Revolutionizing Network Automation with Kubernetes: Beyond Containers

22-OCT-2024

Wim Henderickx

Introducing Kubernetes

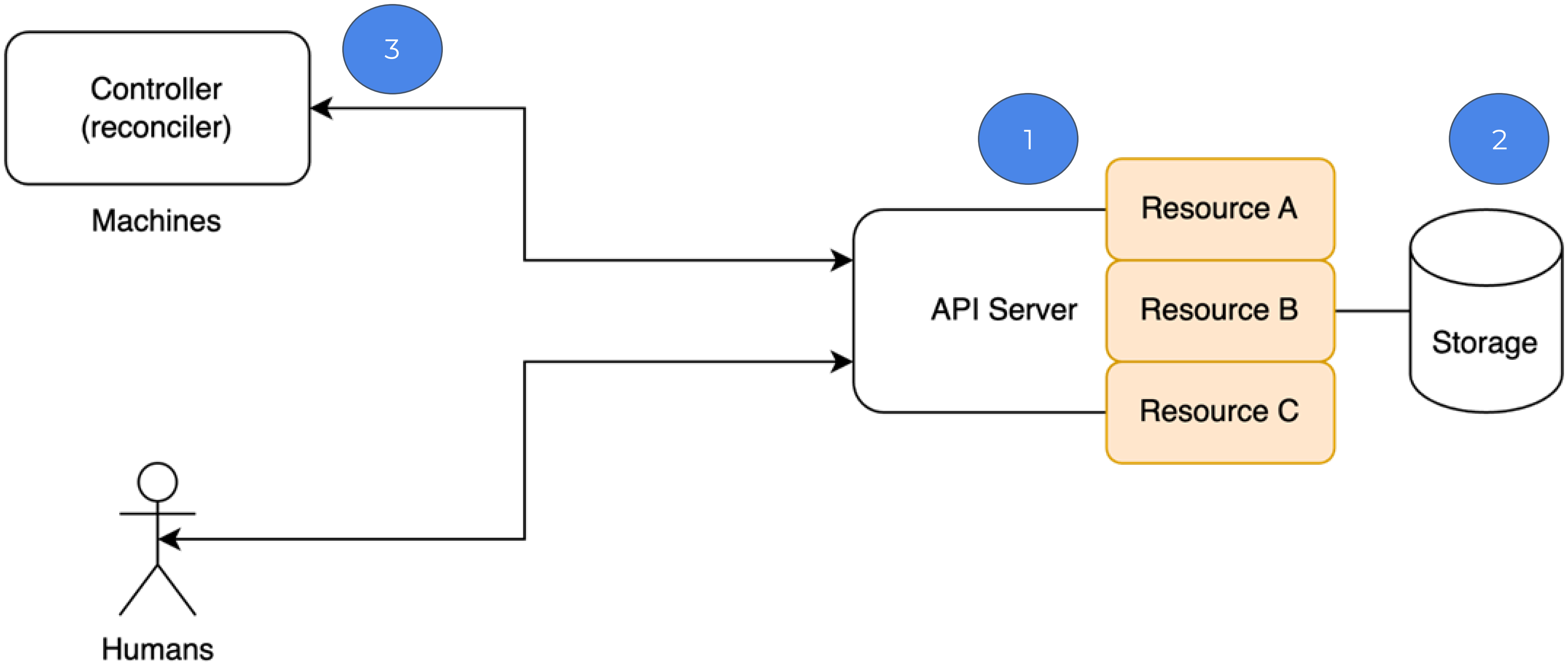
Kubernetes

Original use case:
Container Orchestration

Machinery:
Declarative automation framework

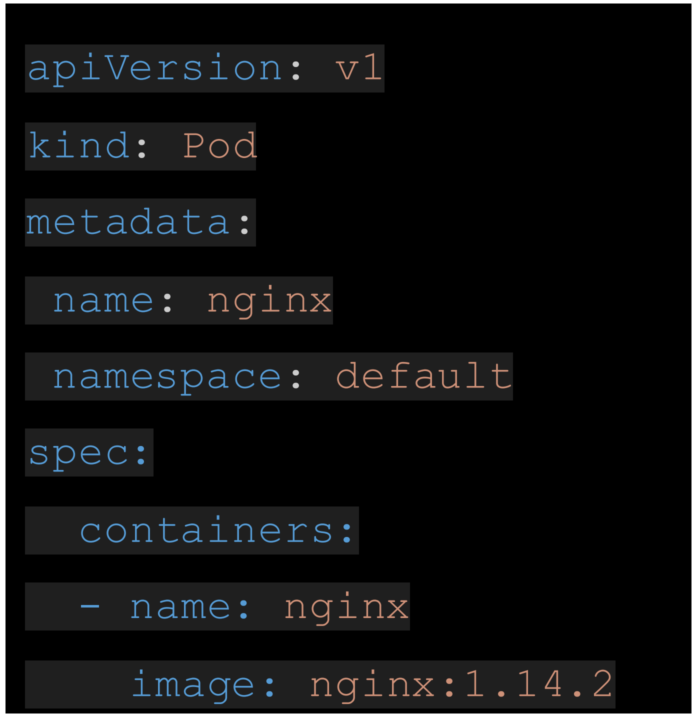
Different ways to look at kubernetes -> in this session we focus on the automation machinery

