for the different tools. These can be used to post some of the problems around getting SDC to work and get hints and solution ideas. This risk is reduced to rare.

10.3.3 18.11.2024

Risk 1

After learning a lot about the functionality of SDC in conjunction with KubeNet and encountering many problems along the way, it was decided to switch the main focus for the remainder of the project. While still trying to get a PoC working in SDC, the authors will concentrate their efforts on the integration of SDC with other cloud-native tools and its usage in production. This new focus aims to provide an assessment of whether SDC is already mature enough to be used in its current development state. The severity is reduced to marginal, but the possibility is raised to certain based on the authors' findings.

11. Project Plan

11.1 Roles and obligations

11.2 Process Model

It was decided to use the Kanban-Method for the short-term planning and the RUP Process model for the long-term planning in this project. The decision is based on the fact that the Kanban method is easy to use, the team is small, so there is no need for extra complexity, and Kanban is already known from the SA and SEP projects. The RUP Process Model is also already known and found to be self-comprehending by the team.

11.3 Time Management

11.3.1 Long-term planning

The long-term planning is for the whole Semester. The project phases, Inception, Elaboration, Construction, and Transition, are mapped in the diagram. It also includes all the milestones. The plan is separated per week. Below is the RUP plan. Also, see chapter Fehler! Verweisquelle konnte nicht gefunden werden. for a retrospective.

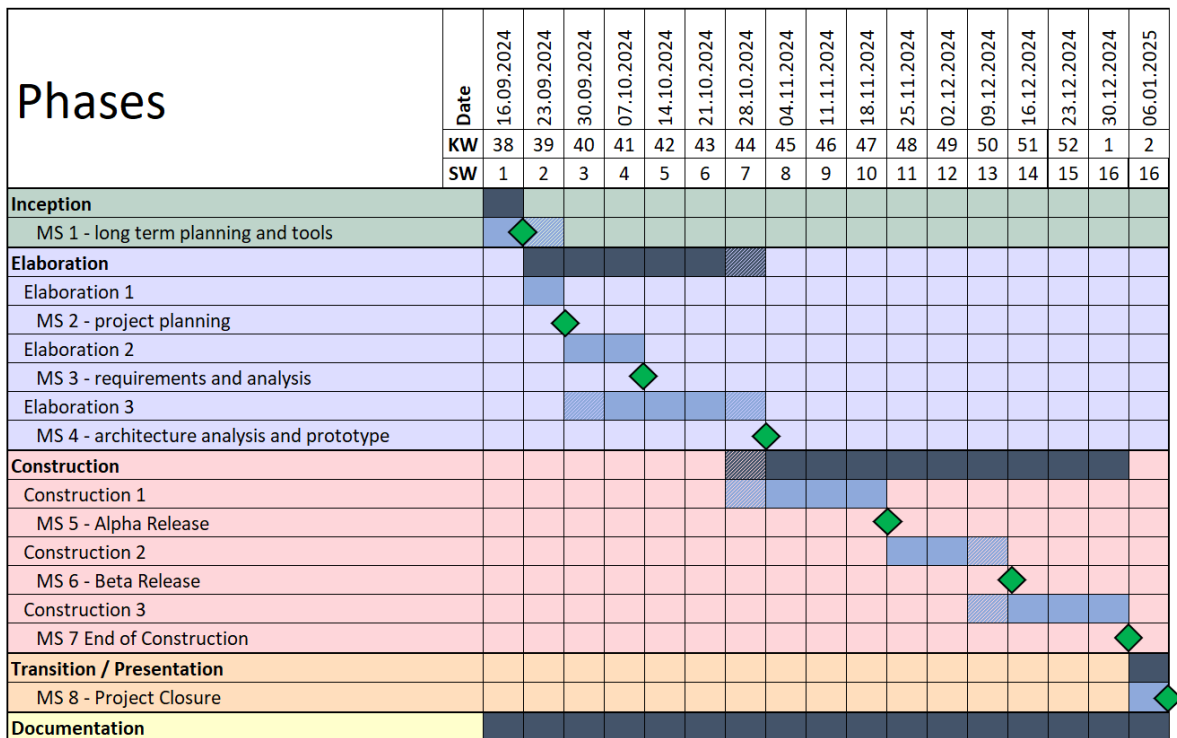


Figure 58 Long-term Plan

Symbols:

- In the diagram, the striped boxes are optional and can be used if needed. In case the team doesn't need the time, they can proceed with the next phase.

