GitOps and Continuous Delivery at scale

Lessons learnt

Mario Manno and Juan Herrera Utande - SUSE

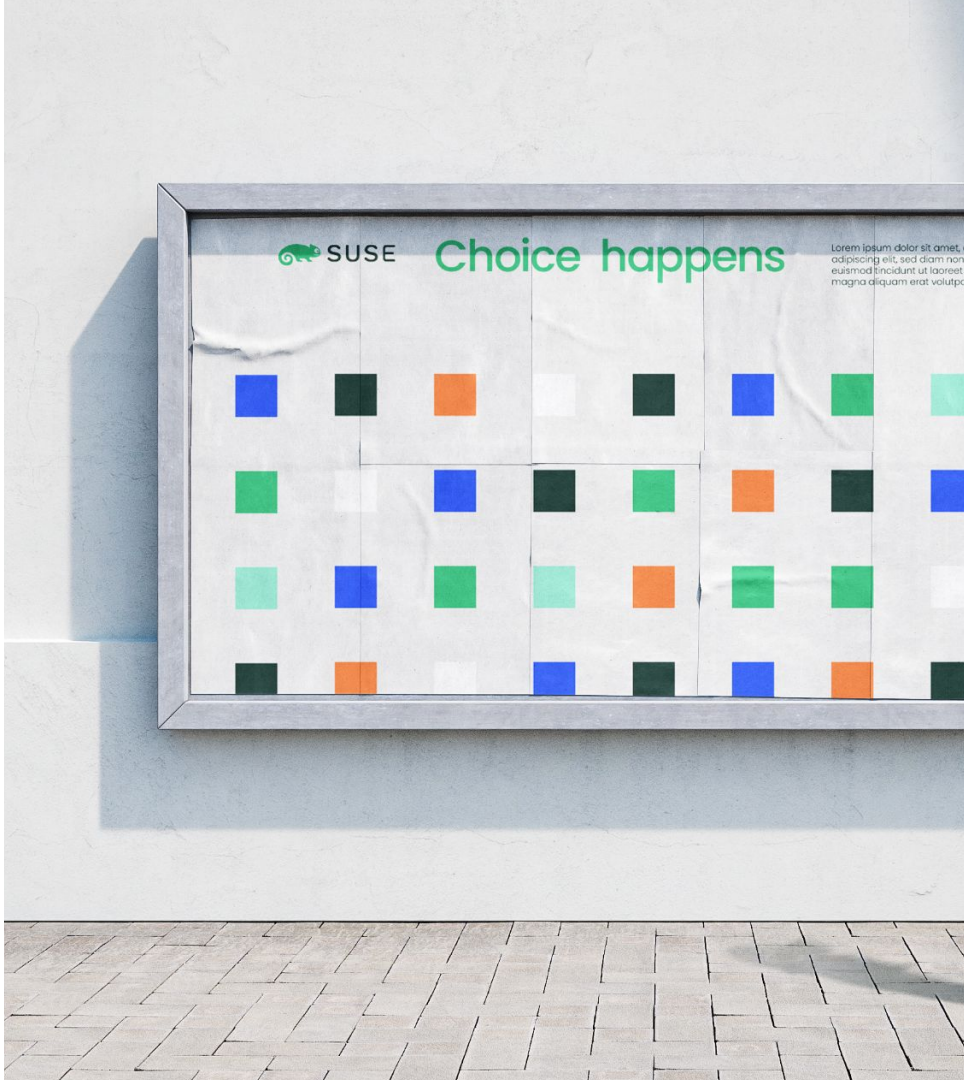
Introducing the Speakers

Mario is a Principal Software Engineer at SUSE, where he leads the development of Rancher Fleet. Previously worked on other system management projects, including Epinio (a Kubernetes based PaaS) and as a project lead for Quarks, which built Cloud Foundry on top Kubernetes.

Juan is a Principal Technical Marketing Manager at SUSE, where he specializes in building value stories around open-source solutions. With a background in cloud-native solutions, Juan previously worked as a Kubernetes consultant, and as a Solution Architect.

Agenda

- Introduction
  - CI, CD, DevOps and GitOps
  - The scale impact on CD
- Fleet
  - Architecture
  - Sharding
  - Target customizations
  - Git repo organization
- Future
  - OCI Ops



SUSE

Introduction to CI/CD

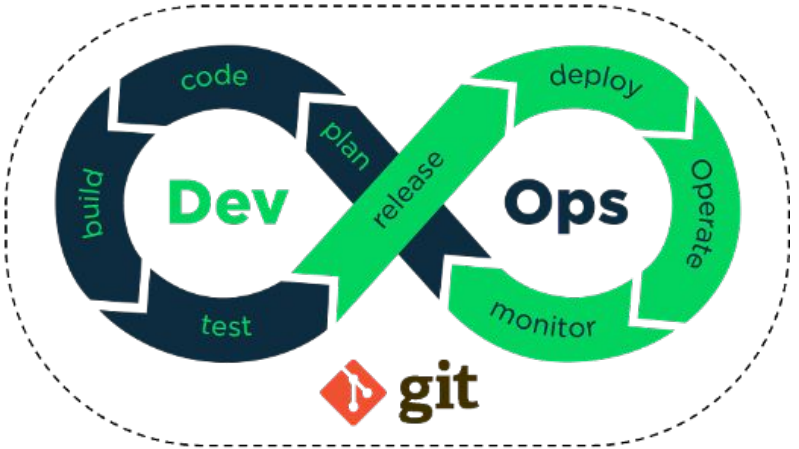
The pillars of modern software build and delivery

1. **Continuous Integration (CI)** is a software development approach where developers regularly merge their code in a central repository. Automated validations, builds, and tests are then triggered helping to surface any integration problem early in the process

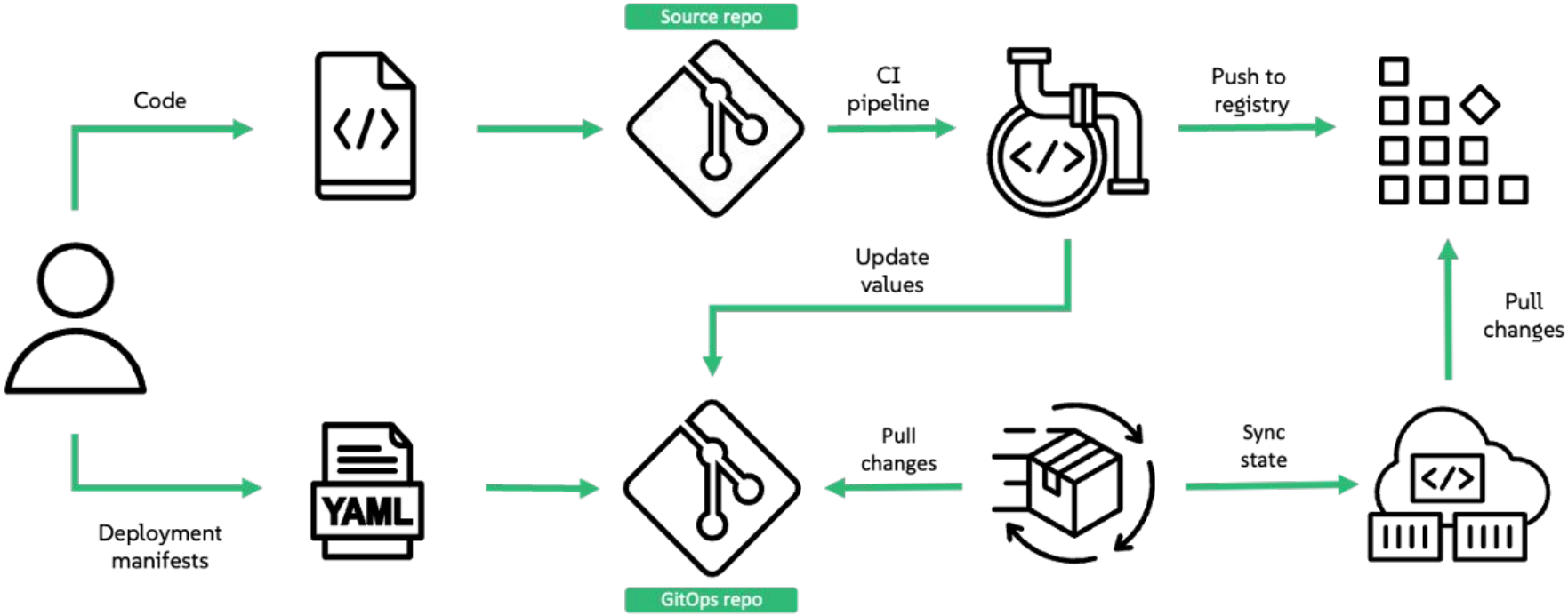
2. **Continuous Delivery/Deployment (CD)** complements CI to cover the automated deployment of all those software artifacts produced by the CI process in a set of target environments

3. **DevOps** is a much broader practice that integrates the development best practices as defined on CI/CD to also cover the operational aspect of application management and life cycle. The name comes from the union of the Development and Operations worlds.