Kubernetes Building Blocks



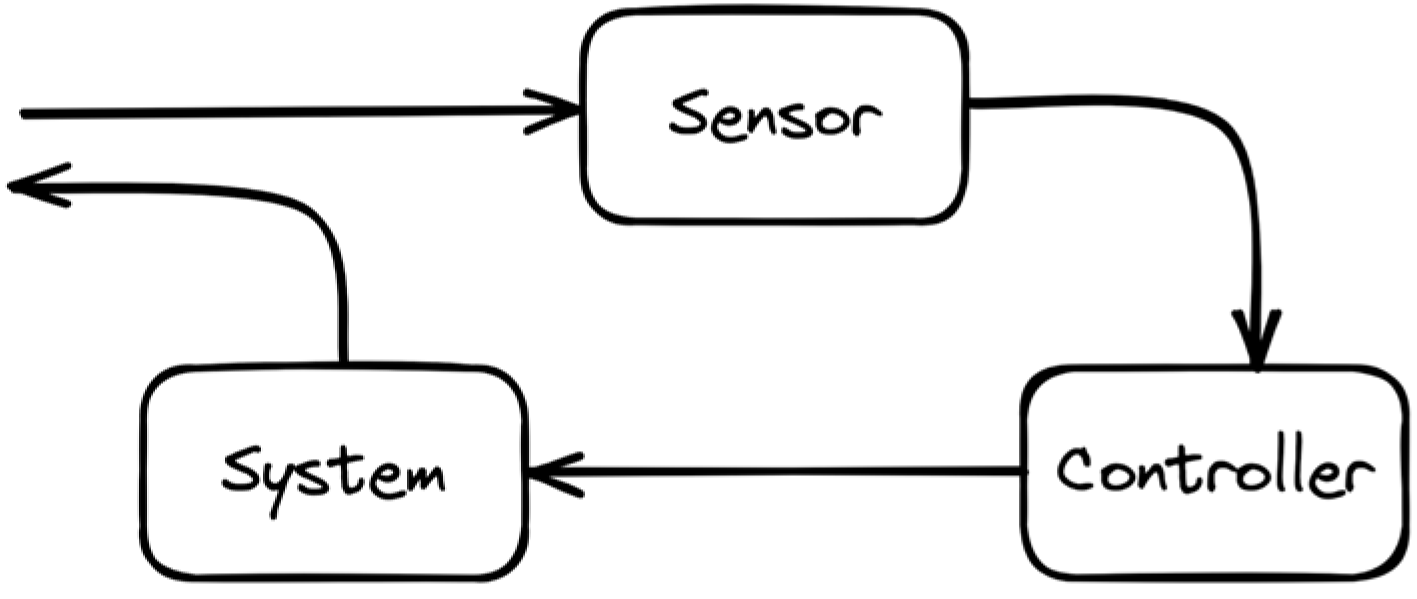
Kubernetes Resource Model (KRM)

- API resources
  - Group, Version, Kind
  - Metadata
- Declarative API Operations
  - Desired state
  - Observed state
- Extendable (CRD)
- Event Driven (On Change)
- Manipulate at REST



```yaml
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 namespace: default
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
```

Kubernetes Controller/Reconciler



closed loop

NANOG

Kubernetes Automation Engine

Technical

- API centric
- Language agnostic
- Declarative
- Extendable
- Event Driven (On Change)
- Continuous reconciliation

Non technical

- Vendor agnostic
- Use case agnostic
- Cross-platform
- Open source
- Wide community
- Knowledge base

Why would we not use kubernetes for network automation ? YAML ?

NANOG™

Let’s compare automation machinery for containers with networking

What did Kubernetes do?

