The Cloud-Native Advantage: Intent-based Network Automation

10.09.2025
Alexander North
Markus Vahlenkamp

Alexander North
Senior DevOps Engineer
Swisscom

alexander.north1@swisscom.com
Slack: CNCF, Nephio, NetDev

Markus Vahlenkamp
Technology & Architecture Consultant
Nokia

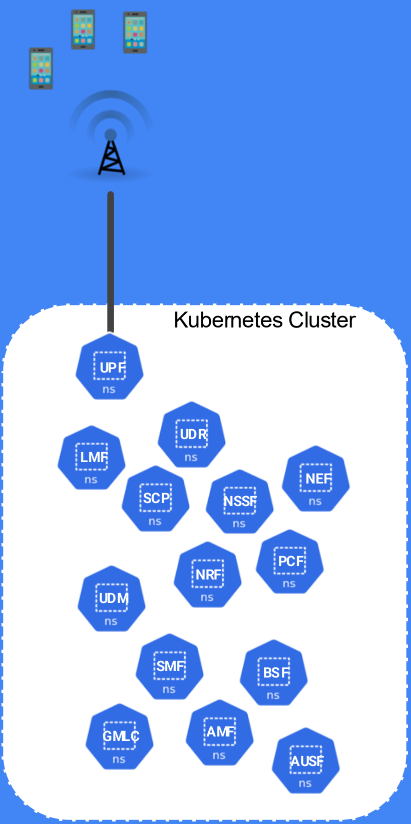
markus.vahlenkamp@nokia.com

What is a 5G Mobile Core?

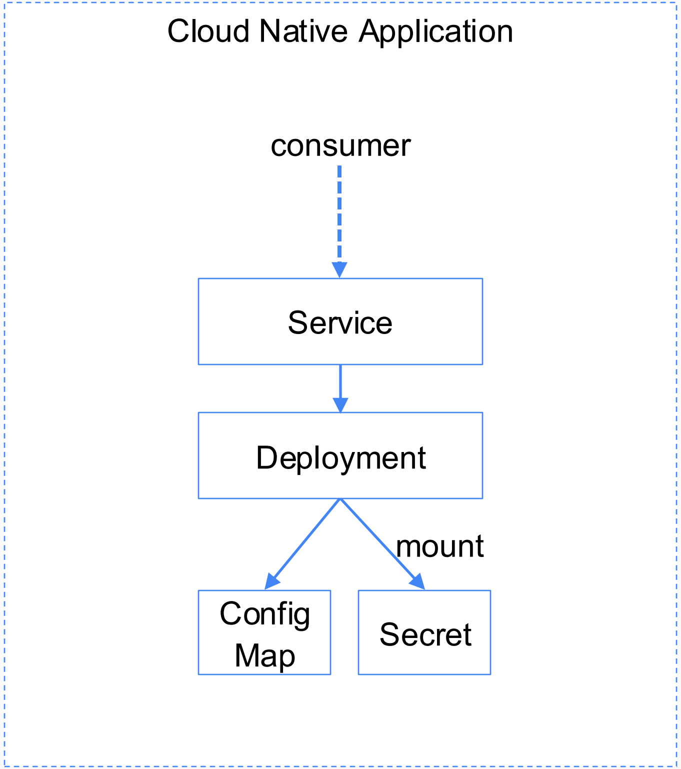
- Each blue object
  - is a «CNF» aka «Cloud-Native Network Function»
    - e.g. Router (UPF), Authentication Service (AUSF)
  - Deployed using Helm

- Configuration is done via
  - Helm Values
  - Other Configuration Interfaces
    - NETCONF, REST