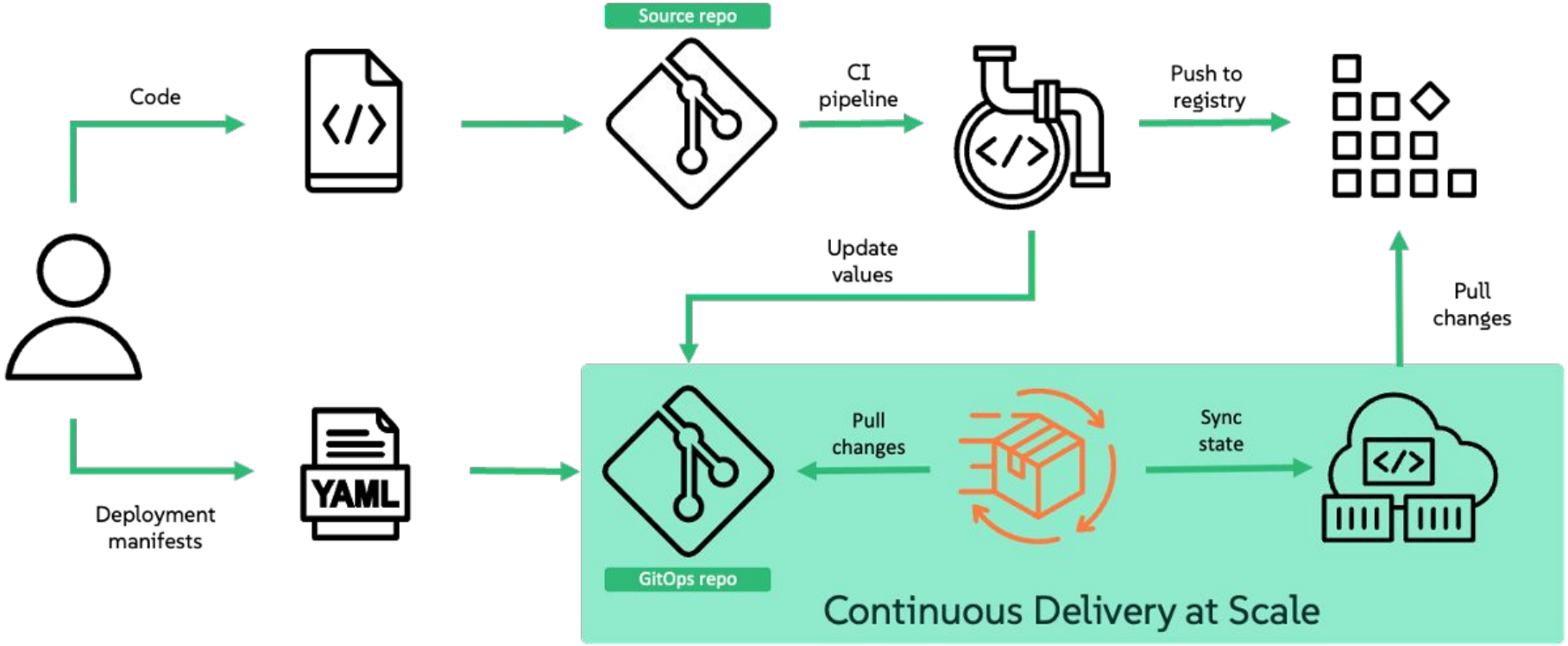
4. **GitOps** is an operations framework built around DevOps best practices that use Git as the source of truth for deployments and environment configuration management (e.g., Kubernetes), ensuring system state alignment and providing an audit trail with rollback capability



The DevOps & GitOps cycle



The DevOps & GitOps cycle … at scale



Where do the problems come from?

Tens to hundreds of repos may need to be constantly monitored for changes

The CD software should keep all in sync and up to date based on complex targeting rules

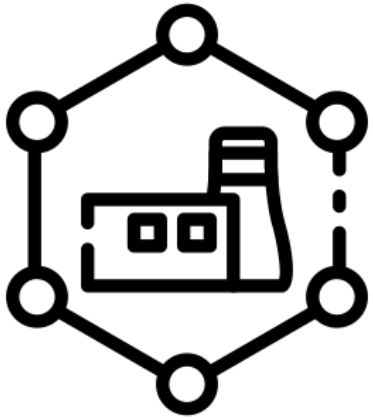
Hundreds or thousands of target clusters should be monitored and reconciled

Just **horizontal scaling** processing power, **sharding** or **delegating** may not enough ... **Securiy** and **operations** may also be impacted by **scale**

SUSE

Our customers' scenarios

Industrial IoT customer



- Scale: 100-120 clusters
- Both owned and franchise factories: non consistent IT
- True global presence (latencies, bandwidth, ...)
- Some factories are considered high security assets
- A shared set of services but factories many different SW
- Many application types and sizes:
  - Custom ERP like local factory management
  - Production linces control electronics
  - Monitoring and tracing stacks

Telco customer

- Scale: 1000 to 5000 clusters
- Basic set of applications: Cloud-native Network Functions (CNF): routers, load balancer, 5g user management, ...
- Highly constrained HW environment

Customer challenges summary

Industrial IoT	• Big scale +100 clusters and ~100 repos • Complex global cluster topology • Many source applications • Complex software/configs targeting rules • Non reliable connections • Many source applications • Security	*Both tried ArgoCD and Flux but faced limitations related to their scale and operational complexity*
Telco	• Massive scale (+1000 clusters) • Not many repos. Simple SW stack. • Resource constraints • HW aware software delivery • Security	

The “real world”

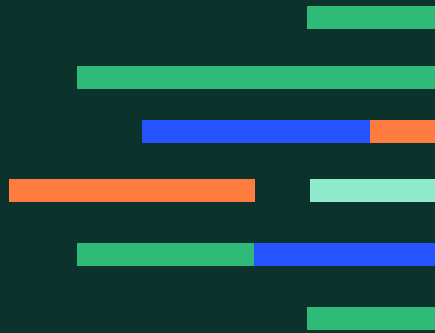
Helmuth von Moltke

“No plan of operations extends with certainty beyond the first encounter with the enemy’s main strength.”