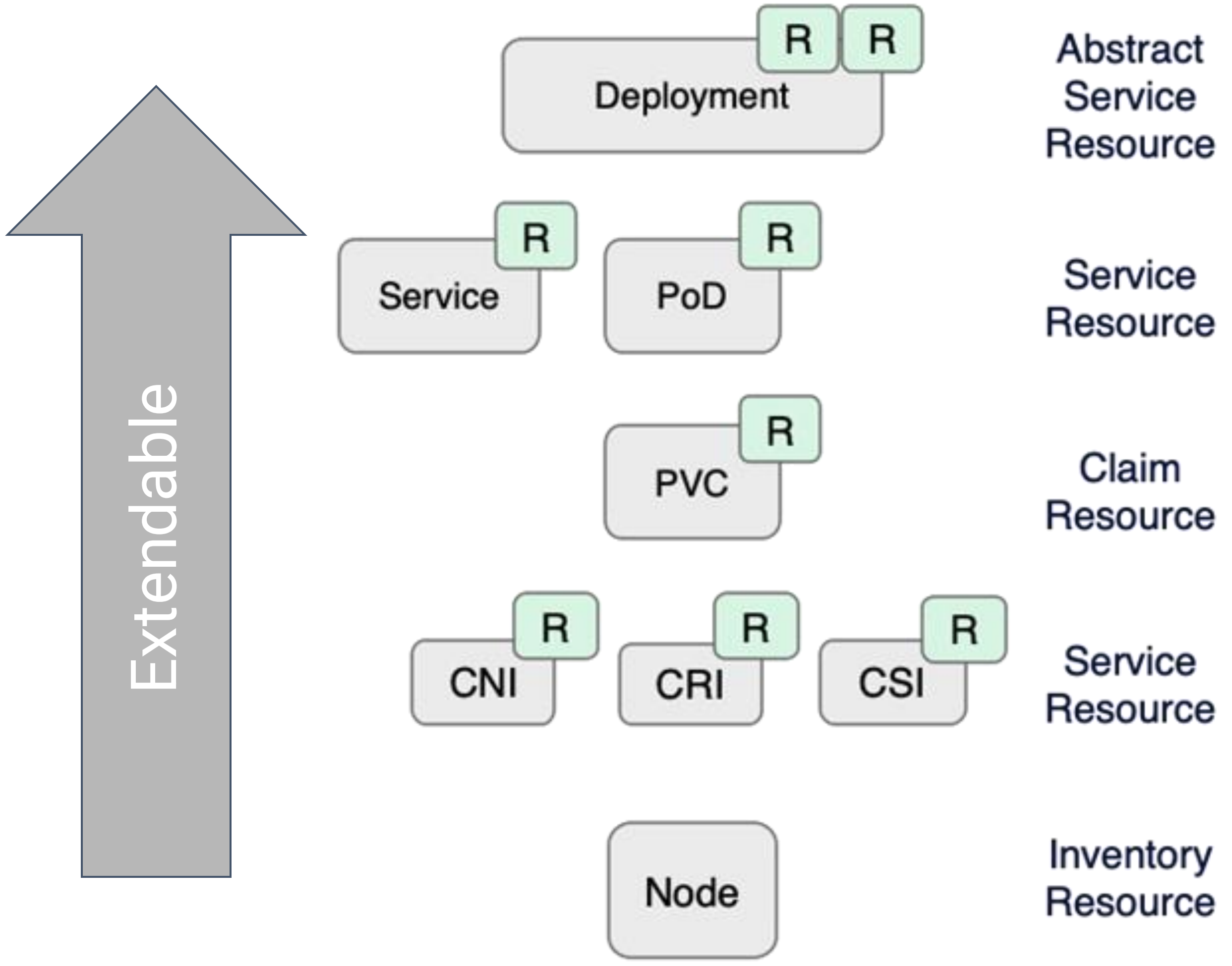
Abstraction

Segment the problem space

Reusable Building blocks

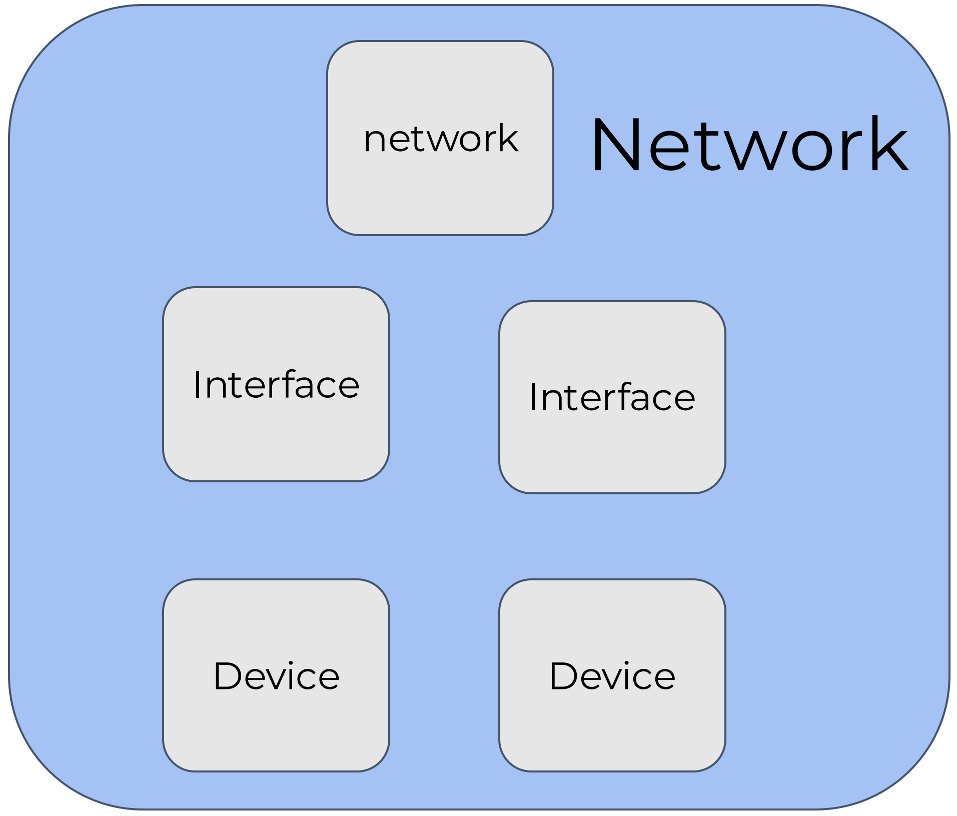
Declarative

Continuous Reconcile
(On Change/Event Driven)

