Evolving GitOps: Harnessing Kubernetes Resource Model for 5G Core

Alexander North, Ashan Senevirathne & Joel Studler

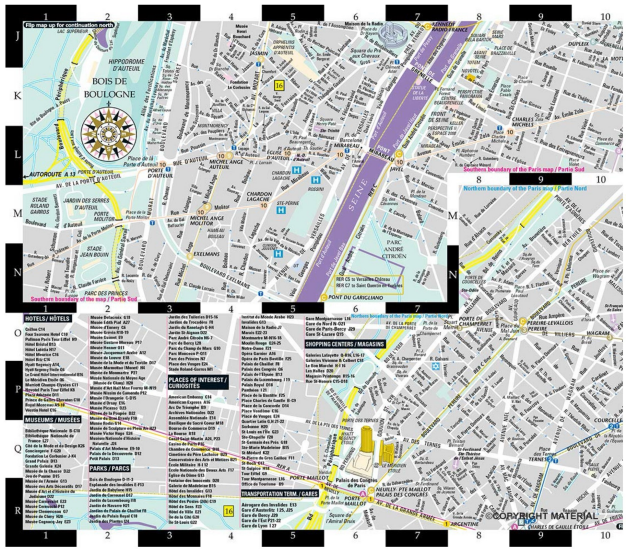
swisscom

An Analogy...

Static paper map

- Fixed
- Static
- Unchanging
- Overwhelming

This is GitOps today



Apple maps

- Dynamic
- Changes based on external conditions
- More focused
- Simple to navigate

This is GitOps w/ KRM

Alexander North
DevOps Engineer

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

Ashan Senevirathne
Product Owner

ashan.senevirathne@swisscom.com
Slack: CNCF, Nephio, NetDev

Joel Studler
Senior DevOps Engineer

joel.studler@swisscom.com
Slack: CNCF, Nephio, NetDev


swisscom

From Telco to TechCo: What Does It Mean for Us?

Cloud Nativeness

we need to introduce Reconciliation and leverage Kubernetes Patterns

Simplicity

we need to introduce abstraction in our configuration and break it into smaller manageable parts

Automation

we need to move from an imperative workflow-driven model to a declarative intent-based model

5G - driving our journey from Telco to TechCo
by Swisscom CTIO Mark Düsener at Connect Conference 2022
https://www.youtube.com/watch?v=hND7TiXJED8

What Is a 5G Core?

Each blue object

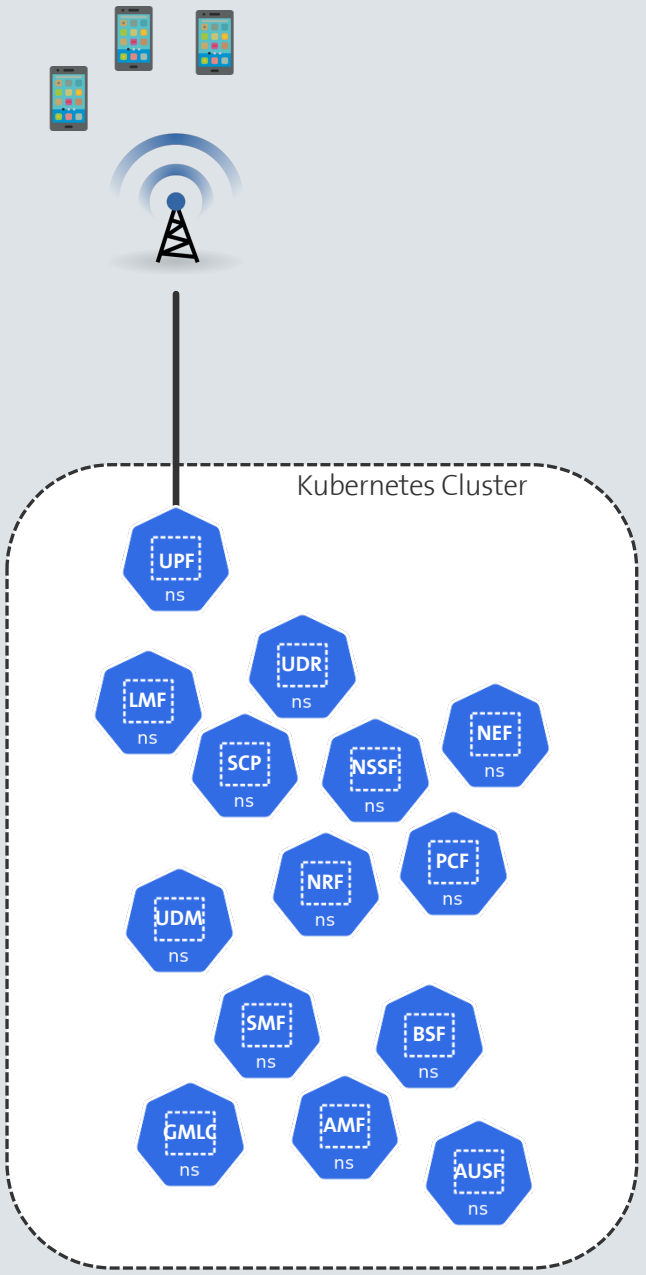
- is a «CNF» aka «Containerized Network Function»
  - e.g. Router (UPF), Authentication Service (AUSF)
- Deployed using Helm

Configuration is done via

- Helm Values
- Other Configuration Interfaces

Scale

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



What Is a 5G Core?