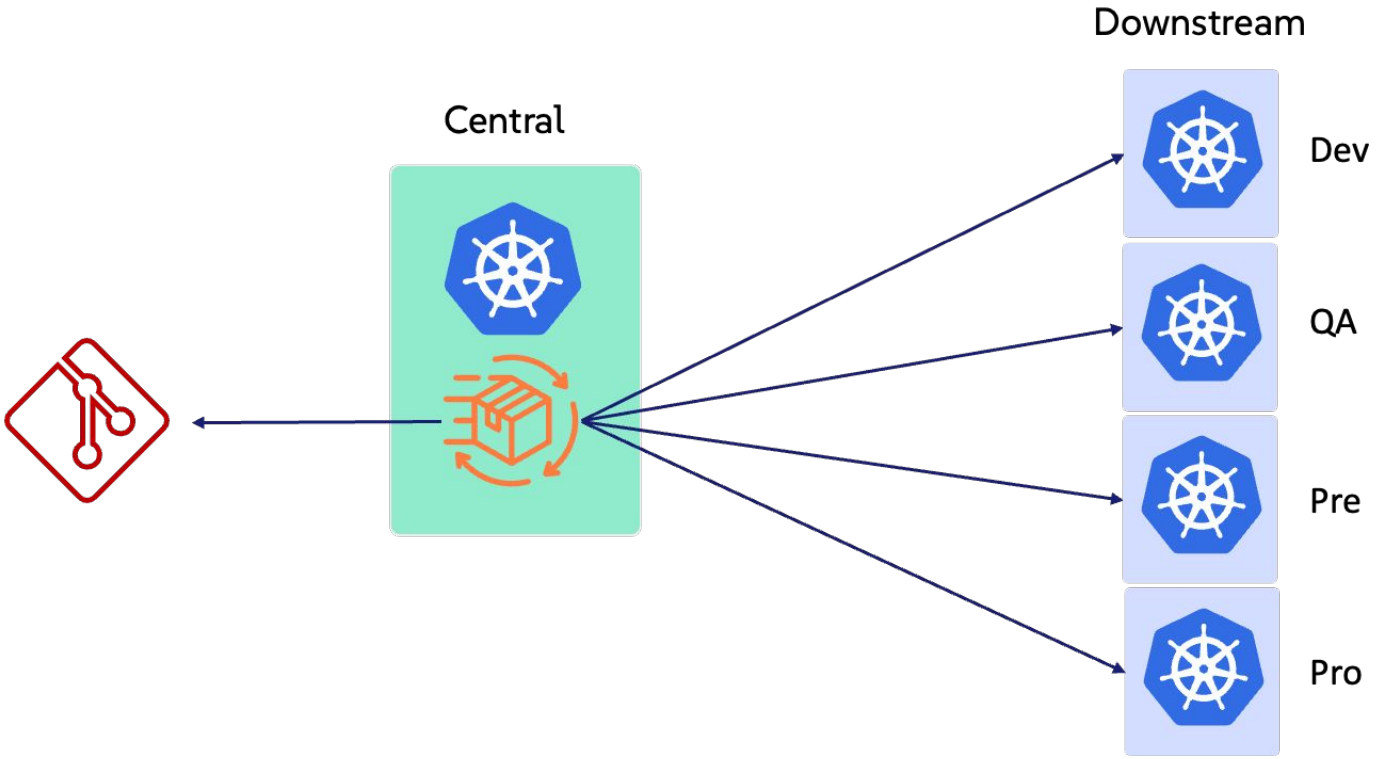
“ArgoCD always look great until you want to use it for more than 10 clusters ;) “