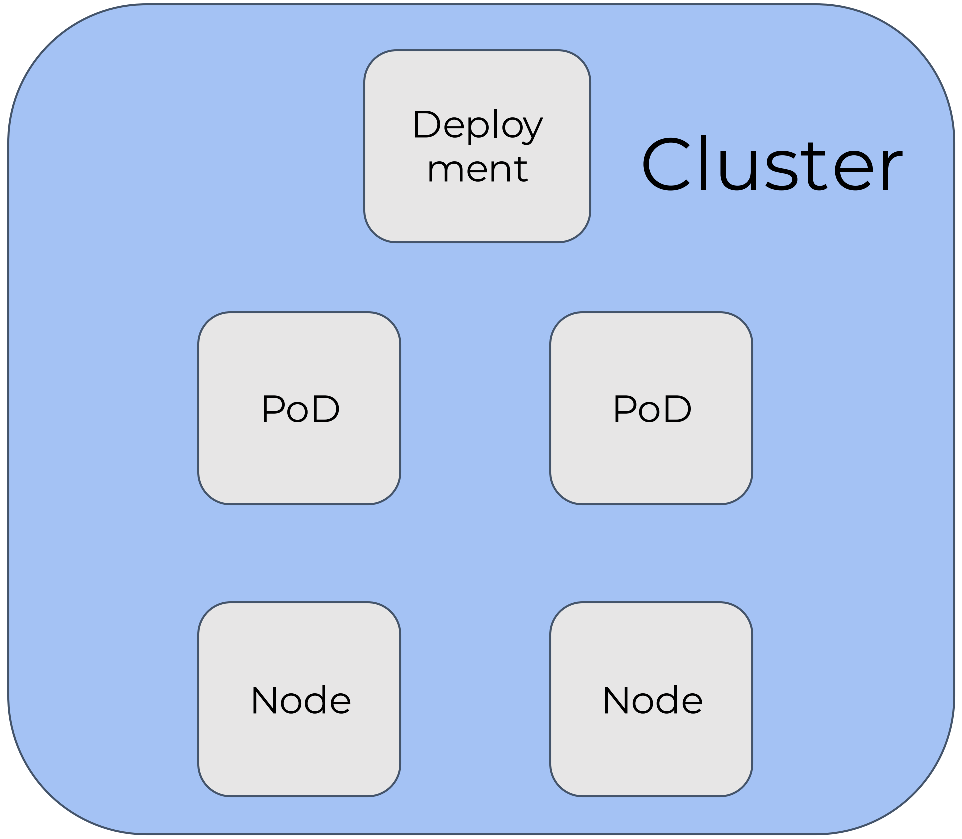


NANOG

Comparing network automation with k8s



Networking



Kubernetes

Differences

- Use case
- Protocols
- API(s)
- Skills

Let’s see how we can apply kubernetes principles to network automation

Introducing Kubenet

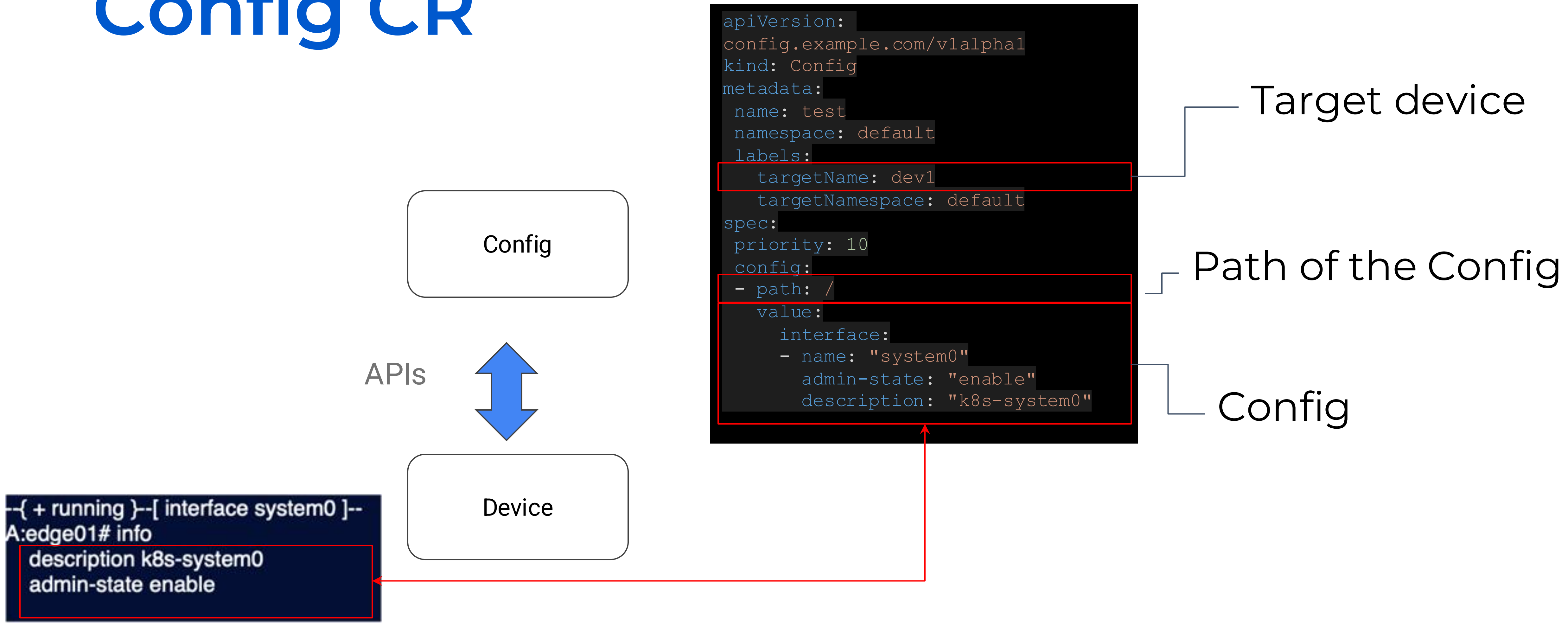
Initiative focussed to help network engineers understand the potential of kubernetes for network automation

Open source projects supporting the initiative

NANOG™

How to use a k8s API towards a network device?

Config CR

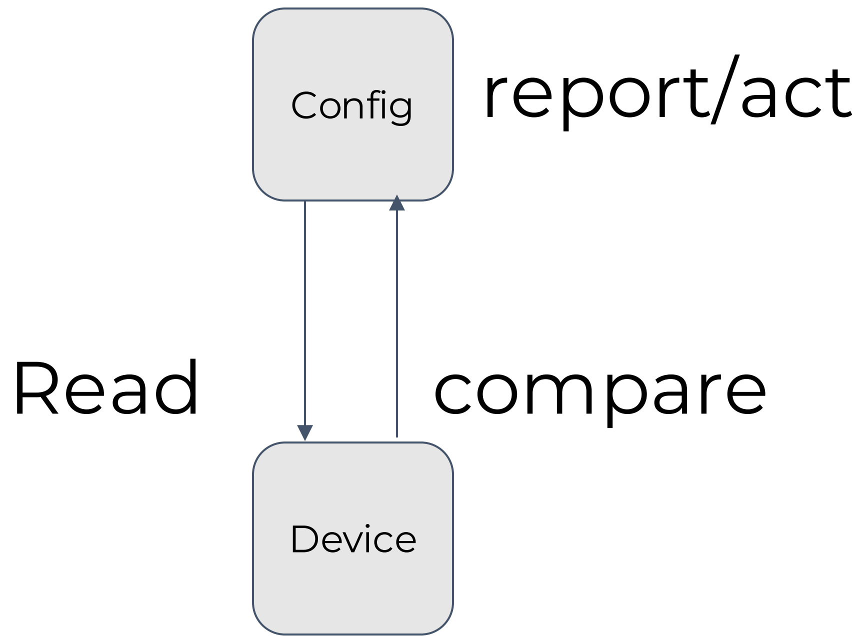


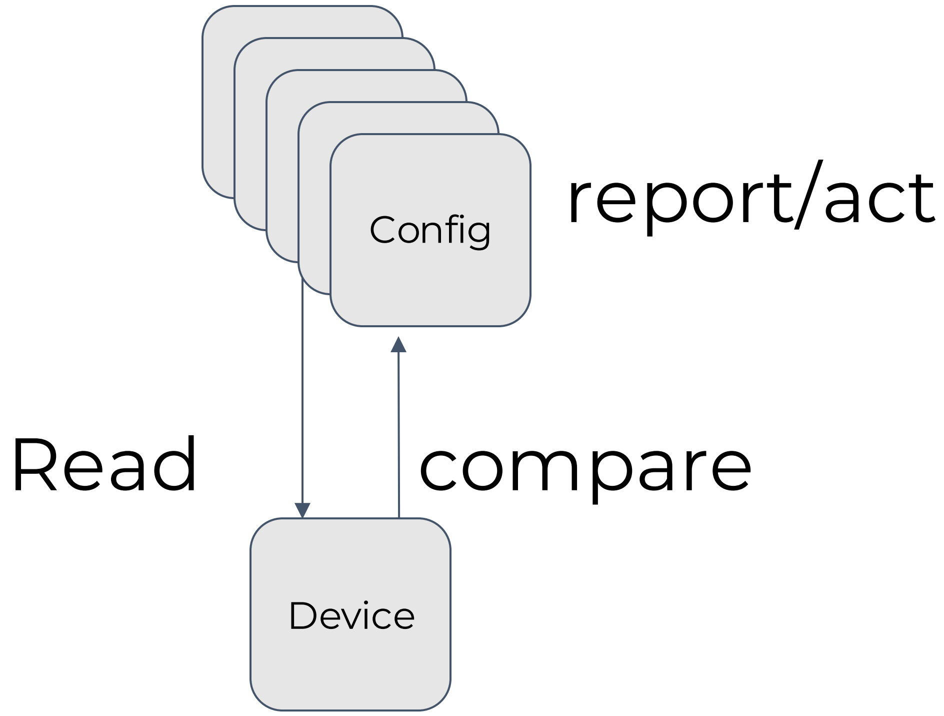
```
apiVersion:
config.example.com/v1alpha1
kind: Config
metadata:
 name: test
 namespace: default
 labels:
   targetName: dev1
   targetNamespace: default
spec:
 priority: 10
 config:
 - path: /
   value:
     interface:
     - name: "system0"
       admin-state: "enable"
       description: "k8s-system0"
```

```
--{ + running }--[ interface system0 ]--
A:edge01# info
    description k8s-system0
    admin-state enable
```

NANOG™

How to scale declarative operations?