- ◆ The green squares are milestones.

11.3.2 Short-Term Planning

For short-term planning, the Kanban-Method is used. It was decided to use the tool in Teams, “Planner”. It is best suited for the team because it’s already known from work, and the team didn’t want to use many different tools, keeping it simple and clean.

11.4 Collaboration

11.4.1 Communication

MS Teams is used to communicate between the two team members.

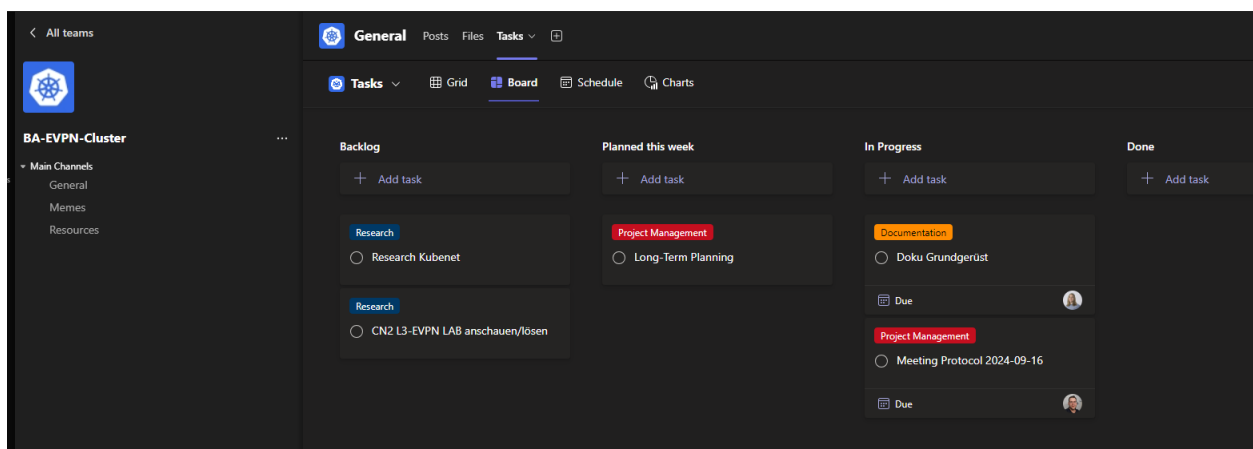


Figure 59 MS Teams Planner

In MS Teams, the team has defined the Buckets “Backlog”, “Planned this week”, “In Progress”, and “Done”. Team members and due dates can be assigned to tasks.

11.4.2 Time Tracking

As Toggl was already known to the authors from the term project (SA) and proved to be satisfactory, it was decided to use Toggl again. In Toggl, various projects were created to represent the different disciplines. Toggl allows the team to automatically create reports, group time by discipline and team member, and filter by time range. Below is a screenshot of Toggl and our defined projects within the workspace “BA”.

The screenshot displays the Toggl Track 'Projects' page. On the left is a dark sidebar with navigation options under 'TRACK', 'ANALYZE', and 'MANAGE'. The 'Projects' option is highlighted. The main content area shows a table of projects with the following data:

PROJECT	CLIENT	TIMEFRAME	TIME STATUS	TEAM
Analysis		16 Sep	0 h	Patrick, Vanessa
Development		16 Sep	0 h	Patrick, Vanessa
Documentation		16 Sep	1.2 h	Patrick, Vanessa
Modelling		16 Sep	0 h	Patrick, Vanessa
Project Management		16 Sep	3 h	Patrick, Vanessa
Requirements		16 Sep	0 h	Patrick, Vanessa
Research		16 Sep	0 h	Patrick, Vanessa
Testing		16 Sep	0 h	Patrick, Vanessa

Figure 60 Toggl Track for Time Tracking.

11.4.3 File Storage, Versioning and Review

For this is the OST Gitlab used. To ensure the expected code quality, each merge request is assigned to the other team member for review. This review ensures also that the code to be merged is structured in a comprehensible fashion.