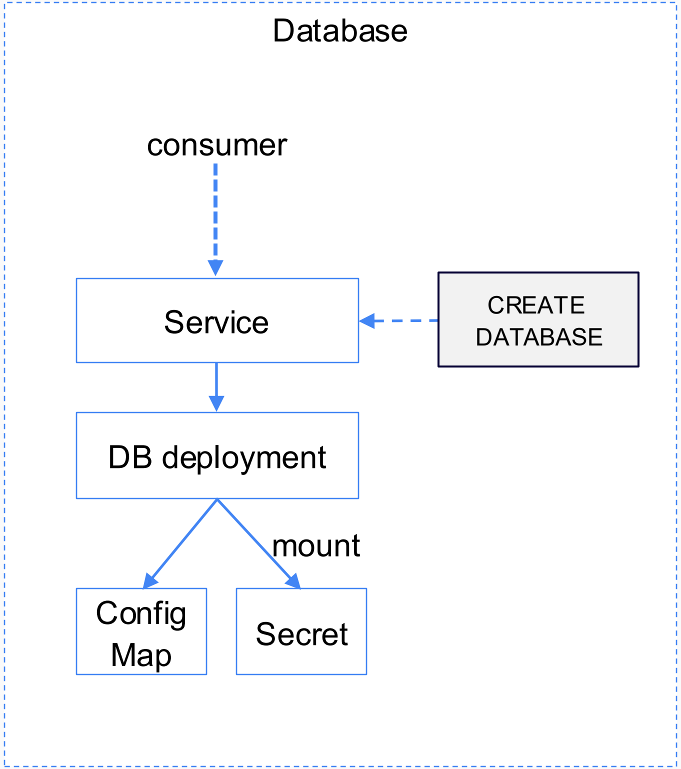
- Scale
  - A development environment contains ~2000 pods
  - A total of **5000 interdependent configuration parameters**



Configuration of Applications



Configuration of Applications



Database

consumer

Service

CREATE DATABASE

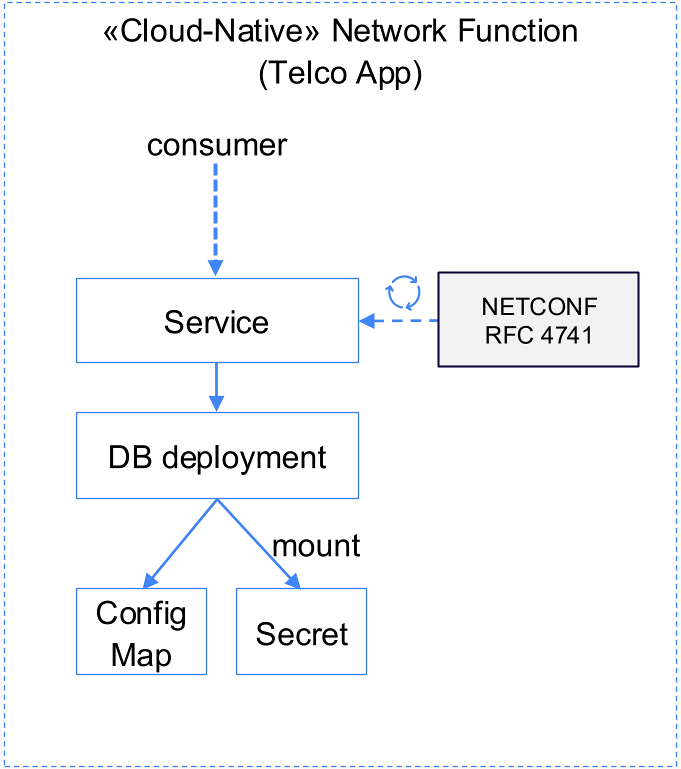
DB deployment

mount

Config Map

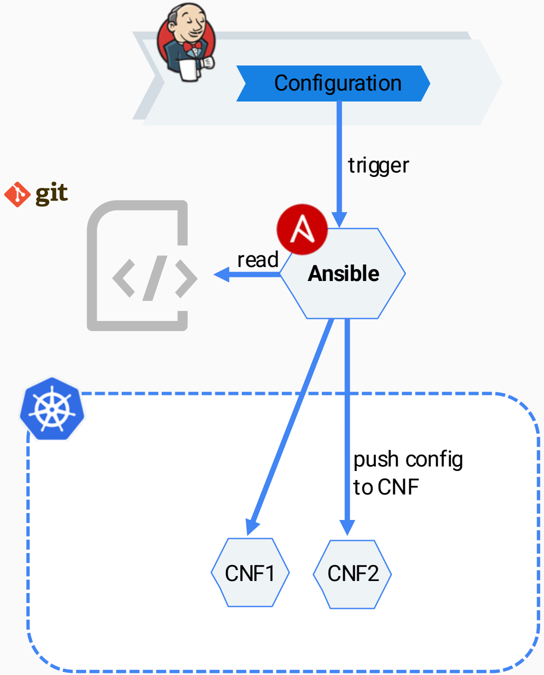
Secret

Configuration of Applications



Imperative Approach

- One-Shot
- Slow iterations
- Lack of robust monitoring
- Either Tracability or Holistic view



Kubernetes Resource Model (KRM)

API extensions

Custom Resource Definitions extend the Kubernetes API.