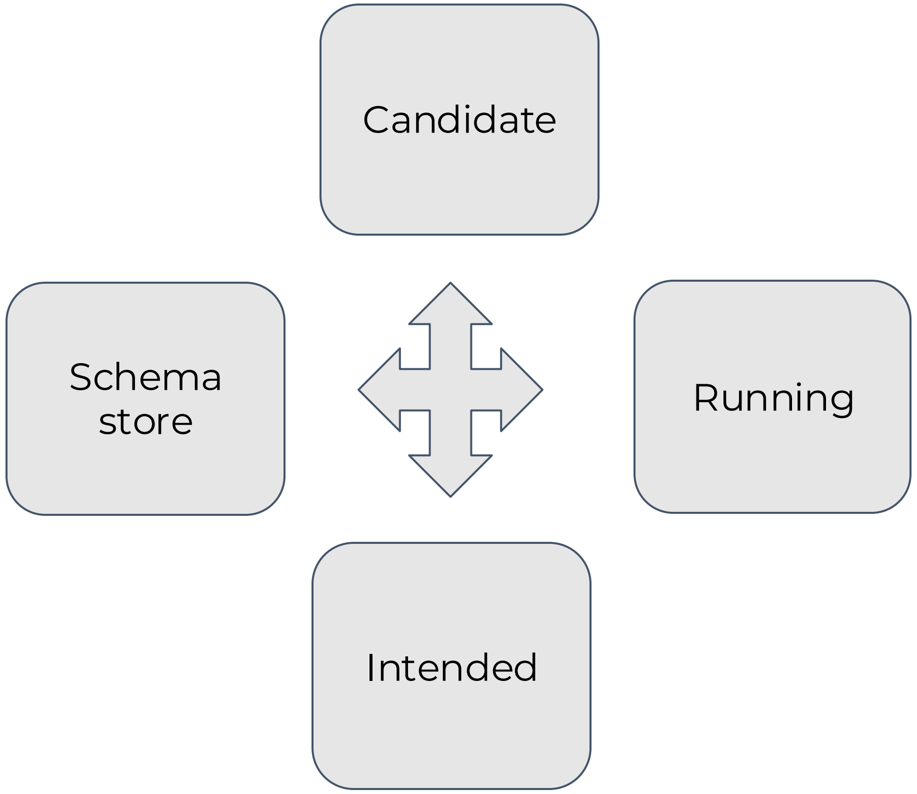


will the device hold up with the API calls ?

NANOG™

Vendor agnostic

- YANG helps given the semantics are standardized
- A unified YANG model helps but is not mandatory
- A schema store with an API
- A Candidate change management
  - Validate change
  - Handles dry-run

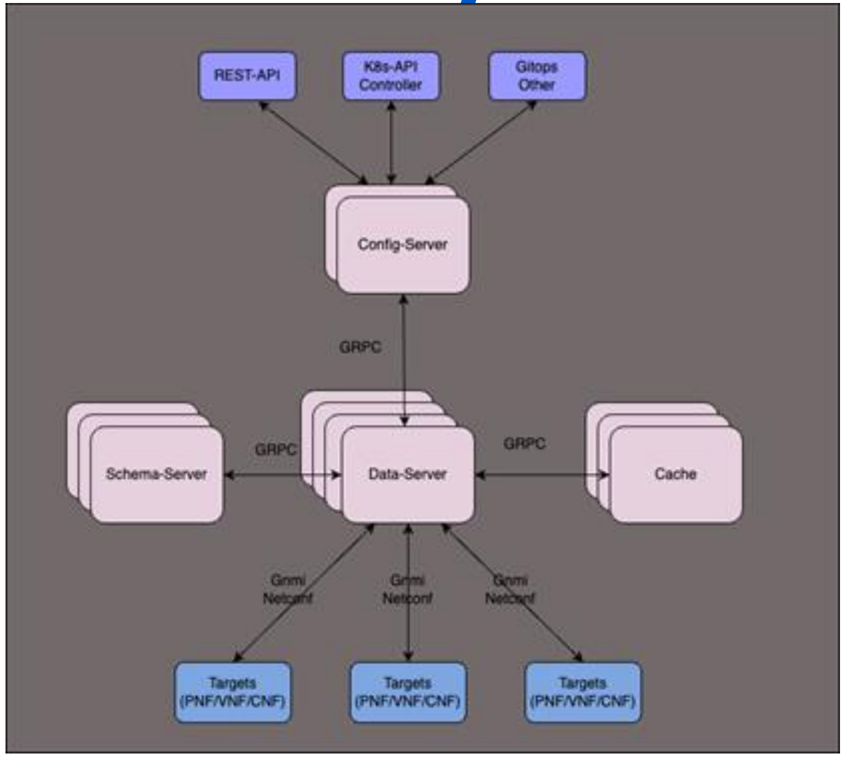


NANOG

SDC

Introducing SDC (Schema Driven Configuration)

- Open source
- Multi-vendor
- PNF, VNF, CNF, NooP
- Config and state
- YANG focussed
- gNMI and Netconf
- Config and config snippets
- Declarative
- Conflict management