In Gitlab, the team used a group and set up three different repositories.

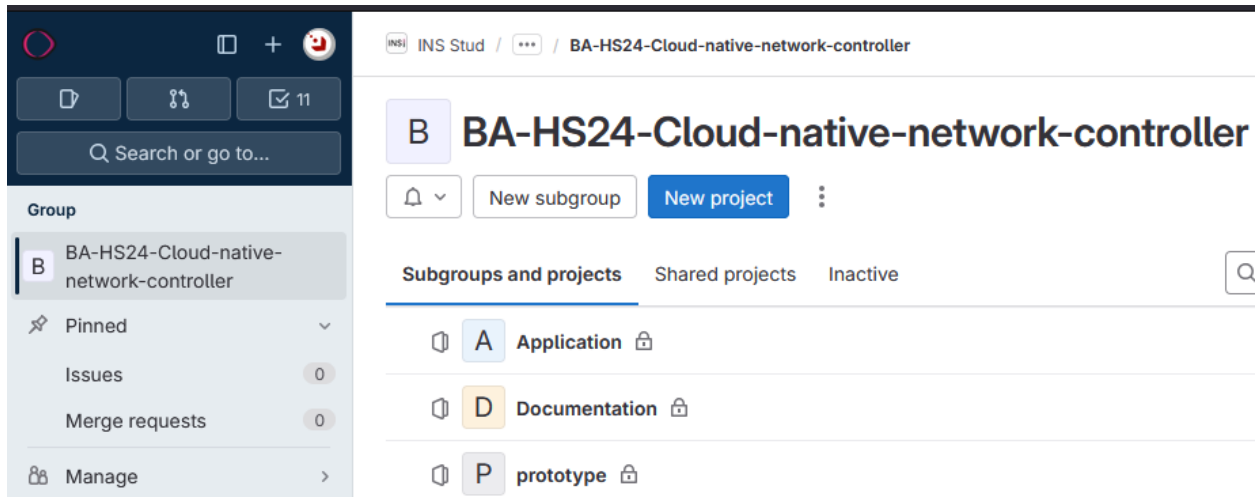


Figure 61 Gitlab Repositories

The authors will also work with MS Teams and use SharePoint as file storage for the documentation to ensure synchronization and versioning.

11.5 Meetings

Meetings are set up on Monday, 10:00 am, with the supervisors Urs Baumann and Jan Untersander and the developers Patrick Lenherr and Vanessa Gyger.

12. Appendix

12.1 Glossary

Abbreviation	Name	Description
API	Application Programming Interface	A way for applications to communicate with each other.
AS	Autonomous Systems	A collection of networks operated by a single organization. All AS together represent the internet.
ASN	Autonomous System Number	Unique identifier of an AS.
BGP	Border Gateway Protocol	A routing protocol between routers.
BYOA	Bring Your Own API	In terms of Choreo, extend the functionality with a custom API.
BYOS	Bring Your Own Schema	In terms of Choreo, extend the compatibility by adding a custom schema for the network device.
CI/CD	Continuous Integration and Continuous Deployment	The practice of automating the integration of code contributions.
CLI	Command-line interface	A way to interact with a program through text commands.
CN2	Computer Networks 2	A course for the bachelor's studies in computer science at OST.
CNF	Container-based networking functions	Network functions provided by a containerized device
CR	Custom Resource	An instance of a CRD.
CRD	Custom Resource Definitions	Define additional types of objects in K8s.
EVPN	Ethernet Virtual Private Network	A network overlay protocol, to span L2 and L3 networks over a routed underlay
GENID	Generation ID	Generation of unique ID.