IP addresses

Subnets

VLANs

DNS Records

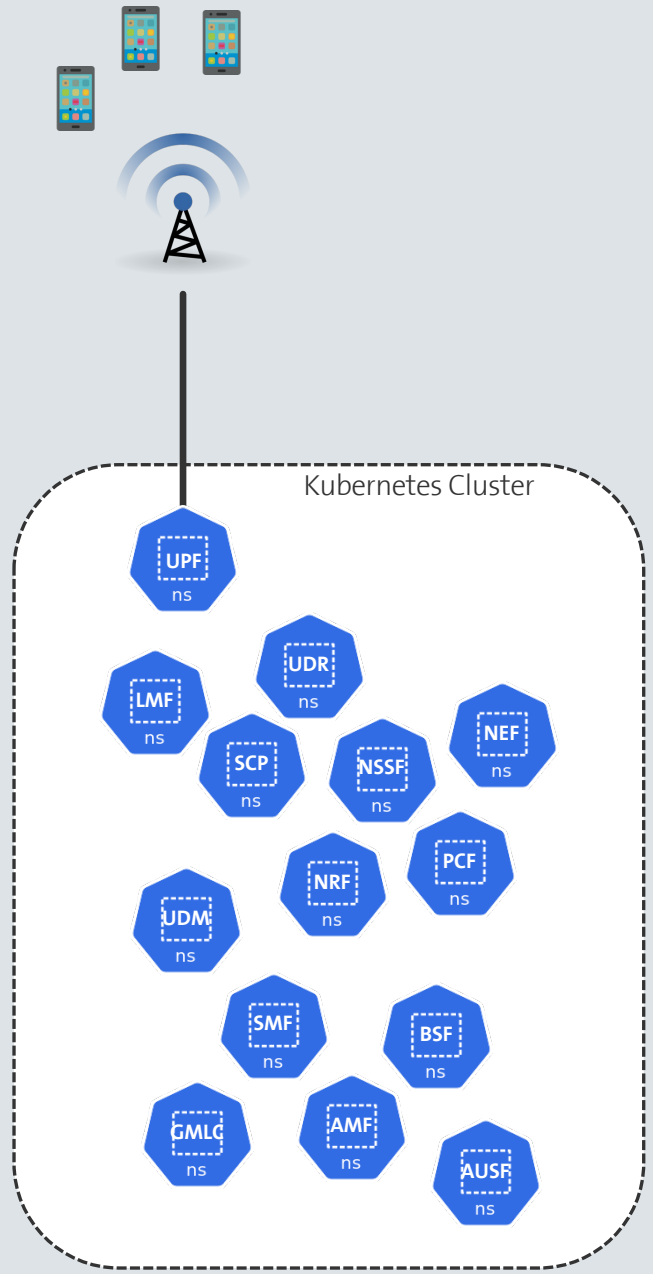
Network function variables

Infrastructure variables

Network function-Network function mapping

Secret references

Certificate references



Configuration Philosophy

Several unsolved problems introduce toil into our configuration management system.

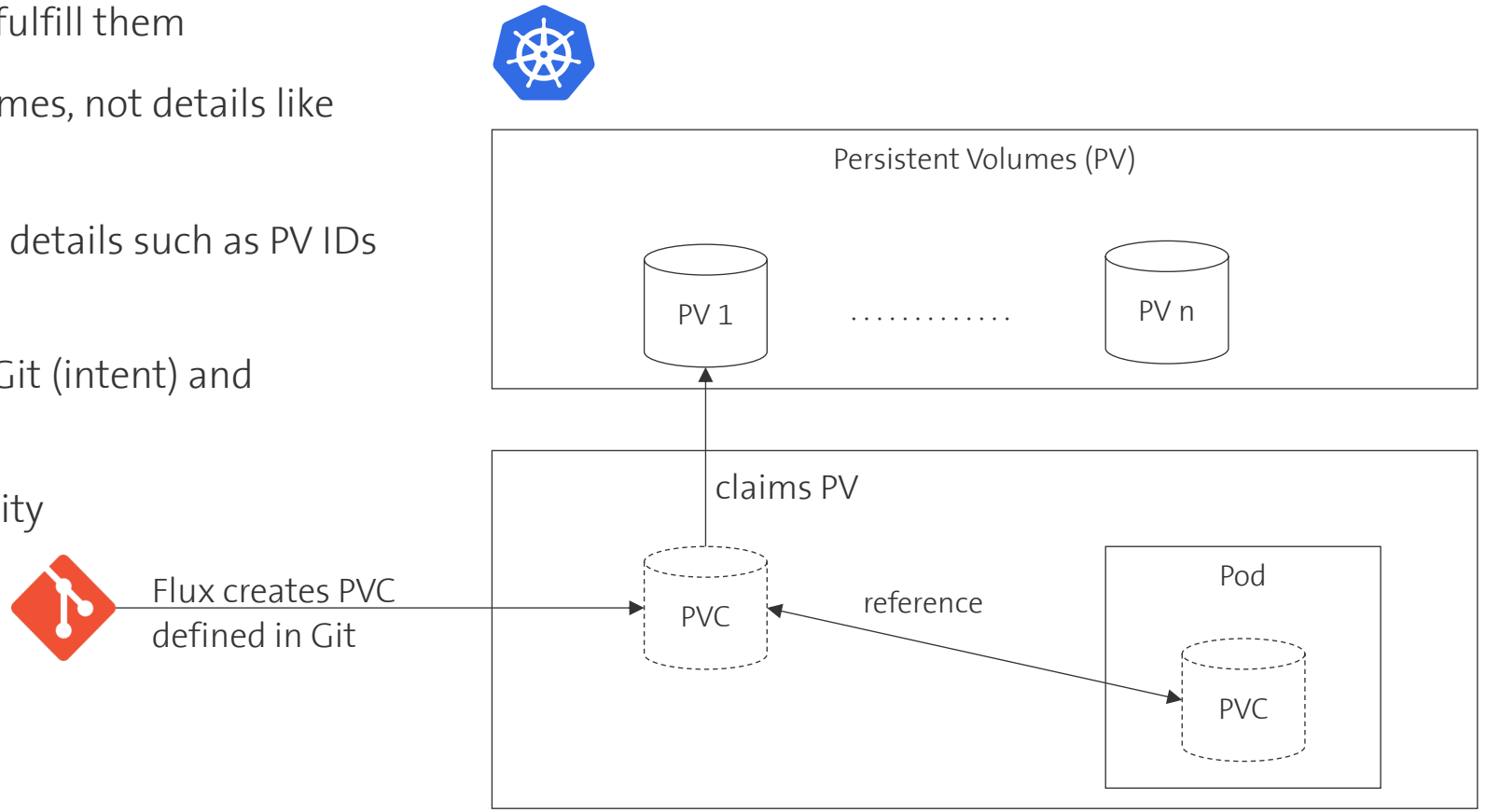
- **Complexity in configurations**
  5G core configurations typically involve thousands of parameters and are thus not user friendly. Abstraction reduces complexity by focusing on essential controls.
- **Redundancies across configurations**
  Repeated parameters across application configuration and helm charts. Abstraction unifies these redundancies into a single source of truth.
  E.g. ASN in a BGP setup needs to be known by both peers.
- **Limited configuration discovery**
  Automatic configuration based on the environment still very weak.
  E.g. Use of status field of a Kubernetes Resource as the input value of a Helm Release.