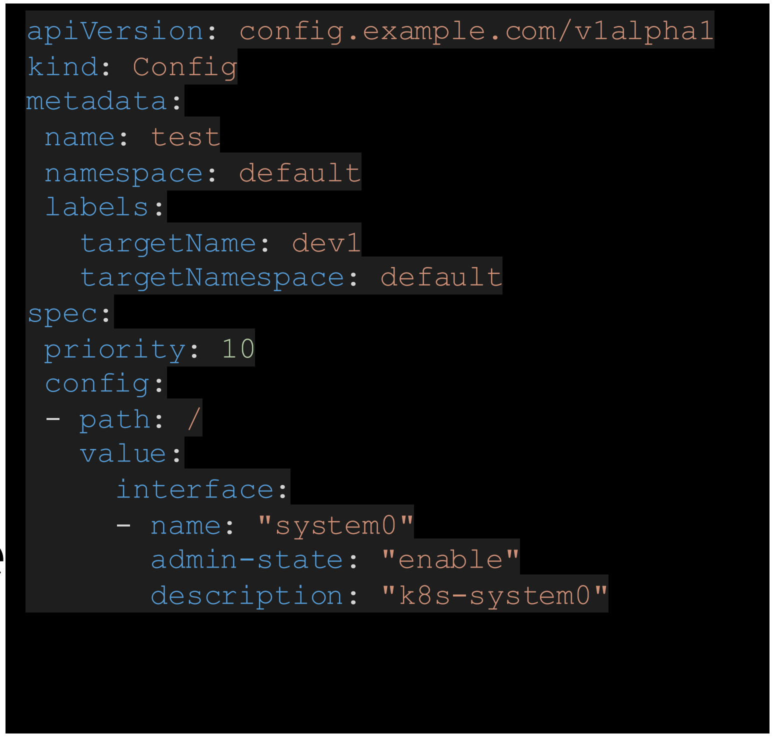
https://docs.sdcio.dev/

SDC enables YANG based system to be consumed from kubernetes API

SDC KRM resources

Config CR
- declarative device configuration

ConfigSet CR
- declarative device configuration across multiple devices



```yaml
apiVersion: config.example.com/v1alpha1
kind: Config
metadata:
 name: test
 namespace: default
 labels:
   targetName: dev1
   targetNamespace: default
spec:
 priority: 10
 config:
 - path: /
   value:
     interface:
     - name: "system0"
       admin-state: "enable"
       description: "k8s-system0"
```

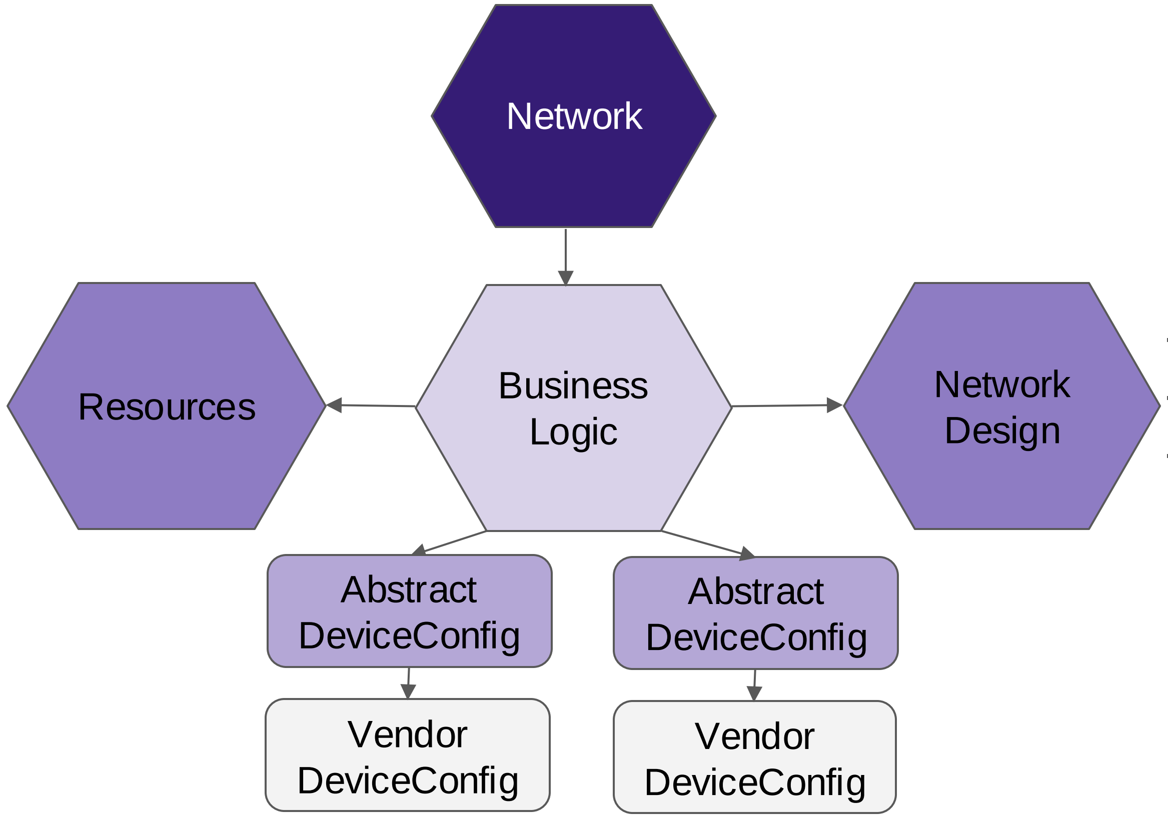
Running Config
- get the running config
- backup/restore

Unmanaged Config
- get config not managed through a config CR

Given we now have an ability to interact with a network device using K8s API, how about automating a network across multiple devices?

What did we do in Kubenet?

- Node
- Link
- IP
- VLAN
- AS



- Addressing
- Protocols
- Encapsulation

Segment the problem space in resources (abstractions) and reconcilers (business logic)

NANOG™

Resources in Kubenet

Node	IPIndex	Network	Interface	PrefixSet
Link	IPClaim	Topology	SubInterface	BFD
Port	ASIndex	NetworkDesign	NetworkInstance	ISIS
Endpoint	ASClaim	NodeTemplate	BGP	OSPF
Adaptor	VLANIndex	...	BGPNeighbor	ACL
Module	VLANClaim	...	BGPDynNeighbor	...
ModuleBay	...	...	RoutingPolicy	...