Abbreviation	Name	Description
gNMI	gRPC Network Management Interface	It is a network protocol for configuration manipulation and state retrieval. It's an enhancement of gRPC[40].
gRPC	Google Remote Procedure Calls	It is a framework which provides communication between clients.
IaC	Infrastructure as Code	Define infrastructure by using code.
IANA	Internet Assigned Numbers Authority	Global authority in managing various network identifiers such as the DNS root and IP-addressing.
IETF	Internet Engineering Task Force	A global organization, responsible for releasing standards in the network environment
INS	Institute for Networks and Security	Institute at OST Eastern Switzerland University of Applied Sciences.
IPAM	IP Address Management	Managing IP address inventories.
K8s	Kubernetes	Orchestration system for automating resource deployments in a declarative way.
KRM	Kubernetes Resource Model	An architecture pattern for defining, managing, and interacting with Kubernetes resources
Kuid	Kubernetes Identities	An extension of Kubernetes, a collection of tools to manage inventory and identifiers
NOOP	No operation	A testing target for SDC that doesn't do anything.
OSPF	Open Shortest Path First	A link-state network routing protocol.
PIM	Protocol-Independent Multicast	Derived from multicast, it defines a one-to-many distribution.
PNF	Physical Network Functions	Network functions provided by a physical appliance.
PoC	Proof of Concept	A test proving the feasibility of an idea.
SA	term project	A project in bachelor's degree studies is required for the bachelor thesis.

Abbreviation	Name	Description
SDC	Schema Driven Configuration	A declarative way to deploy configuration as K8s resources to a network device.
SSOT	Single Source of Truth	One place to store all needed information, avoiding conflicts
VLAN	Virtual Local Area Network	A mechanism to separate logical networks in a single physical one
VM	Virtual Machine	An OS running within a separate environment on the physical machine.
VNF	Virtual Network Functions	Network functions provided by VM.
VXLAN	Virtual Extensible LAN	Uses similar encapsulation as VLAN using a 24-bit VLAN ID[41]
WSL	Windows-Subsystem for Linux	A Linux environment within Windows.
YAML	Yet Another Markup Language	It is a human-readable data serialization configuration file format, which is similar to XML.
YANG	Yet Another Next Generation	A data-modeling language.

Figure 62 Glossary

12.2 List of Figures

Figure 1 Overview of the project path [1]	10
Figure 2 Proposed Ecosystem	11
Figure 3 Ecosystem Workflow.....	12
Figure 4 VXLAN tunneling[2]	13
Figure 5 Overview of EVPN with MP-BGP and VNIs[3].....	13
Figure 6 Spine-Leaf topology[4]	14
Figure 7 SDC Logo[1].....	16
Figure 8 SDC components[11]	16
Figure 9 SDC Workflow	17
Figure 10 Overview of SDC Manifests	18
Figure 11 Choreo KForm Logo[42]	19
Figure 12 Diagram Configuration Rendering	19
Figure 13 Specific Device Configuration.....	20
Figure 14 Choreoctl Help Command.....	21
Figure 15 Starlark symbol[43]	21
Figure 16 Infrahub Logo[44].....	22
Figure 17 Workflow of using Infrahub	23
Figure 18 Timeline of encountered Challenges	27
Figure 19 SDC and Kubenet.....	28
Figure 20 Components of SDC and Kuid	29
Figure 21 Enable the Netconf Service through Bash.....	30
Figure 22 Cisco hostname Deviation from OpenConfig in NX-OS 9.2-3.....	31
Figure 23 Cisco hostname Deviation from OpenConfig in NX-OS 10.4-1	31
Figure 24 Error when configuring Hostname on Arista	32
Figure 25 Schema load Error using proposed Schema Manifest.....	32
Figure 26 Interface Test Configuration #1 Figure 27 Interface Test Configuration #2	35
Figure 28 Schema Validation Error.....	35
Figure 29 The Interfaces Model of the Schema.....	35
Figure 30 Invalid UTF-8 Characters	37
Figure 31 Choreo HelloWorld Reconciler Example not working	38
Figure 32 Choreo HelloWorld Reconciler Example works now	39
Figure 33 SDC Debugging Telepresence Intercept Error	40
Figure 34 Logs from Traffic Manager Pod.....	41
Figure 35 Adding containerPort name to data-service	42
Figure 36 Adding "targetPort: data-service" to data-server	42
Figure 37 Arista 4.33.0F Error Config change Hostname	46
Figure 38 Configuration Parts Underlay.....	48
Figure 39 Configuration Parts Overlay.....	48
Figure 40 Configuration Parts VLAN.....	48
Figure 41 Configuration Parts VLAN mapping.....	49
Figure 42 Configuration Parts Underlay.....	49