https://sre.google/workbook/configuration-design

Shared Source of Truth in Kubernetes and GitOps

Kubernetes claim model used in Persistent Volumes (PV) and Persistent Volume Claims (PVC)

- Decouples resource requests from provisioning
- PVCs represents user needs, PVs fulfill them
- Users interact using on claim names, not details like volume IDs
- Only the claims are stored in Git, details such as PV IDs are stored in Kubernetes
- Source of truth is split between Git (intent) and Kubernetes (dynamic state)
- Enables abstraction and reusability



Benefits of Abstraction

Reduce Toil

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

Increased Reliability

Ensures consistent configurations across systems, minimizing errors and mismatches.

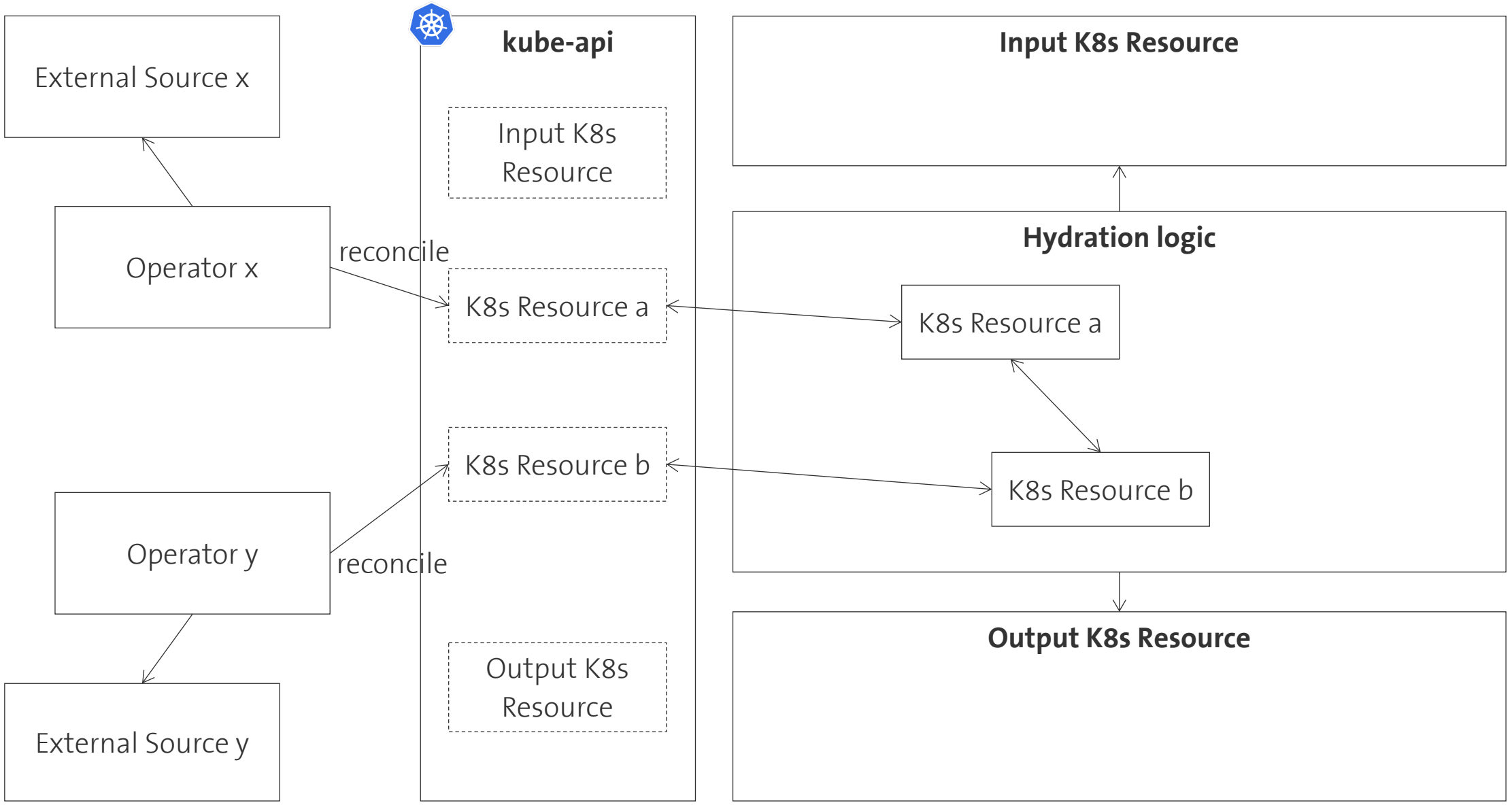
Enhanced Efficiency

Streamlines configuration processes, saving time and resources.

Scalability

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

Flow of Config Abstraction



What Are Our Requirements?

KRM Based

Input and Output can be KRM resources

Garbage Collection

Resources belonging together are lifecycled together

Output Flexibility

Generated resources stored with various backends

Reconciliation

Output is updated as Input is changed

Functions

Ability to define helper functions (e.g. string manipulation)

Gradual Adoption

Migration from static to dynamic taken at own pace.