CRs as Instances

Custom Resources instantiate a CRD.

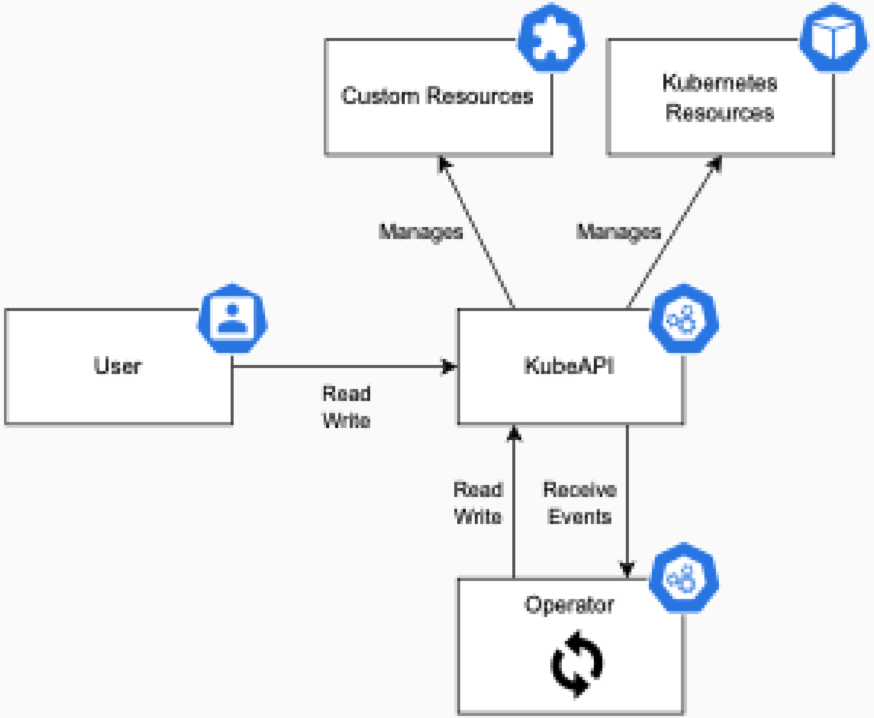
Business Logic

Use of Operators or templates to run custom logic.

Kubernetes Operators

Kubenet

- The Control Loop(s)
  - Watches for changes
  - Performs action
  - Updates Resources
- Built-in
  - Deployment
  - Job
  - PersistentVolume
- Third-Party
  - Prometheus Operator
  - External Secrets Operator
  - cert-manager



Networking Configuration Protocols

	NETCONF	gNMI	RESTCONF
Transport	SSH	gRPC	HTTP(S)
Encoding	XML	PROTO JSON JSON_IETF	JSON_IETF XML

- Common base (YANG schema)
  - Conversion between encodings
  - Validation

YANG – Network Function Schema

Schema:

```
list device {
  key "name";
  min-elements 1;

  leaf name { type string; }
  leaf role {
    type enumeration { enum spine; enum leaf; }
    mandatory true;
  }

  leaf mgmt-ip {
    type inet:ipv4-address;
    must "starts-with(., 10.)" {
      error-message "IP must be in 10.x.x.x range";
    }
  }
}
```

Configuration:

```
<device>
    <name>leaf-1</name>
    <role>leaf</role>
    <mgmt-ip>10.1.1.10</mgmt-ip>
</device>
<device>
    <name>spine-1</name>
    <role>spine</role>
    <mgmt-ip>10.2.0.1</mgmt-ip>
</device>
```

Schema Driven Configuration (SDC)