Resource Examples

Default network

```yaml
apiVersion: network.kubenet.dev/v1alpha1
kind: Network
metadata:
  name: topo3nodesrl.default
spec:
  topology: topo3nodesrl
```

Routed Network

```yaml
apiVersion: network.kubenet.dev/v1alpha1
kind: Network
metadata:
  name: topo3nodesrl.vpc2
spec:
  topology: topo3nodesrl
  routingTables:
  - name: rt20
    networkID: 20
    interfaces:
    - endpoint: e1-1
      node: edge01
      region: region1
      site: site1
      addresses:
      - address: 10.1.1.1/24
    - endpoint: e1-1
      node: edge02
      region: region1
      site: site1
      addresses:
      - address: 10.2.2.1/24
```

Network Design

```yaml
apiVersion: network.kubenet.dev/v1alpha1
kind: NetworkDesign
metadata:
  name: kubenet
  namespace: default
spec:
  interfaces:
    loopback:
      addressing: dualstack
      prefixes:
      - prefix: 10.0.0.0/16
      - prefix: 1000::/64
    underlay:
      addressing: dualstack
      prefixes:
      - prefix: 192.0.0.0/16
      - prefix: 1192::/56
  protocols:
    ibgp:
      as: 65535
      localAS: true
      routeReflectors:
      - topo3nodesrl.default.core01.ipv4
    ebgp:
      asPool: 65000-65100
      bfd: true
    bgpEVPN: {}
  encapsulation:
    vxlan: {}
```

NANOG™

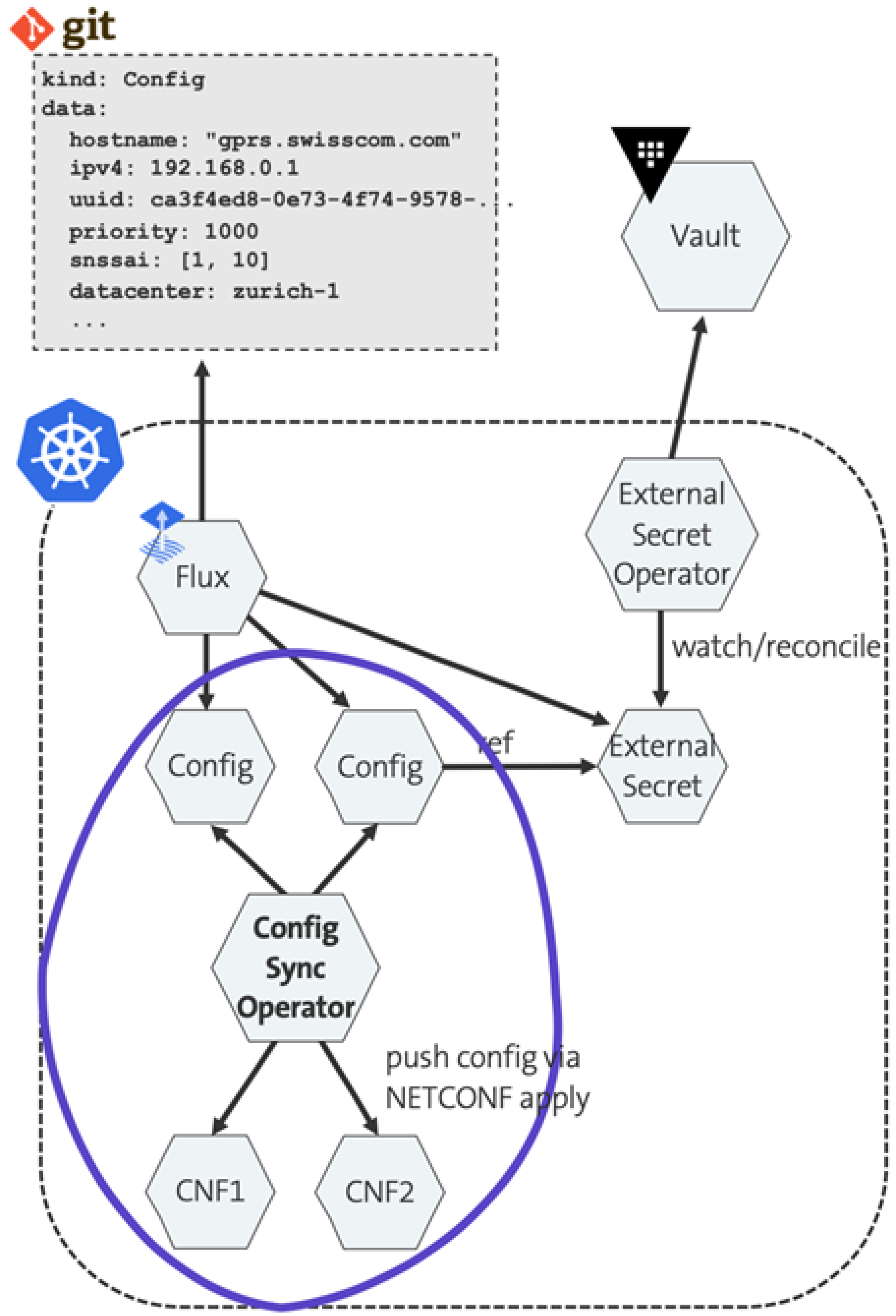
As a network automation engineer you can use existing resources, modify existing resources or create your own resources.

Creating/Updating a resource is easy, the modeling is the hard part.