But What About Helm, Kustomize, Argo and Flux?

Limited Dynamic Assembly

We cannot use live Kubernetes resources as source of information for e.g. a helm release

No Custom Functions

We cannot invoke arbitrary code into Kustomize/Helm templating

Only Partially KRM

Not all Resources involved follow KRM (e.g. Helm Values)

Kubernetes Resource Model (KRM)

API extensions

Custom Resource Definitions extend the Kubernetes API.

e.g. API definition for NetBox PrefixClaim

CRs as Instances

Custom Resources instantiate a CRD.

e.g. NetBox PrefixClaim resource

Business Logic

Use of Operators or templates to run custom logic

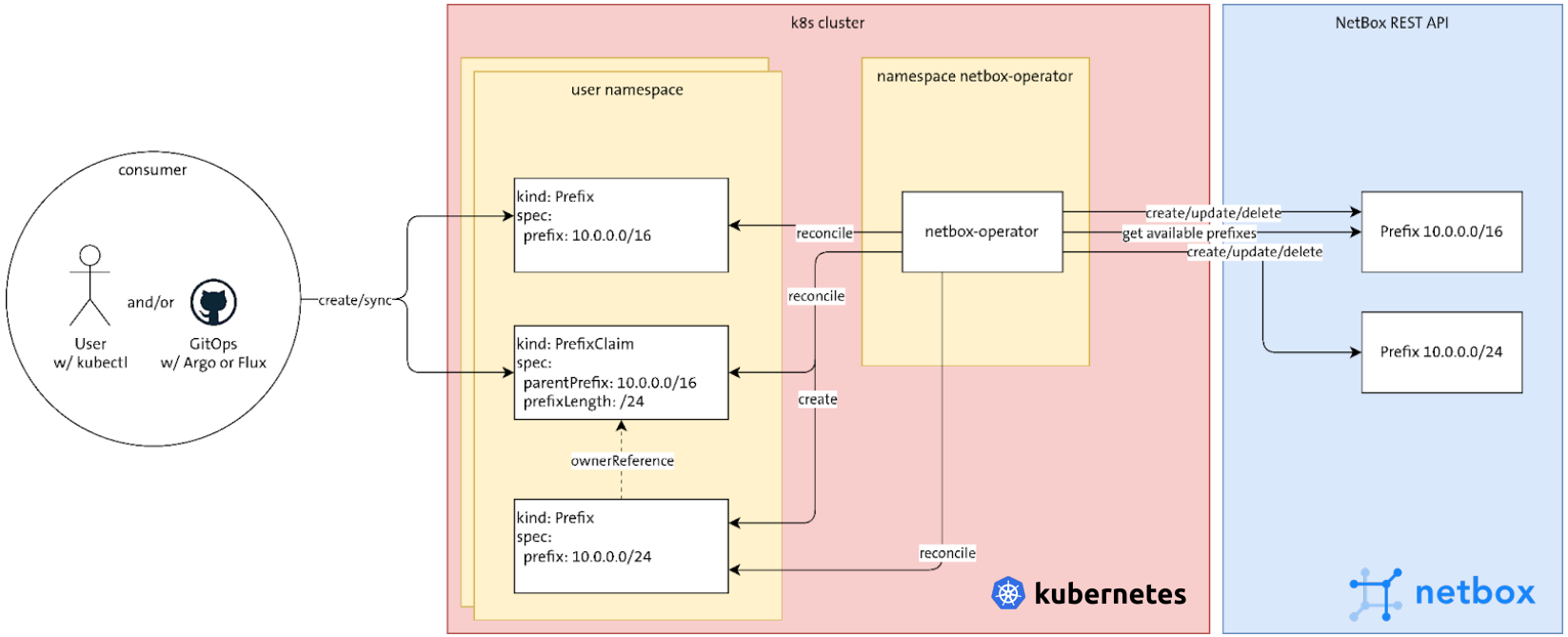
e.g. Assemble a config, Self healing

NetBox Operator – Now Open Source

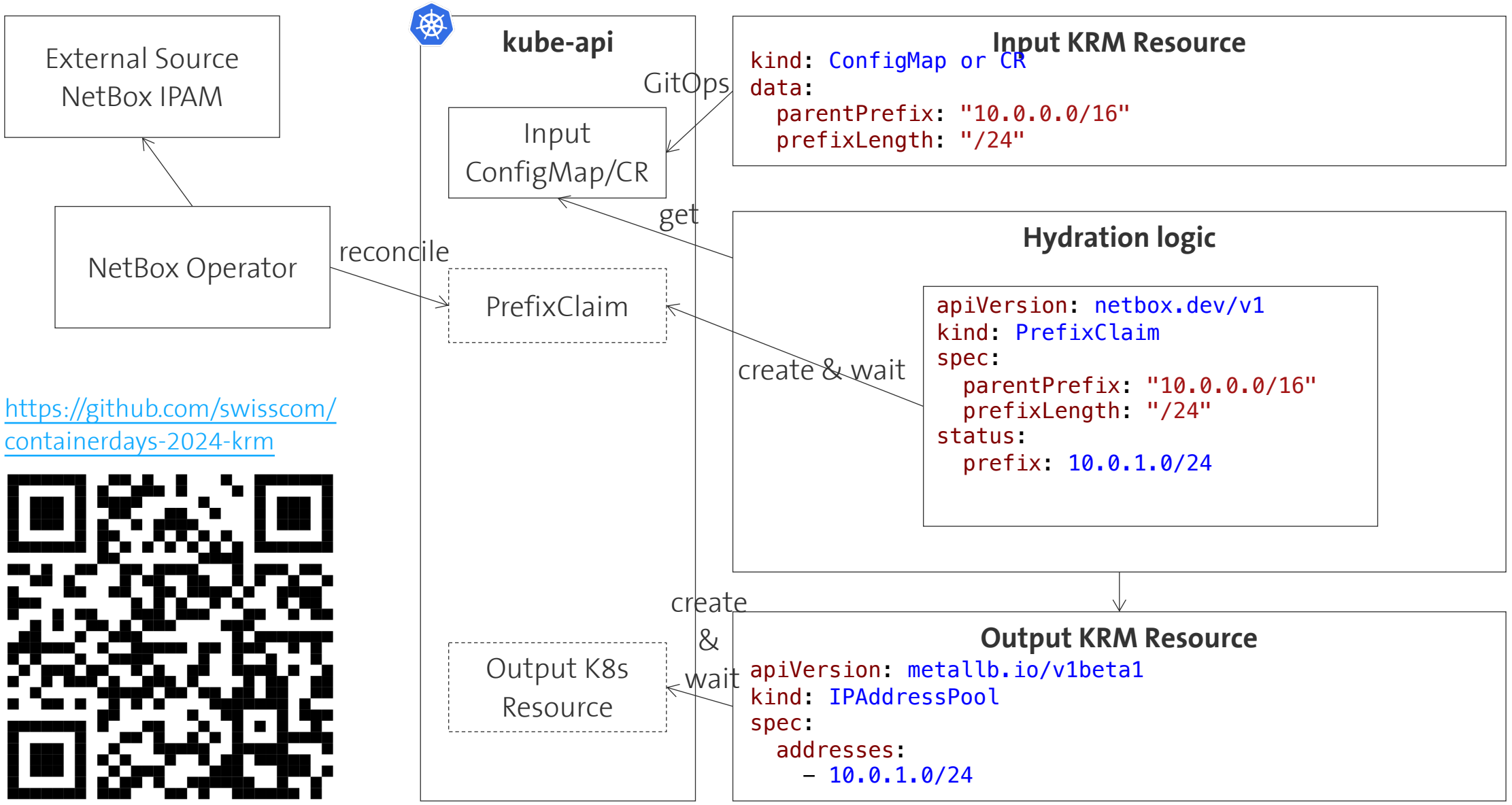
NetBox Operator, an open-source tool designed to integrate NetBox resource management directly into your Kubernetes environment.

https://github.com/netbox-community/netbox-operator

Announcement on www.swisscom.ch



Config Abstraction Demo Example: Dynamically Assign IPs for MetalLB



https://github.com/swisscom/containerdays-2024-krm

Ansible + Jinja2

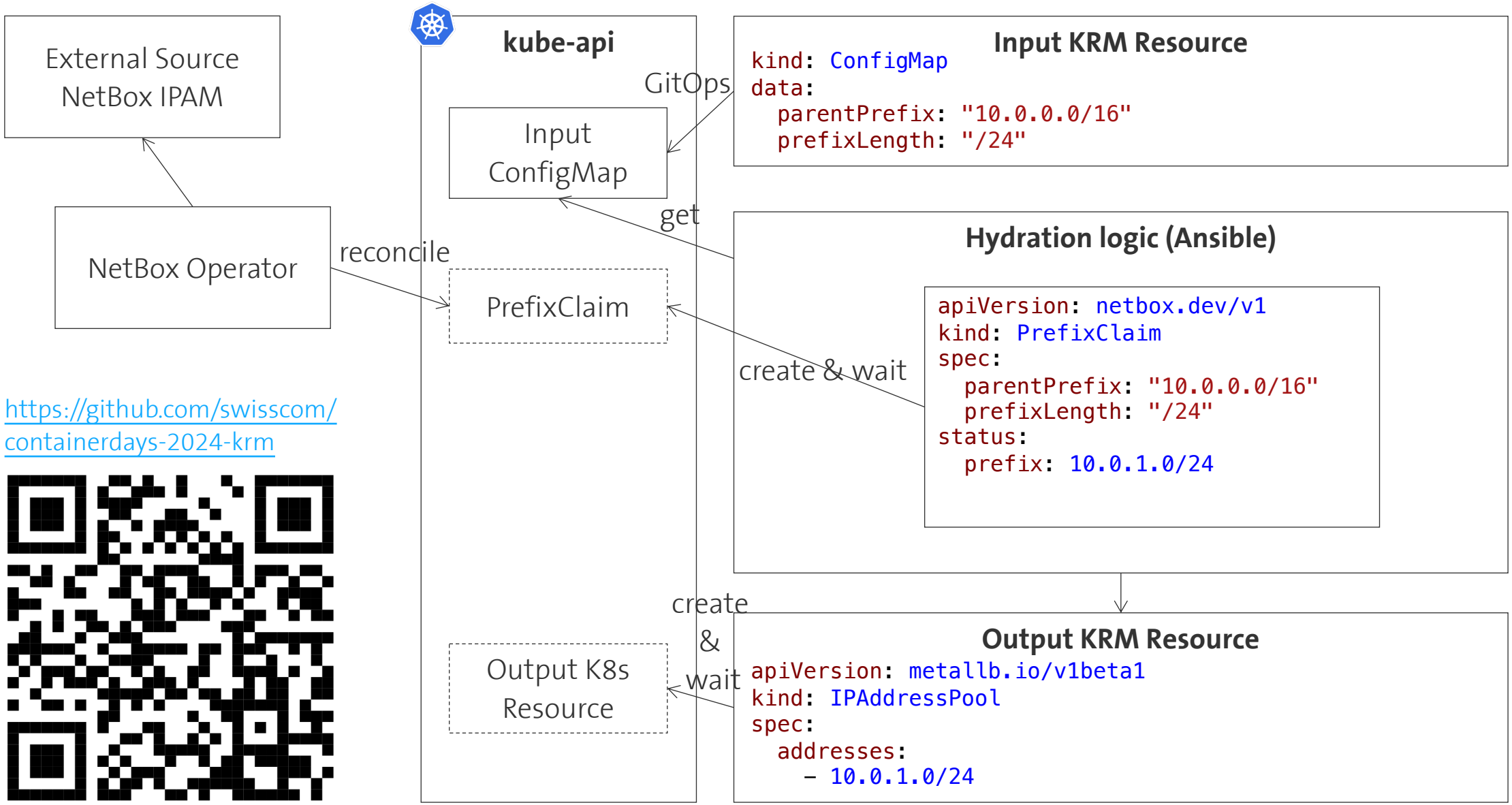
Open-Source automation framework
https://github.com/ansible/ansible (62k Stars, 23k Forks)