SUSE

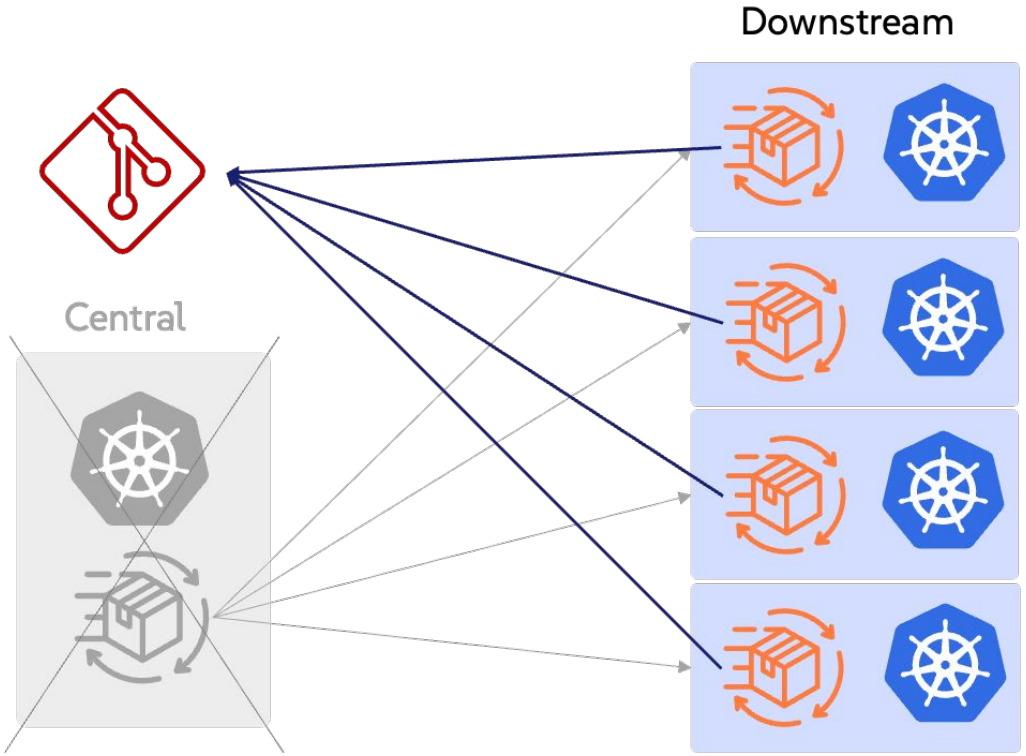
Continuous Delivery architectures



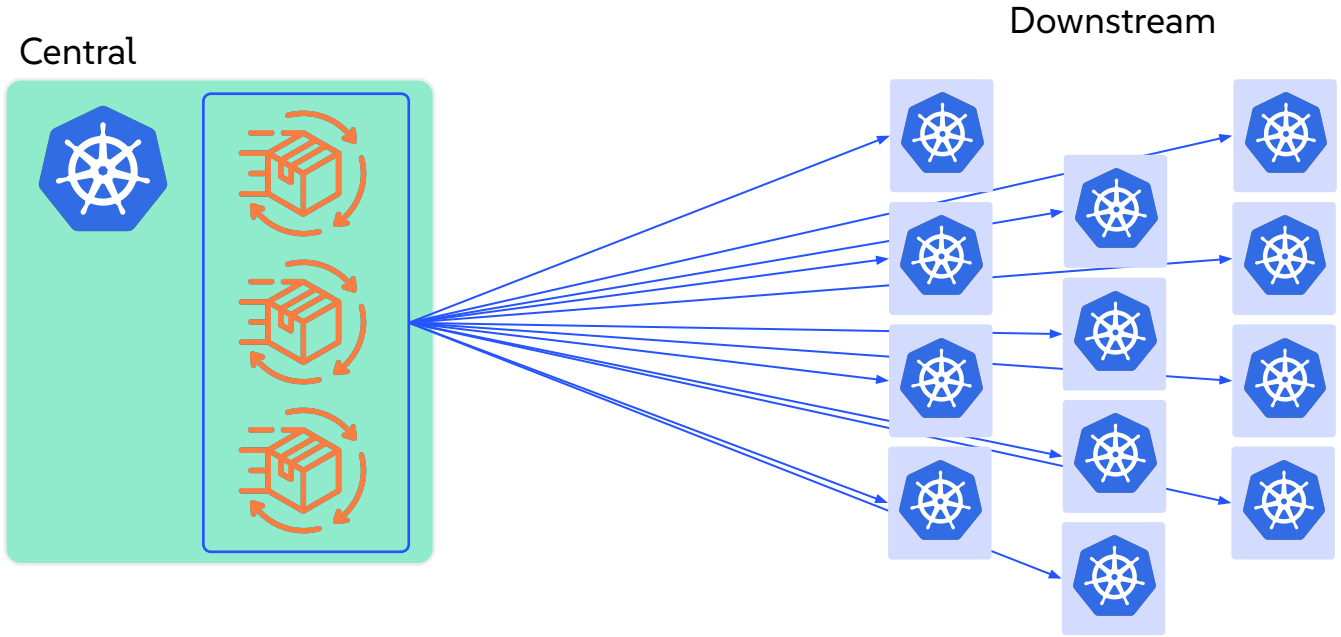
Base models: Hub and Spoke



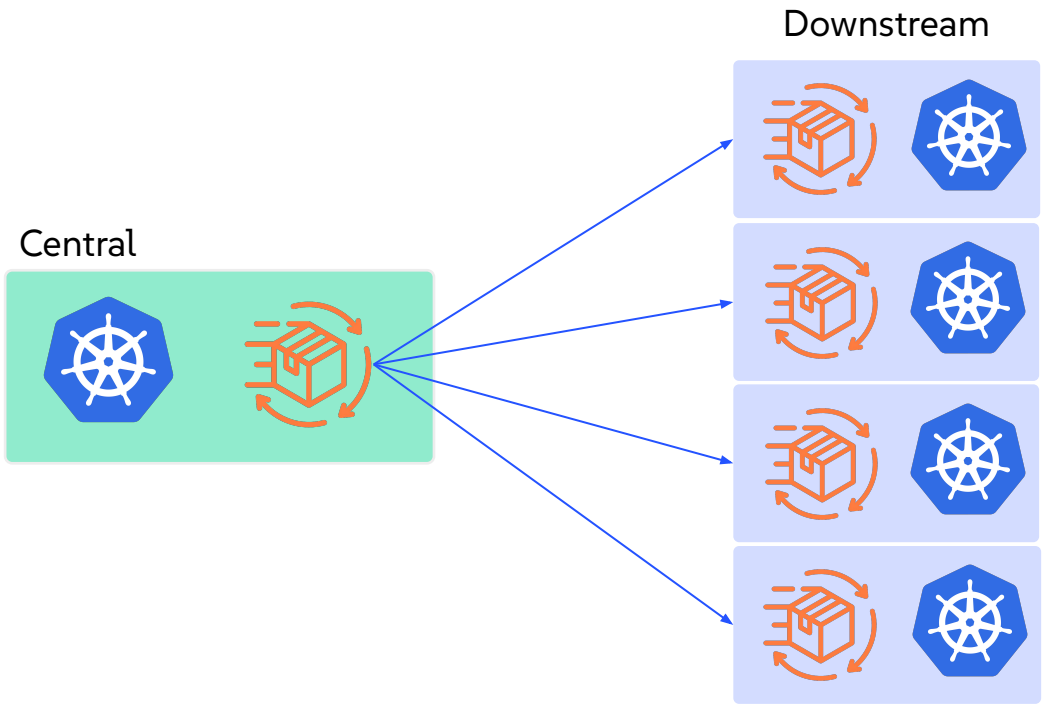
Base models: distributed not coordinated