Figure 43 Configuration Parts Overlay.....	49
Figure 44 Configuration Parts Underlay.....	50
Figure 45 Configuration Parts Overlay.....	50
Figure 46 Partitioning of the Configuration Parts	52
Figure 47 Dependencies of the different Configuration Parts.....	53
Figure 48 Warning IP Routing	54
Figure 49 OSPF successfully activated.	54
Figure 50 Configuration Parts composing the Tasks.....	55
Figure 51 Interfaces on Device before adding system0	55
Figure 52 Interfaces after adding the Configuration	55
Figure 53 The Interface is gone again.....	56
Figure 54 Configuration Sources for SDC	56
Figure 55 Configuration Intent with Label targetSelector; taken from [45].....	59
Figure 56 Fulfillment of Requirements.....	62
Figure 57 Risk Overview	68
Figure 58 Long-term Plan.....	71
Figure 59 MS Teams Planner	72
Figure 60 Toggl Track for Time Tracking.	73
Figure 61 Gitlab Repositories	74
Figure 65 Glossary.....	77

12.3 Bibliography & References

- [1] sdcio team. "SDC Logo." Accessed: Jan. 10, 2025. [Online]. Available: <https://docs.sdcio.dev/assets/logos/SDC-transparent-noname-100x100.png>
- [2] "What is VXLAN?" [Online]. Available: <https://support.huawei.com/enterprise/en/doc/EDOC1100086966>
- [3] Pasanen, Toni. "VXLAN Part VI: VXLAN BGP EVPN – Basic Configurations." [Online]. Available: <https://nwktimes.blogspot.com/2018/04/vxlan-part-vi-vxlan-bgp-evpn-basic.html>
- [4] Juniper Networks. "Was ist EVPN-VXLAN?" [Online]. Available: <https://www.juniper.net/de/de/research-topics/what-is-evpn-vxlan.html>
- [5] Cloud Native Computing Foundation. "Kubernetes (K8s)." [Online]. Available: <https://github.com/kubernetes/kubernetes/>
- [6] The Kubernetes Authors. "Kubernetes Documentation." [Online]. Available: <https://kubernetes.io/docs/>
- [7] "Network Modeling (netmod)." [Online]. Available: <https://datatracker.ietf.org/wg/netmod/documents/>
- [8] M. Bjorklund. "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)." [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc6020#section-4.1>
- [9] "The YANG Data Modeling Language." [Online]. Available: <https://developer.cisco.com/docs/nso-guides-6.3/the-yang-data-modeling-language/#yang-introduction>
- [10] OpenConfig Project. "Data models." [Online]. Available: <https://openconfig.net/projects/models/>
- [11] sdcio team. "SDC (Schema Driven Configuration)." Accessed: Nov. 26, 2024. [Online]. Available: <https://docs.sdcio.dev/>
- [12] sdcio team. "Colocated file." [Online]. Available: <https://docs.sdcio.dev/artifacts/basic-usage/colocated.yaml>
- [13] Choreo Team. "Welcome to Choreo." Accessed: Dec. 12, 2024. [Online]. Available: <https://choreo-docs.kform.dev/#filtering-example-using-cel-expressions>
- [14] Toon Van Deuren. "Kubernetes Scaling: The Event Driven Approach." Accessed: Jan. 10, 2025. [Online]. Available: <https://medium.com/kapuani/kubernetes-scaling-the-event-driven-approach-bdd58ded4e3f>
- [15] laurentlb. "Awesome Starlark." Accessed: Dec. 13, 2024. [Online]. Available: <https://github.com/laurentlb/awesome-starlark>
- [16] netbox-community. "Netbox Documentation." [Online]. Available: <https://netbox.itel.com/static/docs/>
- [17] OpsMill. "Transformation: Infrahub Documentation." Accessed: Dec. 17, 2024. [Online]. Available: <https://docs.infrahub.app/topics/transformation>
- [18] pkgserver team. "pkgserver." [Online]. Available: <https://docs.pkgserver.dev/>
- [19] kubenet team. "kubenet." [Online]. Available: <https://learn.kubenet.dev/>
- [20] kubenet team. "Kubenet Git Repositories." [Online]. Available: <https://github.com/kubenet-dev>