- Cloud Native Network Management
- Developed at Nokia, open sourced 2024, available on GitHub
- Community
  - Discord channel for latest news
  - Meetup every Monday at 14:00 CET
- Aiming to become a CNCF project

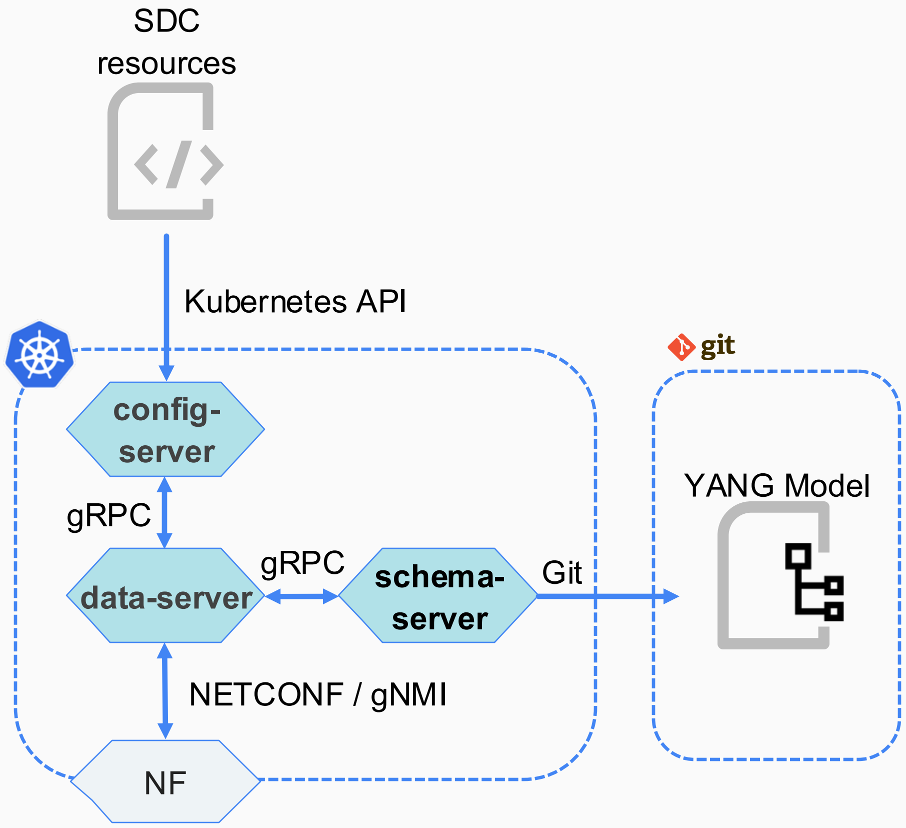
Components

- Config-Server

  Aggregated API Server
- Data-server

  Communicate with NF, maintain in-memory config tree, validation
- Schema-Server

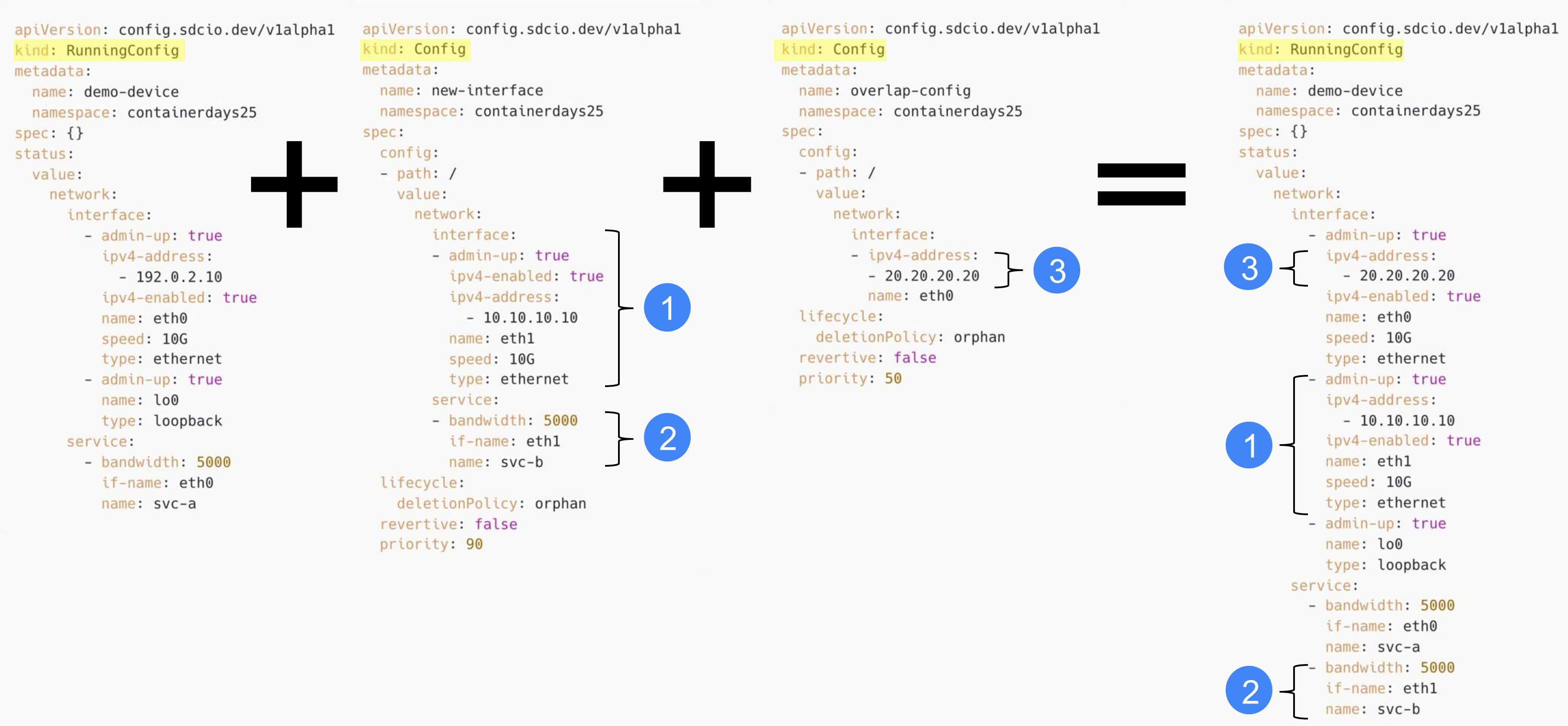
  path-based schema element store



Main Custom Resource Definitions

- Targets
  - Status of the NF within SDC
- Configs
  - Target bound Config (Snippets)
- ConfigSets
  - Blueprint Config (Snippets)
- RunningConfigs
  - Target Running Configuration
- Deviations
  - Configuration Drift Report (actual vs. expected)
- ConfigBlames
  - Configuration Tree annotated with ConfigCR sources

Configuration Intents



```yaml
apiVersion: config.sdcio.dev/v1alpha1
kind: RunningConfig
metadata:
  name: demo-device
  namespace: containerdays25
spec: {}
status:
  value:
    network:
      interface:
        - admin-up: true
          ipv4-address:
            - 192.0.2.10
          ipv4-enabled: true
          name: eth0
          speed: 10G
          type: ethernet
        - admin-up: true
          name: lo0
          type: loopback
      service:
        - bandwidth: 5000
          if-name: eth0
          name: svc-a
```

+

```yaml
apiVersion: config.sdcio.dev/v1alpha1
kind: Config
metadata:
  name: new-interface
  namespace: containerdays25
spec:
  config:
  - path: /
    value:
      network:
        interface:
        - admin-up: true
          ipv4-enabled: true
          ipv4-address:
            - 10.10.10.10
          name: eth1
          speed: 10G
          type: ethernet
        service:
        - bandwidth: 5000
          if-name: eth1
          name: svc-b
  lifecycle:
    deletionPolicy: orphan
  revertive: false
  priority: 90
```

+

```yaml
apiVersion: config.sdcio.dev/v1alpha1
kind: Config
metadata:
  name: overlap-config
  namespace: containerdays25
spec:
  config:
  - path: /
    value:
      network:
        interface:
        - ipv4-address:
          - 20.20.20.20
          name: eth0
  lifecycle:
    deletionPolicy: orphan
  revertive: false
  priority: 50
```