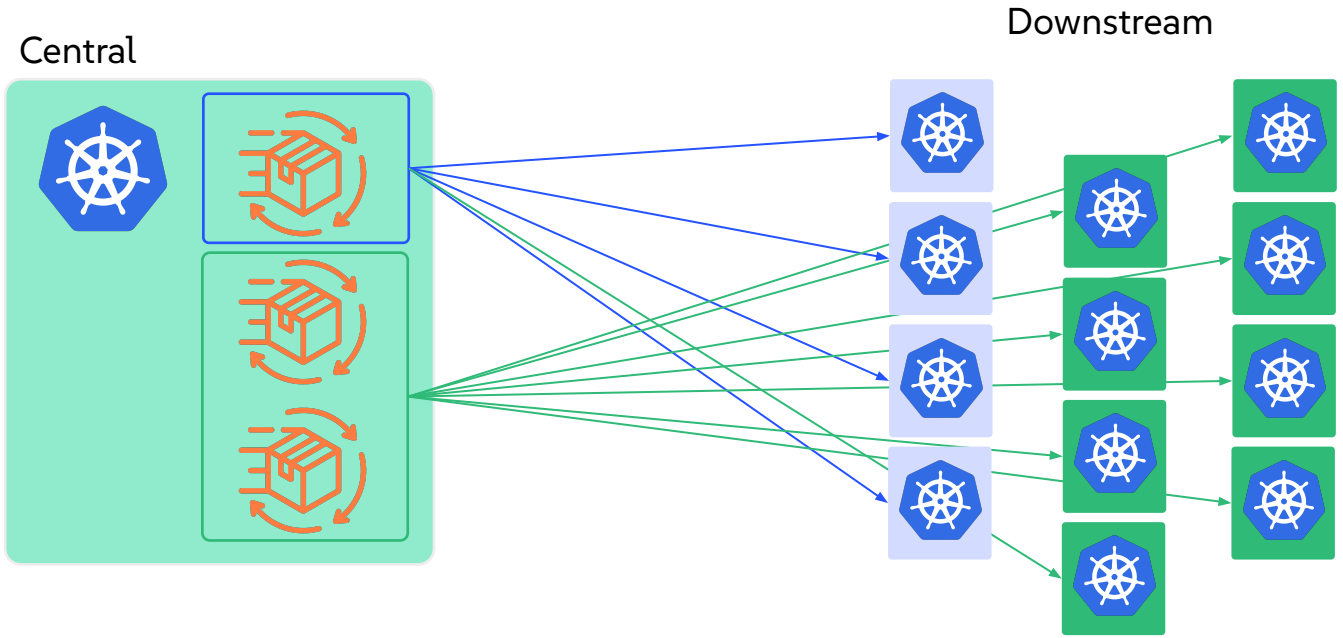
Hub and Spoke horizontal scaling



Hub and Spoke delegated execution

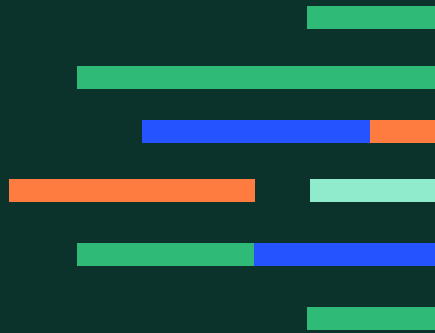


Hub and Spoke sharding and grouping

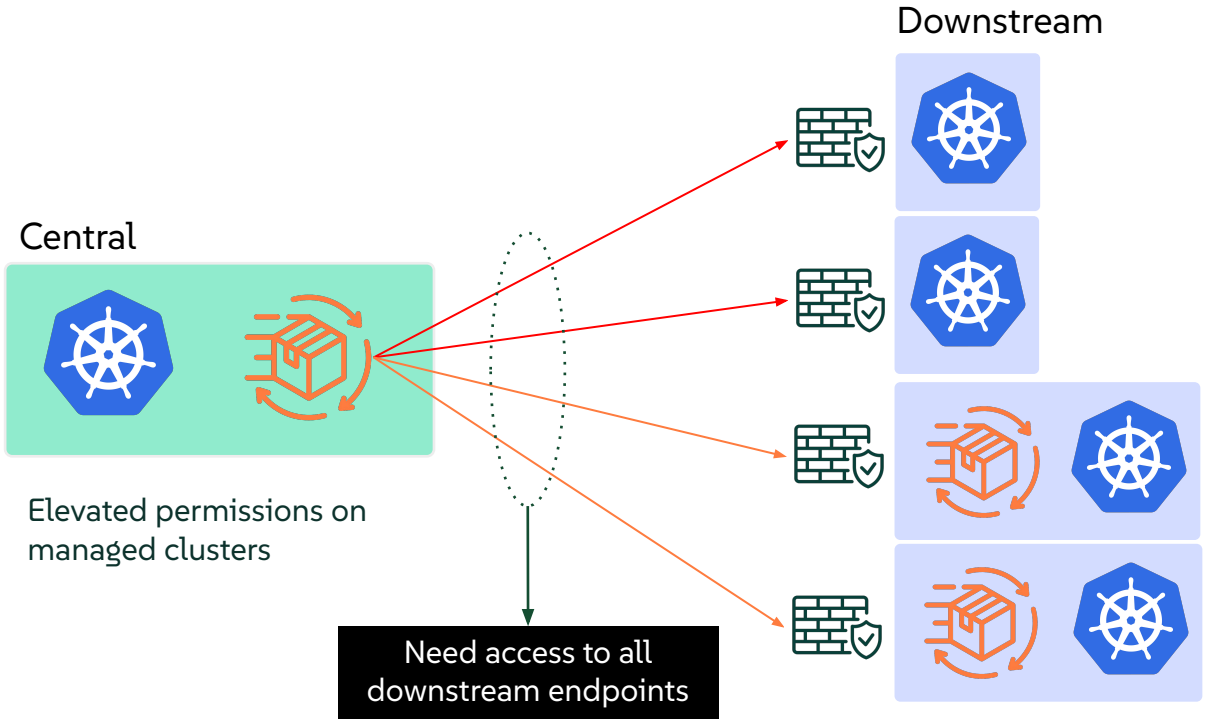


SUSE

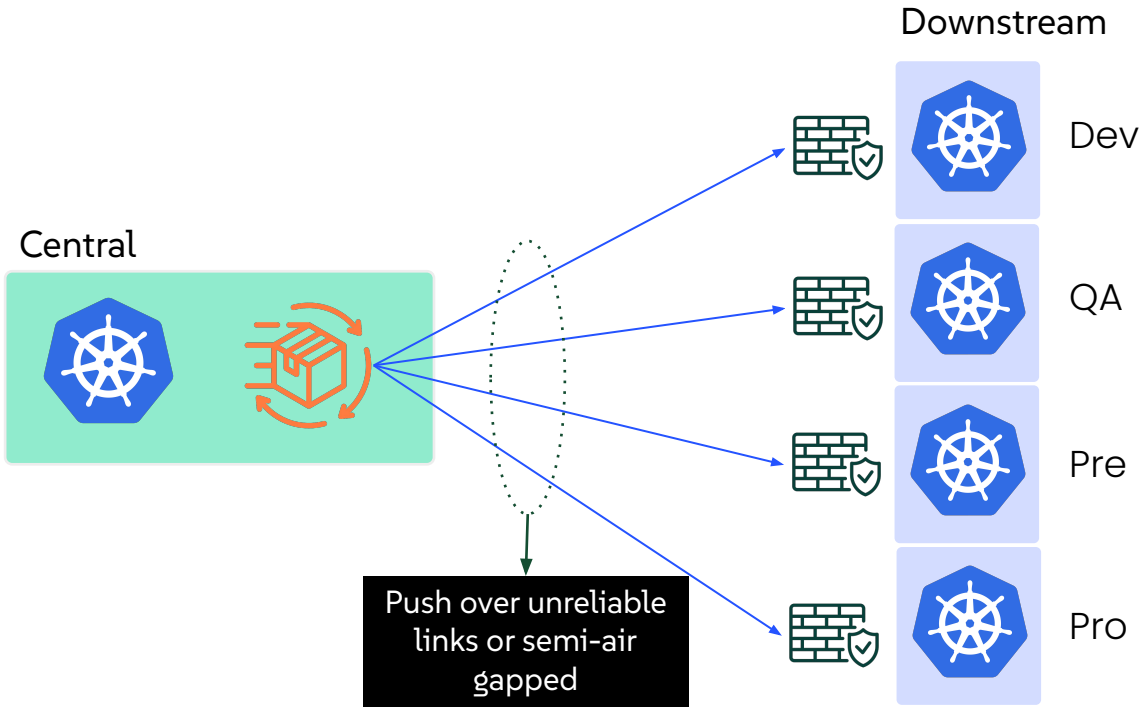
Security and networking concerns



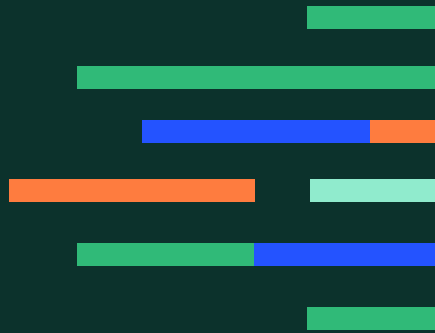