- [21] Cisco. "Cisco Nexus 9000 Series NX-OS Programmability Guide, Release 9.2(x)." [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/92x/programmability/guide/b-cisco-nexus-9000-series-nx-os-programmability-guide-92x/b-cisco-nexus-9000-series-nx-os-programmability-guide-92x_chapter_011010.html
- [22] YANG. "Yang Models Cisco NX-OS 9.2-3." [Online]. Available: <https://github.com/YangModels/yang/tree/main/vendor/cisco/nx/9.2-3>
- [23] sdcio team. "SDC User Guide Schema." [Online]. Available: <https://docs.sdcio.dev/user-guide/configuration/schemas/>
- [24] YANG. "Yang Models Cisco NX-OS 9.2-3 Deviations." [Online]. Available: <https://github.com/YangModels/yang/blob/main/vendor/cisco/nx/9.2-3/cisco-nx-openconfig-system-deviations.yang>
- [25] Arista Networks. "Yang Models Arista EOS 4.33.0F." [Online]. Available: <https://github.com/aristanetworks/yang/tree/master/EOS-4.33.0F>
- [26] Openconfig. "Openconfig Interfaces documentation." [Online]. Available: <https://openconfig.net/projects/models/schemadocs/yangdoc/openconfig-interfaces.html#interfaces-interface>
- [27] "Choreo-Examples." Accessed: Dec. 12, 2024. [Online]. Available: <https://github.com/kform-dev/choreo-examples>
- [28] sdcio team. "SDC Development." [Online]. Available: <https://docs.sdcio.dev/dev/dev/>
- [29] sdcio team. "Github data-server repository." [Online]. Available: <https://github.com/sdcio/data-server>
- [30] Vanessa. "Telepresence Slack Post." Accessed: Dec. 28, 2024. [Online]. Available: <https://cloud-native.slack.com/archives/C06B36KJ85P/p1731934835646289>
- [31] gnmic team. "gnMIC." [Online]. Available: <https://gnmic.openconfig.net/>
- [32] OpsMill. "Infrahub integrations." [Online]. Available: <https://docs.infrahub.app/overview/integrations/>
- [33] CNCF. "Canary Deployment." [Online]. Available: <https://glossary.cncf.io/canary-deployment/>
- [34] argoproj. "Argo Rollouts." [Online]. Available: <https://argoproj.github.io/argo-rollouts/features/canary/>
- [35] J. Suen. "Does Argo Rollouts support non-deployment resource?" [Online]. Available: <https://github.com/argoproj/argo-rollouts/discussions/1346>
- [36] The Flux authors. "Flux Documentation." [Online]. Available: <https://fluxcd.io/flux/>
- [37] The Flux authors. "Flagger Deployment Strategies." [Online]. Available: <https://fluxcd.io/flagger/usage/deployment-strategies/>
- [38] Larsson, Kristian. "YANG validation in the real world." [Online]. Available: <https://plajjan.github.io/2017-10-30-yang-validation-in-the-real-world.html>
- [39] 2024 iso25000.com. "ISO/IEC 25010." Accessed: Jan. 10, 2025. [Online]. Available: <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>
- [40] P. Borman *et al.* "gNMI." [Online]. Available: <https://www.openconfig.net/docs/gnmi/gnmi-specification/>

- [41] Wikipeda. "Virtual Extensible LAN." Accessed: Jan. 4, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Virtual_Extensible_LAN
- [42] Choreo Team. "Choreo KForm Logo." [Online]. Available: <https://choreo-docs.kform.dev/assets/logos/KForm-transparent-noname-96x96.png>
- [43] Bazelbuild. "starlark." Accessed: Dec. 13, 2024. [Online]. Available: <https://github.com/bazelbuild/starlark>
- [44] OpsMill. "Infrahub Logo." Accessed: Dec. 17, 2024. [Online]. Available: <https://github.com/opsmill/infrahub-demo-edge>
- [45] sdcio team. "SDC User Guide ConfigSet." [Online]. Available: <https://docs.sdcio.dev/user-guide/configuration/config/configset/>