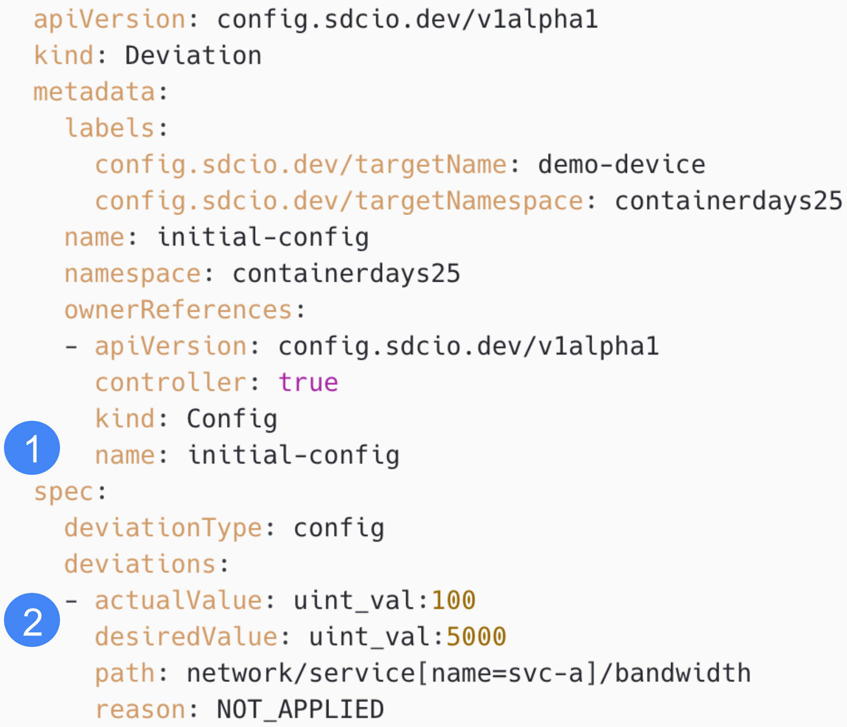
=

```yaml
apiVersion: config.sdcio.dev/v1alpha1
kind: RunningConfig
metadata:
  name: demo-device
  namespace: containerdays25
spec: {}
status:
  value:
    network:
      interface:
        - admin-up: true
          ipv4-address:
            - 20.20.20.20
          ipv4-enabled: true
          name: eth0
          speed: 10G
          type: ethernet
        - admin-up: true
          ipv4-address:
            - 10.10.10.10
          ipv4-enabled: true
          name: eth1
          speed: 10G
          type: ethernet
        - admin-up: true
          name: lo0
          type: loopback
      service:
        - bandwidth: 5000
          if-name: eth0
          name: svc-a
        - bandwidth: 5000
          if-name: eth1
          name: svc-b
```

How do we report deviating configuration state?

1 Deviation Parent

2 Deviation Details



```yaml
apiVersion: config.sdcio.dev/v1alpha1
kind: Deviation
metadata:
  labels:
    config.sdcio.dev/targetName: demo-device
    config.sdcio.dev/targetNamespace: containerdays25
  name: initial-config
  namespace: containerdays25
  ownerReferences:
  - apiVersion: config.sdcio.dev/v1alpha1
    controller: true
    kind: Config
    name: initial-config
spec:
  deviationType: config
  deviations:
  - actualValue: uint_val:100
    desiredValue: uint_val:5000
    path: network/service[name=svc-a]/bandwidth
    reason: NOT_APPLIED
```