Scale and security



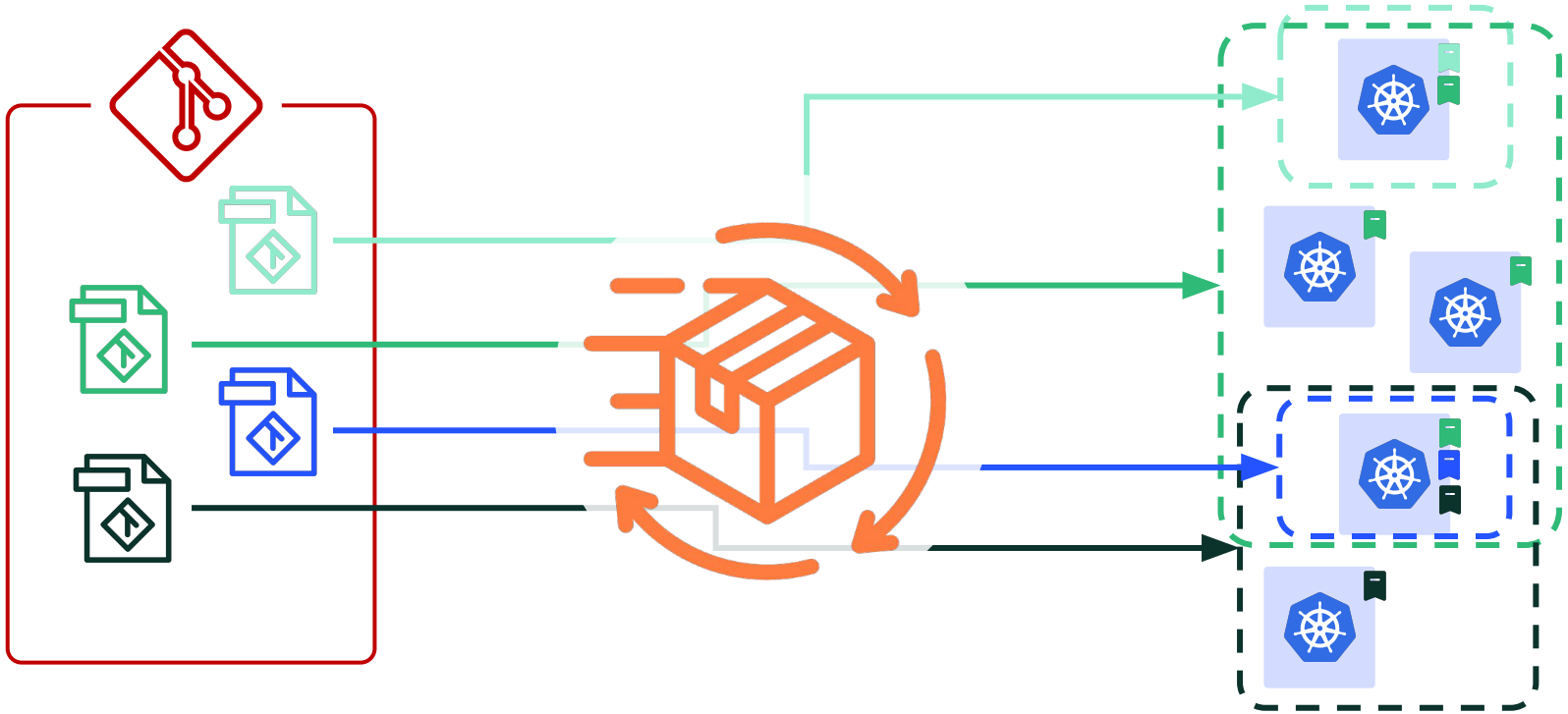
Scale and networking



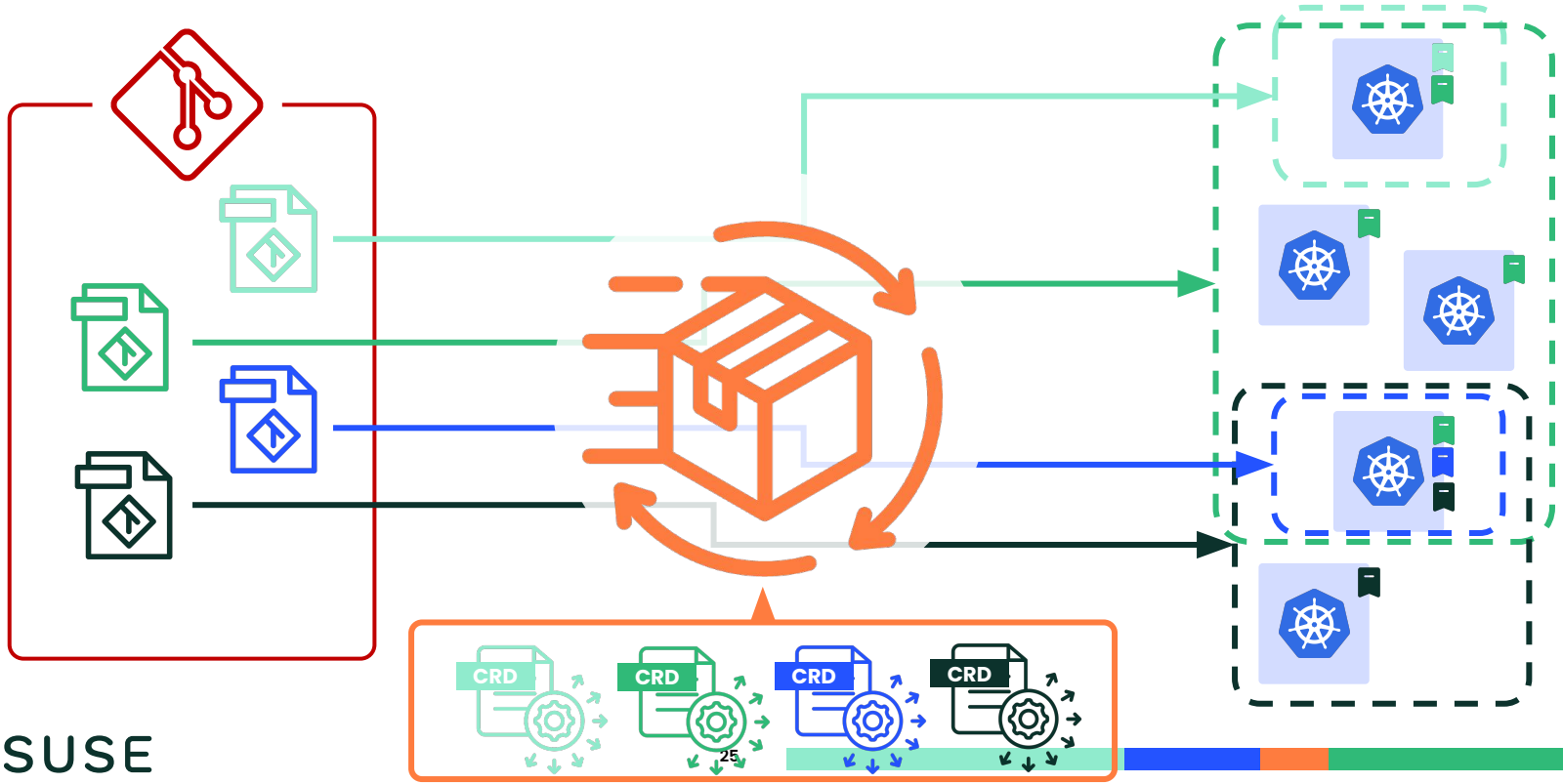
Targeting and repo pooling



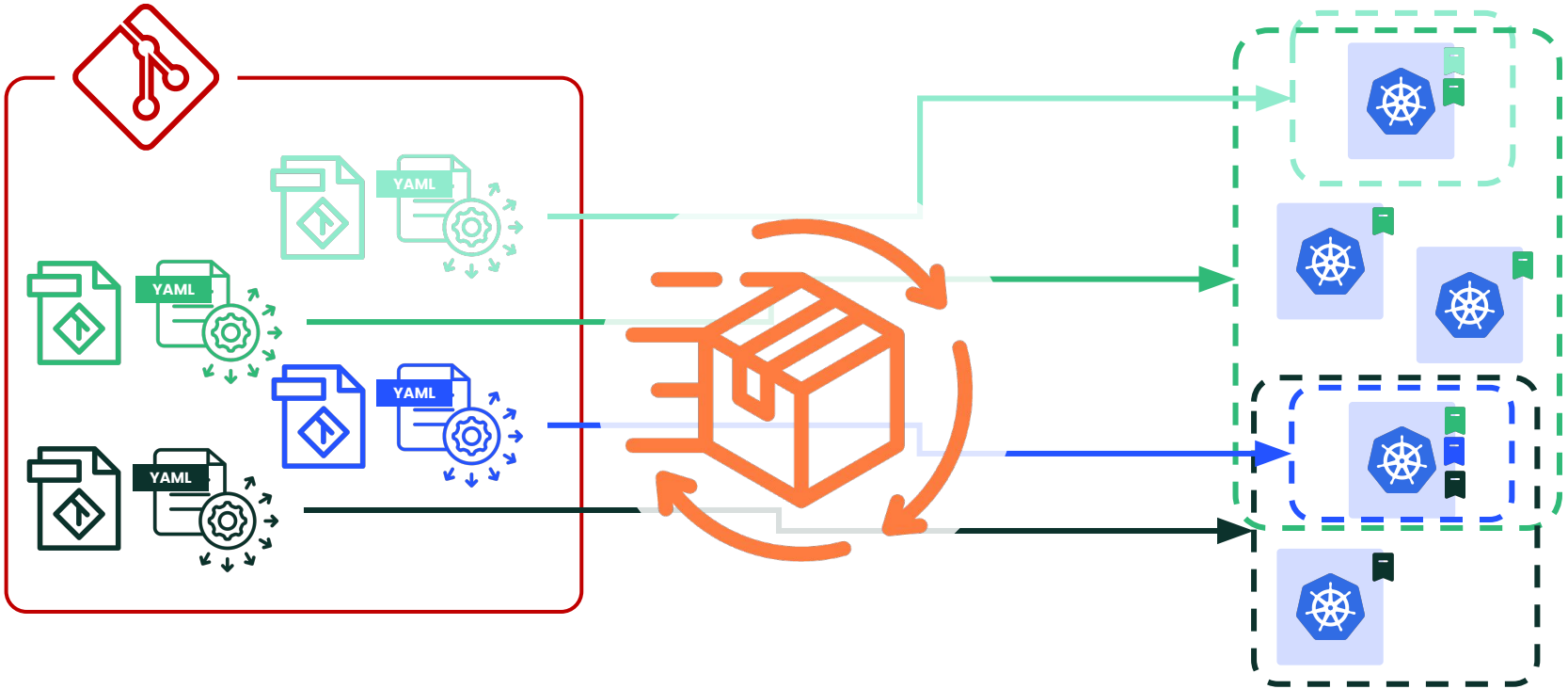
Application targeting in large scenarios



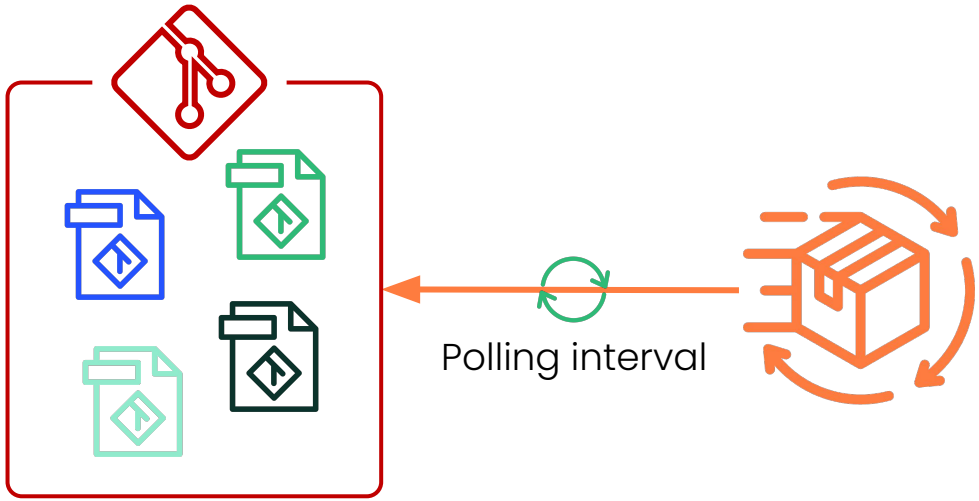
The impact of target rules defined as CRDs