Business Logic in Templates and Playbooks
Jinja2 for templating, Playbooks for imperative logic

State Management
Stateless, can use Kubernetes Garbage Collection

No Reconciliation
Just a binary, triggering/execution not solved

Config Abstraction With Ansible + Jinja2



https://github.com/swisscom/containerdays-2024-krm

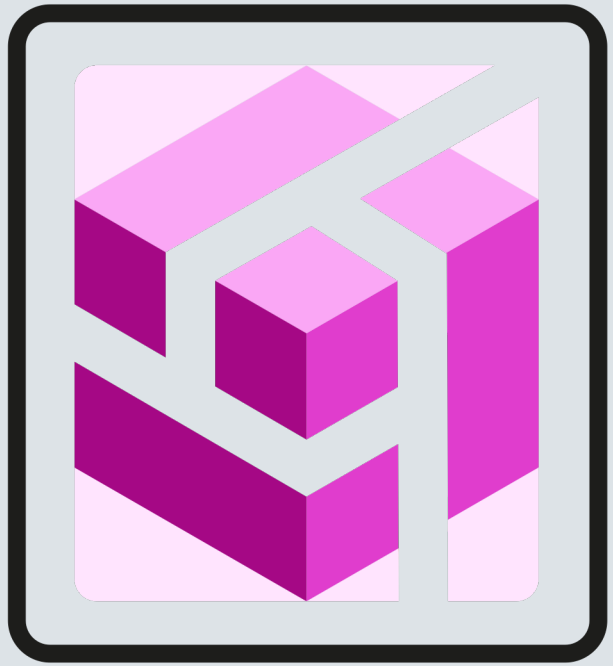
KForm

KRM templating inspired by Terraform
https://github.com/kform-dev/kform (4 Stars, 1 Fork)

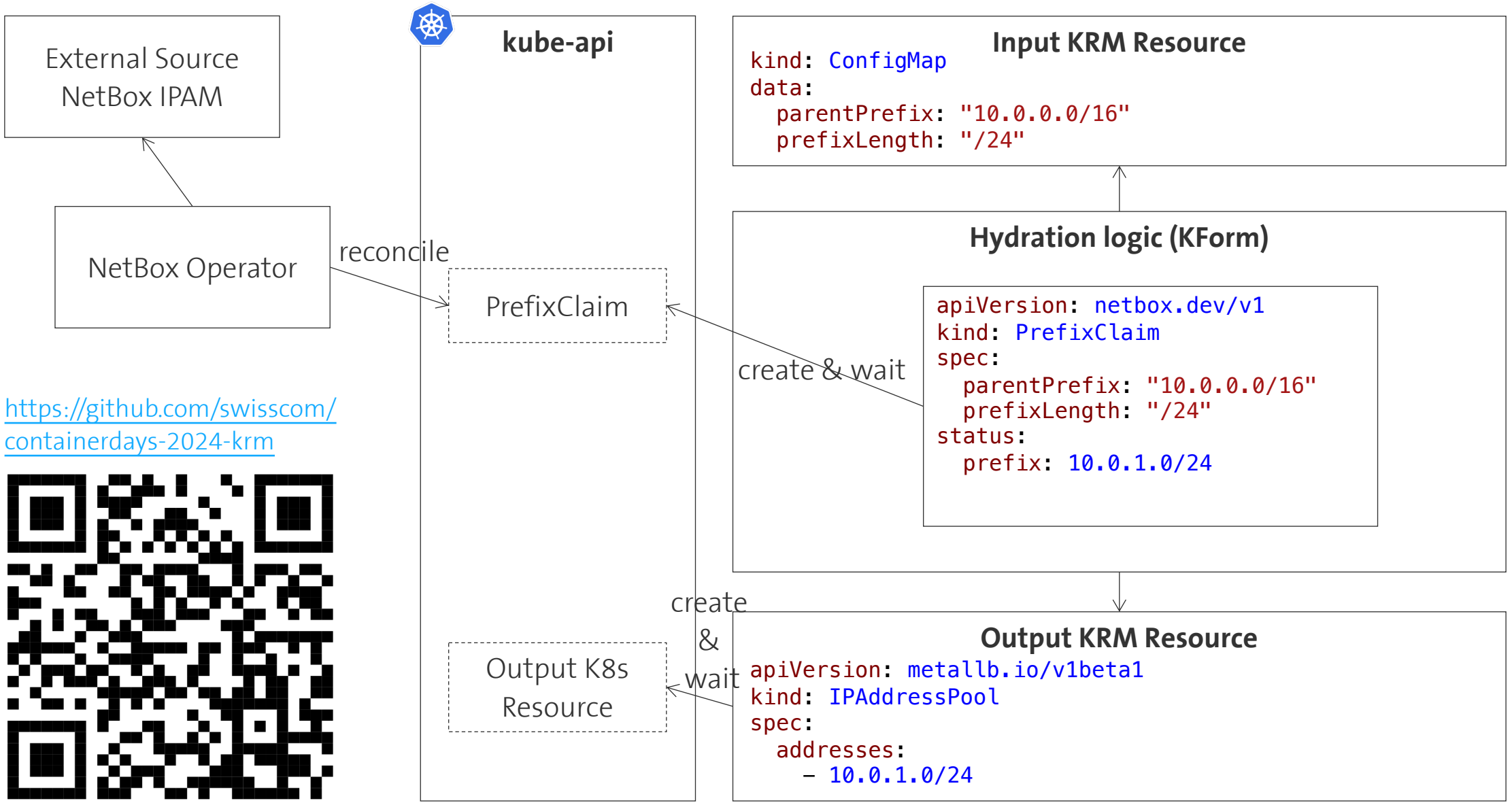
Business Logic with minimum Code
cel-go used for logic within Kubernetes Resources