Let’s look at use cases

Gitops

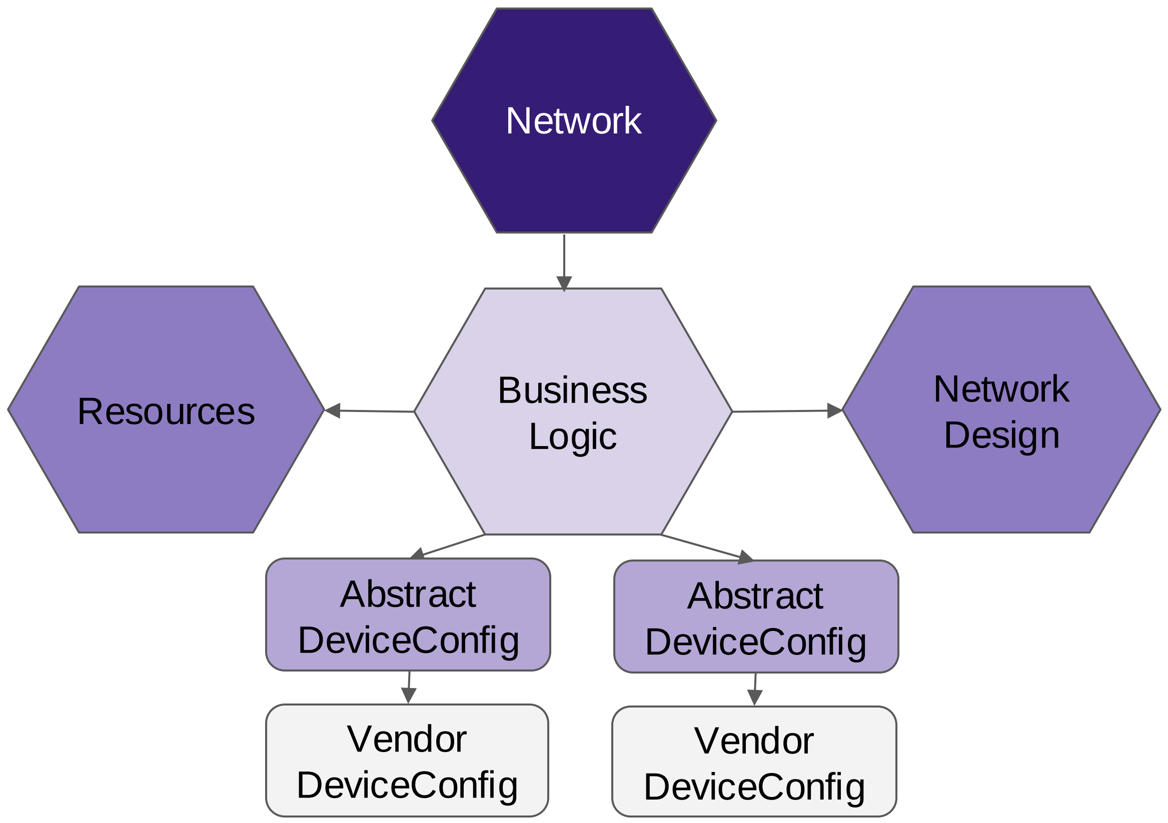
NANOG™



```
kind: Config
data:
  hostname: "gprs.swisscom.com"
  ipv4: 192.168.0.1
  uuid: ca3f4ed8-0e73-4f74-9578-...
  priority: 1000
  snssai: [1, 10]
  datacenter: zurich-1
  ...
```

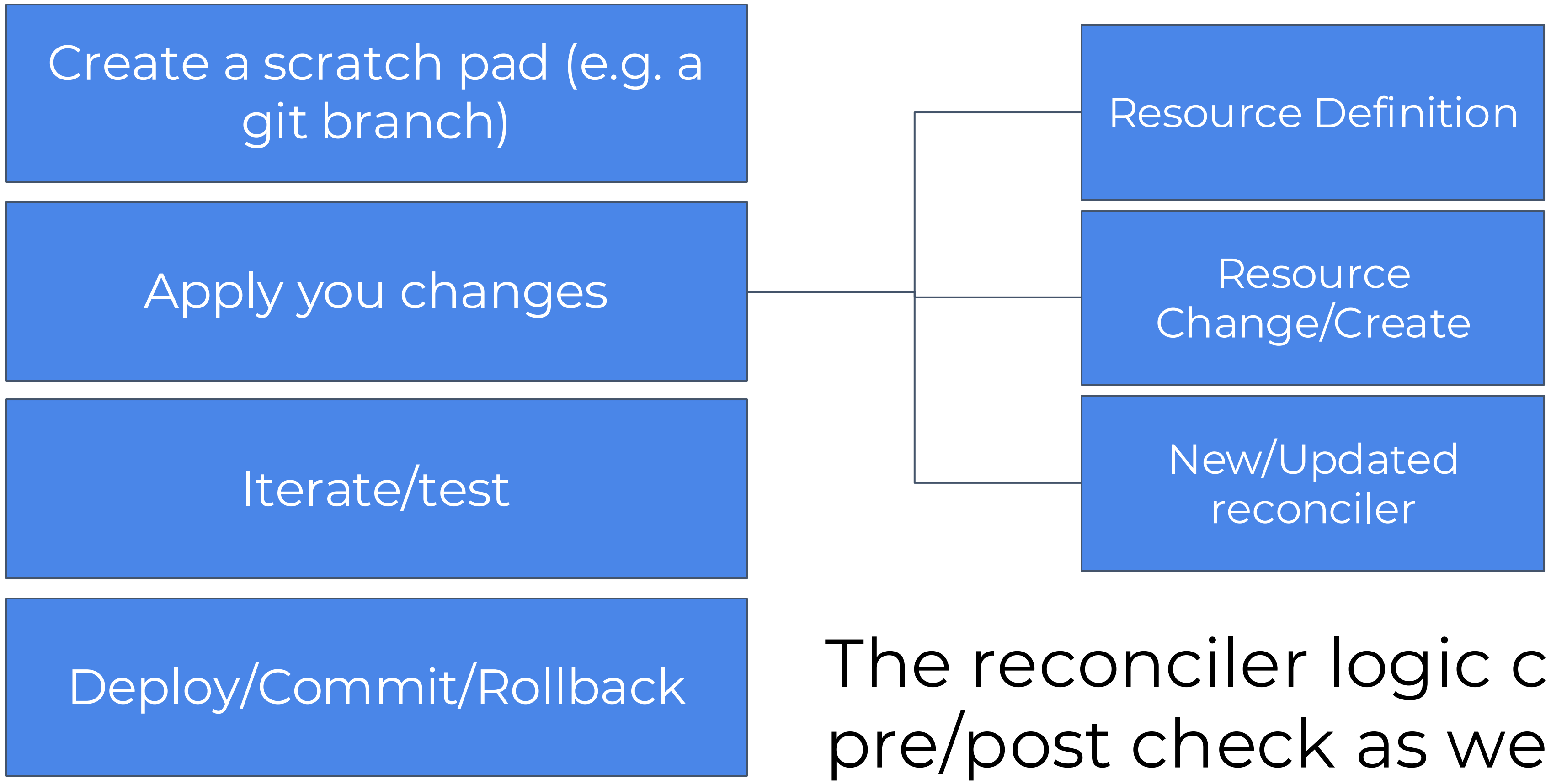
https://sched.co/1YeN2

From abstract to device specific config



Translating the abstract configuration to a vendor specific device configuration can be done using the reconciler logic (Jinja templates/Go Templates, python, ...)

Change management



The reconciler logic can be used for pre/post check as well as rollouts (progressive/network wide transactions/other), etc

Operational tasks

- Backup and restore
- Audits
- Compliance checks
- Closed loop tasks: remediations, mitigations
- OS Upgrades
- Manage Certificates
- Zero touch provisioning
- …

Kubernetes machinery is very powerful, let’s not ignore it for network automation!

Want to learn more, join US

Kubenet

NANOG™

Thank you

22-OCT-2024

NANOG™