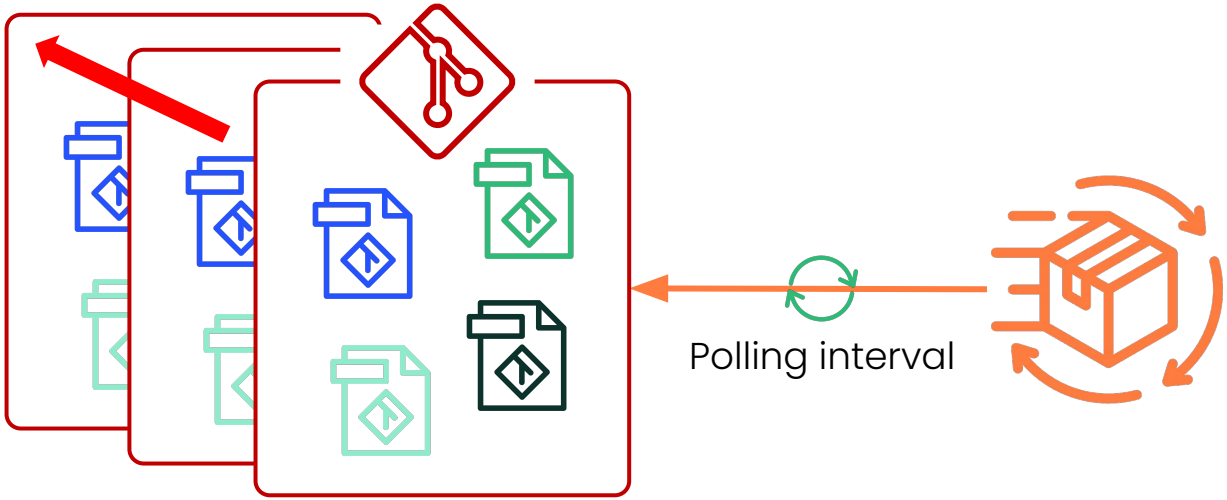
Targeting rules should live in Git



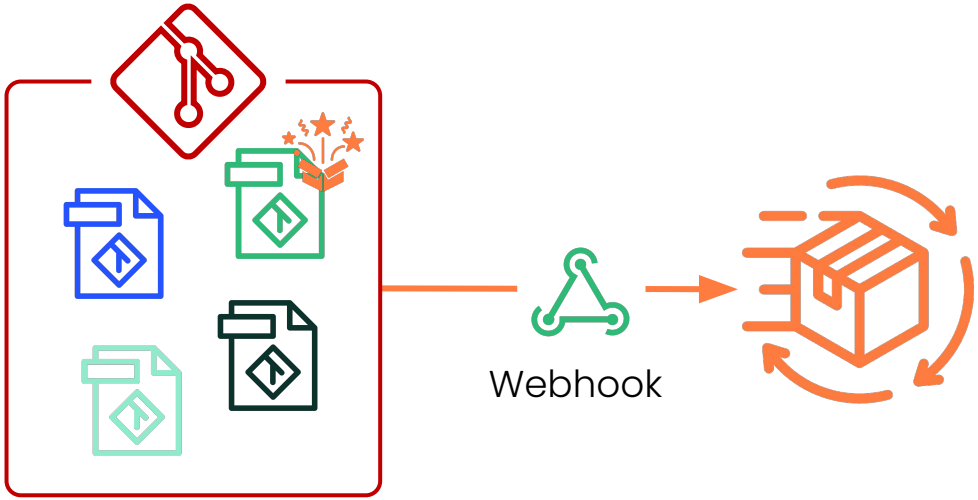
The limiting factors of repo polling



Extra compute power needed to scale

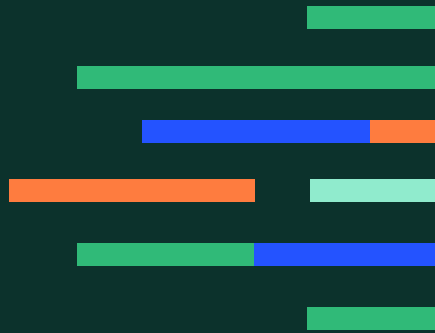


Webhooks are the only way to go ...