State Management
Tracks state using ConfigMap in the kube-api

No Reconciliation
Just a binary, triggering/execution not solved



Config Abstraction With KForm



https://github.com/swisscom/containerdays-2024-krm

Kubernetes Operator

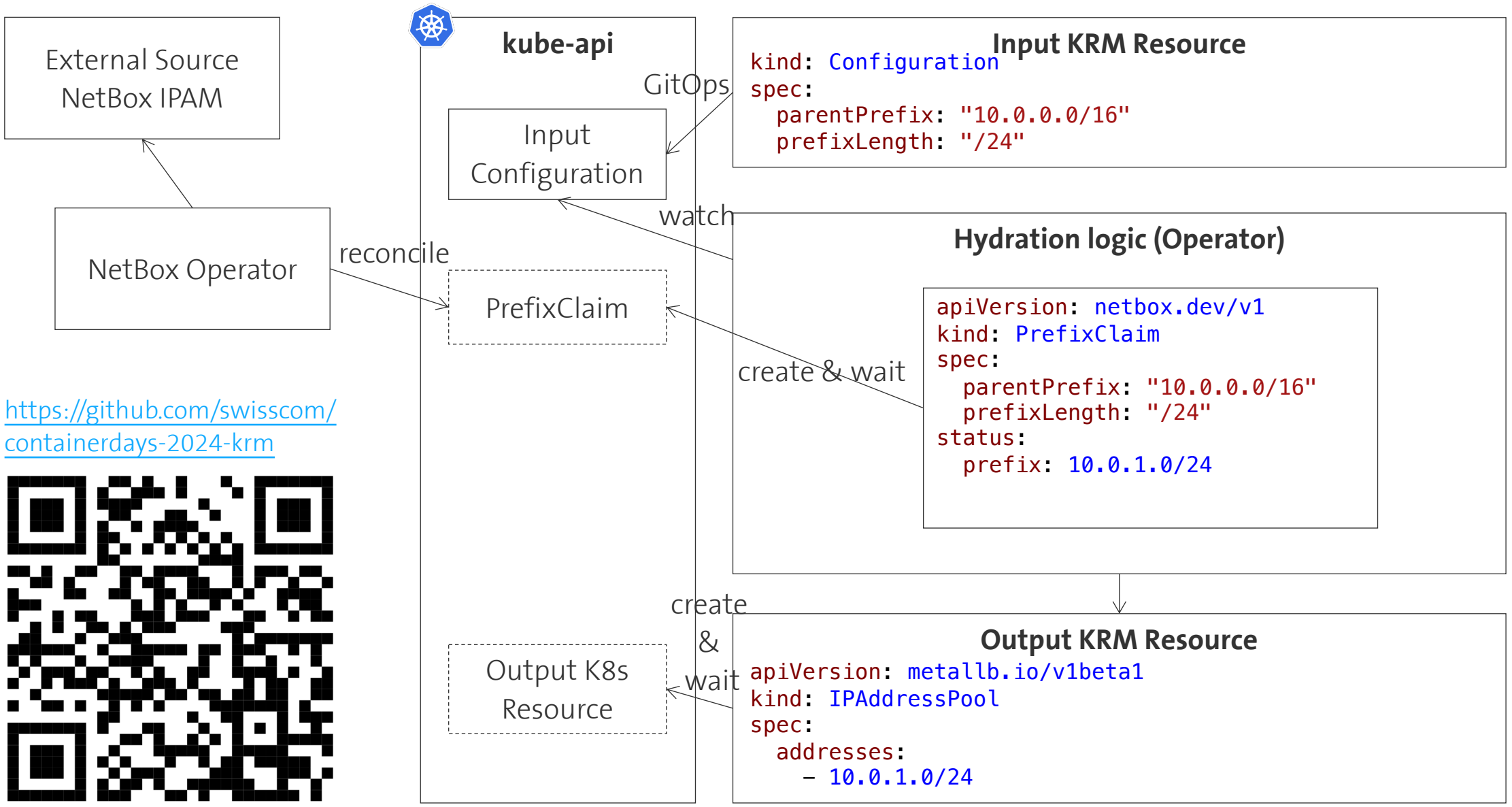
Golang Based Operator scaffolded using Kubebuilder
https://github.com/kubernetes-sigs/kubebuilder

Business Logic in Go Code
Maximum flexibility for imperative logic

State Management
Uses Kubernetes Garbage Collection and Finalizers

Reconciliation
Event driven triggering, constant reconciliation

Config Abstraction With Kubernetes Operator



https://github.com/swisscom/containerdays-2024-krm

Ansible vs. KForm vs. Kubernetes Operator

	Ansible+Jinja2	KForm	Kubernetes Operator
KRM Based	●●○○○	●●●●●	●●●○○
Garbage Collection	●●●●○	●●●○○	●●●●○
Output Flexibility	●●●●●	●●○○○	●●●●●
Reconciliation	●○○○○	●○○○○	●●●●●
Functions	●●●●○	●○○○○	●●●●●
Gradual Adoption	●●●●●	●●●●○	●○○○○