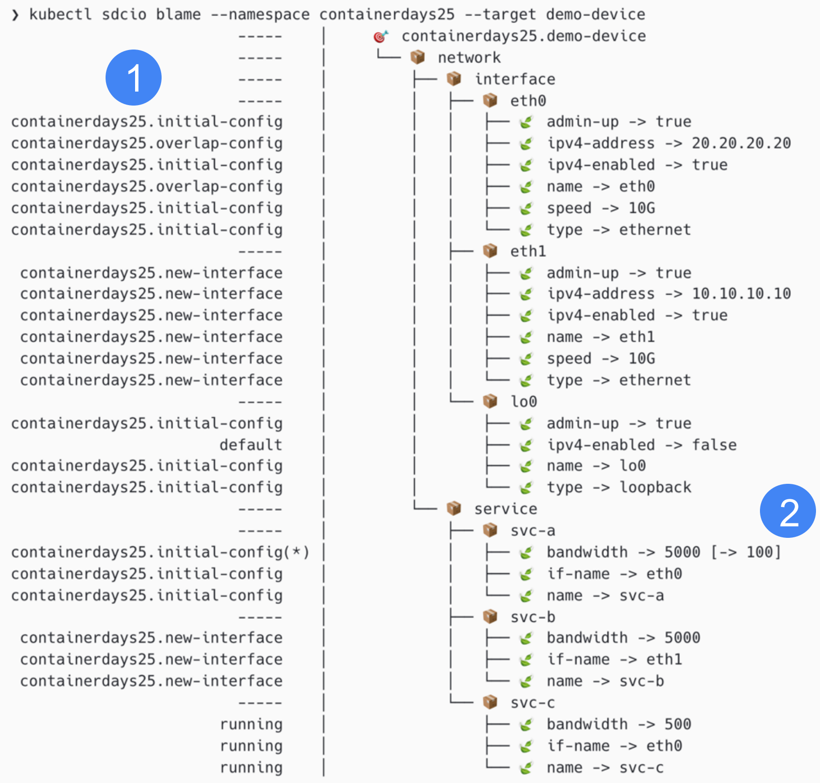
Deviation Reasons:
- NOT_APPLIED -> not set on device
- OVERRULED -> not highest priority
- UNHANDLED -> brownfield config

How can we find out where a parameter came from?

Config Blame is to Config apply what
Git Blame is to Commit

Each parameter shows its:

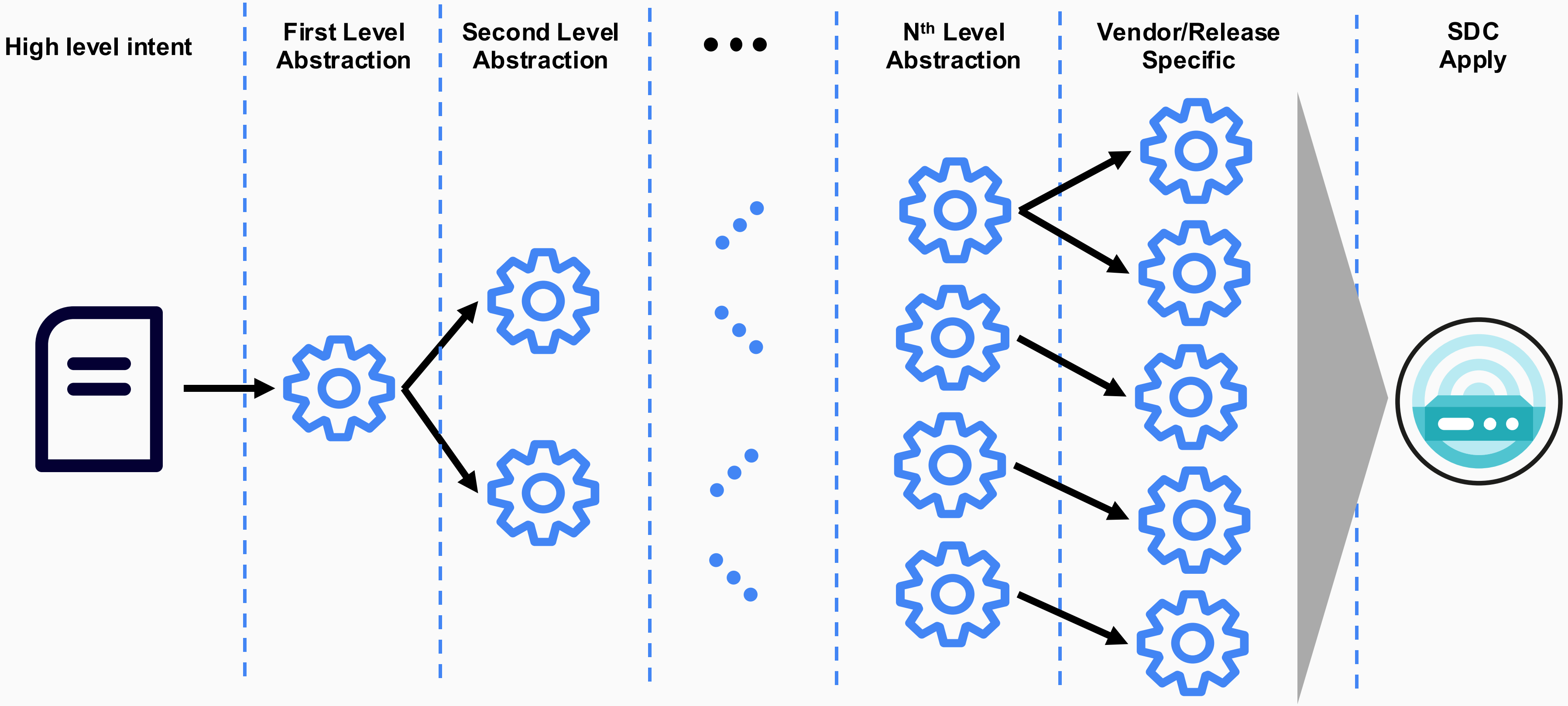
1. Source
2. Actual values and Deviations



```
❯ kubectl sdcio blame --namespace containerdays25 --target demo-device
                            ----- |  🎯 containerdays25.demo-device
                            ----- |  └── 📦 network
                            ----- |      ├── 📦 interface
                            ----- |      │   ├── 📦 eth0
   containerdays25.initial-config |      │   │   ├── 🍃 admin-up -> true
   containerdays25.overlap-config |      │   │   ├── 🍃 ipv4-address -> 20.20.20.20
   containerdays25.initial-config |      │   │   ├── 🍃 ipv4-enabled -> true
   containerdays25.overlap-config |      │   │   ├── 🍃 name -> eth0
   containerdays25.initial-config |      │   │   ├── 🍃 speed -> 10G
   containerdays25.initial-config |      │   │   └── 🍃 type -> ethernet
                            ----- |      │   ├── 📦 eth1
    containerdays25.new-interface |      │   │   ├── 🍃 admin-up -> true
    containerdays25.new-interface |      │   │   ├── 🍃 ipv4-address -> 10.10.10.10
    containerdays25.new-interface |      │   │   ├── 🍃 ipv4-enabled -> true
    containerdays25.new-interface |      │   │   ├── 🍃 name -> eth1
    containerdays25.new-interface |      │   │   ├── 🍃 speed -> 10G
    containerdays25.new-interface |      │   │   └── 🍃 type -> ethernet
                            ----- |      │   └── 📦 lo0
   containerdays25.initial-config |      │       ├── 🍃 admin-up -> true
                          default |      │       ├── 🍃 ipv4-enabled -> false
   containerdays25.initial-config |      │       ├── 🍃 name -> lo0
   containerdays25.initial-config |      │       └── 🍃 type -> loopback
                            ----- |      └── 📦 service
                            ----- |          ├── 📦 svc-a
containerdays25.initial-config(*) |          │   ├── 🍃 bandwidth -> 5000 [-> 100]
   containerdays25.initial-config |          │   ├── 🍃 if-name -> eth0
   containerdays25.initial-config |          │   └── 🍃 name -> svc-a
                            ----- |          ├── 📦 svc-b
    containerdays25.new-interface |          │   ├── 🍃 bandwidth -> 5000
    containerdays25.new-interface |          │   ├── 🍃 if-name -> eth1
    containerdays25.new-interface |          │   └── 🍃 name -> svc-b
                            ----- |          └── 📦 svc-c
                          running |              ├── 🍃 bandwidth -> 500
                          running |              ├── 🍃 if-name -> eth0
                          running |              └── 🍃 name -> svc-c
```

DEMO

Getting to the Config Resource



Benefits of Declarative Abstraction

Kubenet

Reduce Toil

Simplifies tasks by focusing on essential controls, reducing manual interventions.

Increased Reliability

Ensures consistent configurations across systems, minimizing errors and mismatches.

Enhanced Efficiency

Streamlines configuration processes, saving time and resources.

Scalability

Facilitates easier scaling by managing configurations at a higher abstraction level.

Q&A

https://docs.sdcio.dev/