No mature Kubernetes native tool available to dynamically hydrate resources

Key Takeaways

Dynamic Parameter Management

Difficult to dynamically add parameters to Kubernetes resources in GitOps

Tooling Limitations

No mature tools for config hydration with KRM

Existing Solutions' Shortcomings

Ansible & Jinja: Powerful templating, but not designed with K8s in mind

KForm: KRM-based but lacks scalability for multiple instances

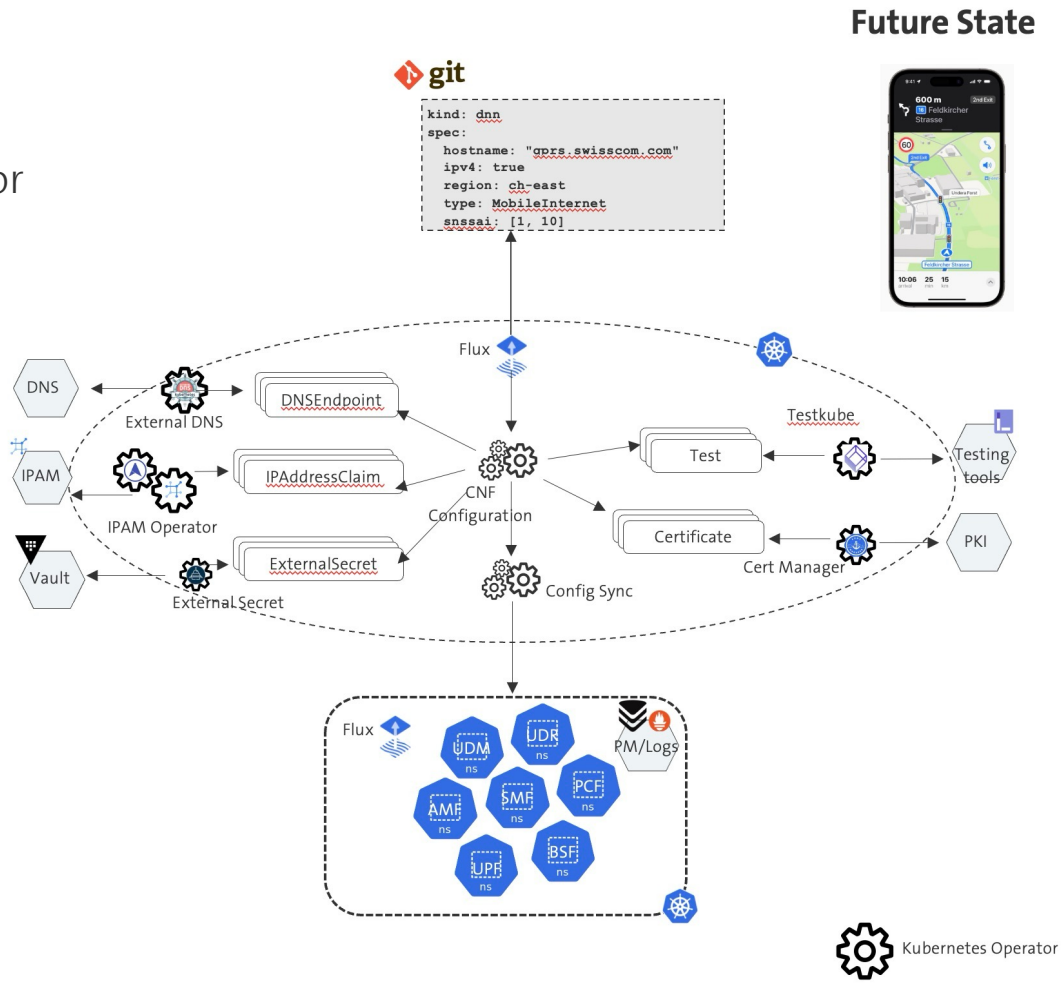
Golang Operator: Too specific, leading to operational overhead

Industry-Wide Relevance:

These challenges are not exclusive to the Telco industry

Next Steps

- **Monitor and Explore:** Track kpt, Kform, and other advancements in Kubernetes management
- **Operator Development:** Implement Kubernetes operators for generic templating
- **Multi-cluster management:** Enhance multi-cluster management and explore cloud-cluster tools (e.g. Nephio, KubeNet)
- **Safe Rollouts:** Ensure safe rollouts with configuration versioning
- **Collaboration:** Engage with open-source community and vendors



Our Learnings & Suggestions

... on configuration abstraction

Avoid

Checking in low level configurations in Git
Things like IP Addresses, VLAN IDs

Complicated configurations
Avoid unnecessary layers of abstraction

Ignore tooling gaps
Avoid using tools like Ansible or Jinja that aren't designed with Kubernetes in mind

Neglect Industry Relevance
Don't assume KRM challenges and solutions are unique to your sector

Aim to

Leverage abstraction
Simplify complex configurations by focusing on essential controls

Reuse cloud native tools to be in-band with K8s
E.g. Flux, Argo, Testkube, cert-manager

Prioritize scalability
Design KRM-based tools with future growth and scaling

Stay Agile
Continuously refine and adapt your KRM practices to evolving needs and technologies

Contribute to the ecosystem
Share your code

Related Talks

Building and Operating a Highly Reliable Cloud Native DNS Service With Open Source Technologies
by Fabian Schulz, Hoang Anh Mai and Joel Studler at ContainerDays 2024
Stage K6, Today 17:30 - 18:05

How We Are Moving from GitOps to Kubernetes Resource Model in 5G Core
by Ashan Senevirathne and Joel Studler at KubeCon Europe 2024
https://www.youtube.com/watch?v=crmTnB6Zwt8

From Spreadsheets to "Everything as Code"
by Joel Studler and Jéremy Weber at ContainerDays 2023
https://www.youtube.com/watch?v=eMAMqW2aLTI

Thanks!

Q&A

GitOps vs. GitOps with KRM

	GitOps	GitOps with KRM
Source of Truth	Git is the single source of truth	Split between Git (intent) and Kubernetes (dynamic state)
Configuration Management	All details are in Git	Git holds high-level configs; Kubernetes manages the rest
Abstraction	Users manage all details	Users focus on intent; Kubernetes abstracts complexity
Flexibility	Less flexible, changes require Git updates	More flexible, Kubernetes adjusts resources dynamically
Scalability	Can be complex to scale	Easier to scale with Kubernetes managing dynamics
Operational Complexity	High; all details tracked in Git	Lower; Kubernetes automates dynamic management