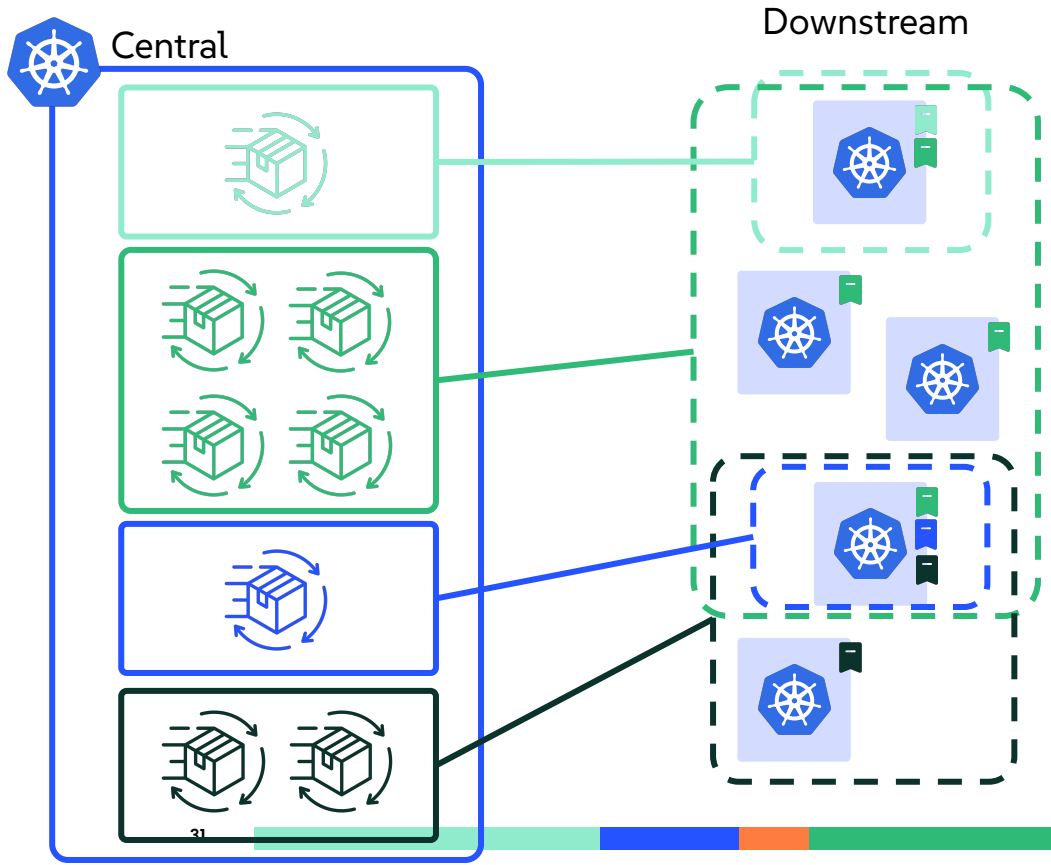


SUSE

The recommended architecture for CD at scale

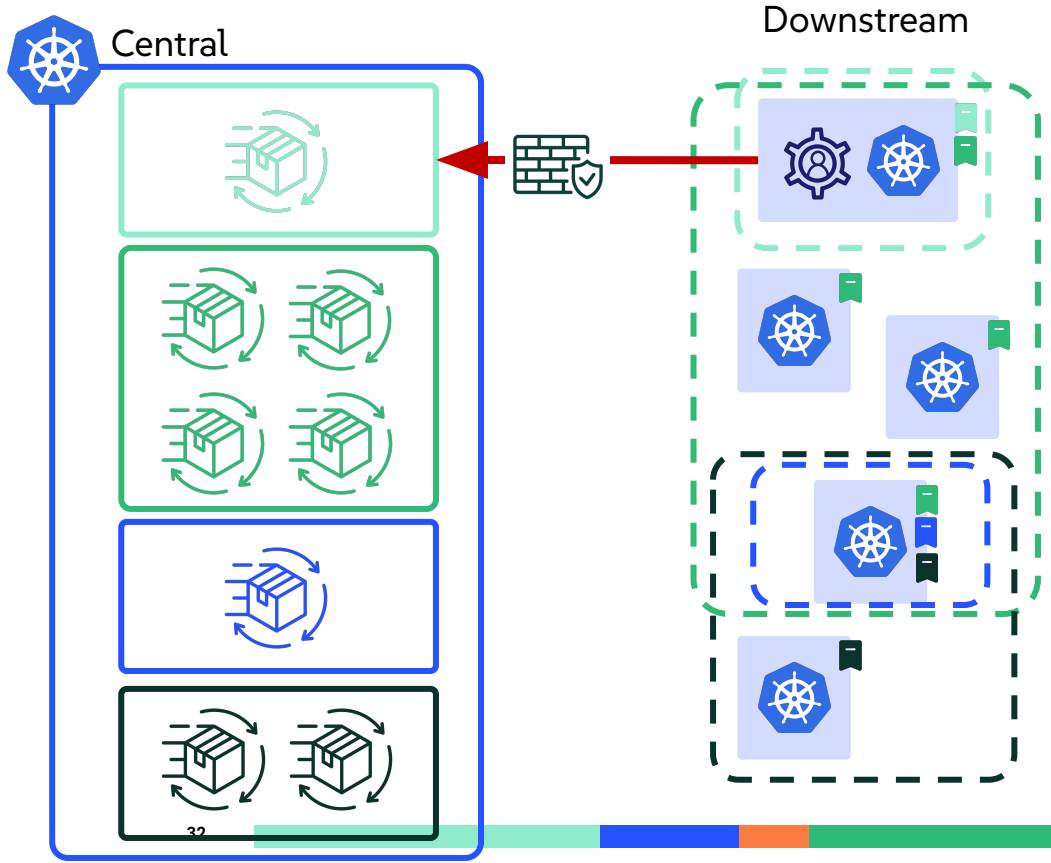


Horizontal scaling + grouping + sharding



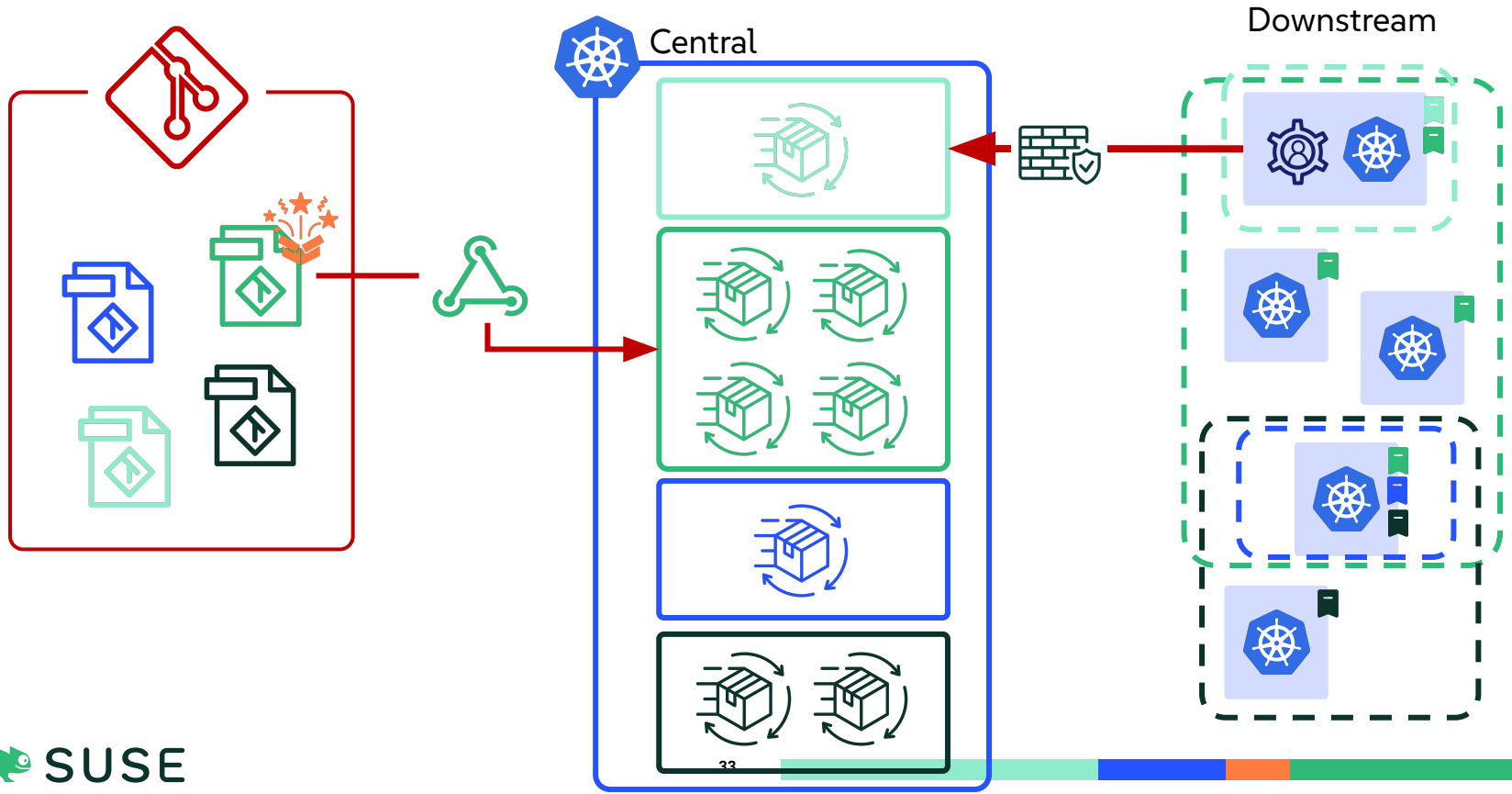
SUSE

Light agent initiates connection and pull changes

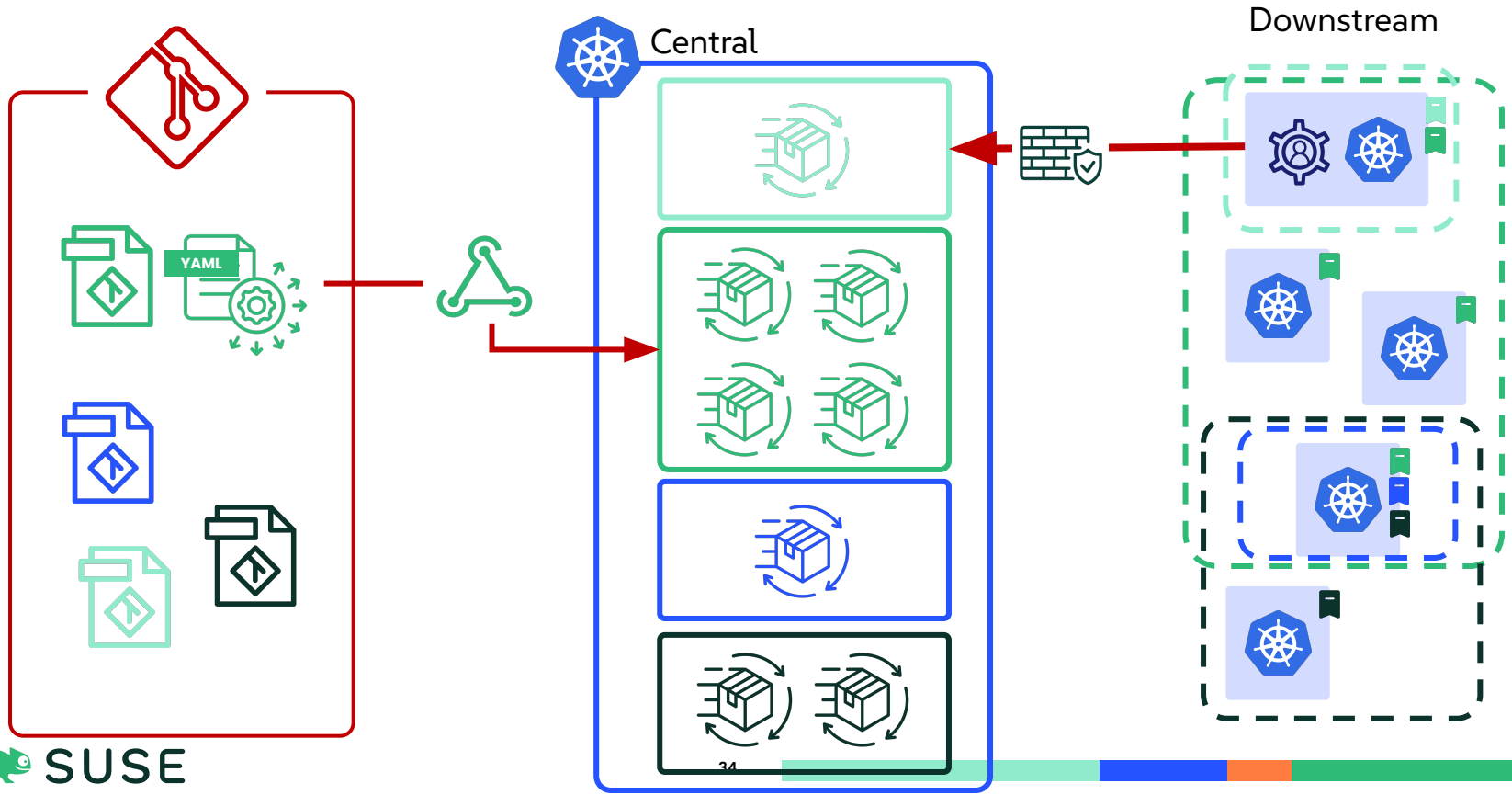


SUSE

Git updates notified via Webhooks



“GitOps friendly” targeting configuration



SUSE

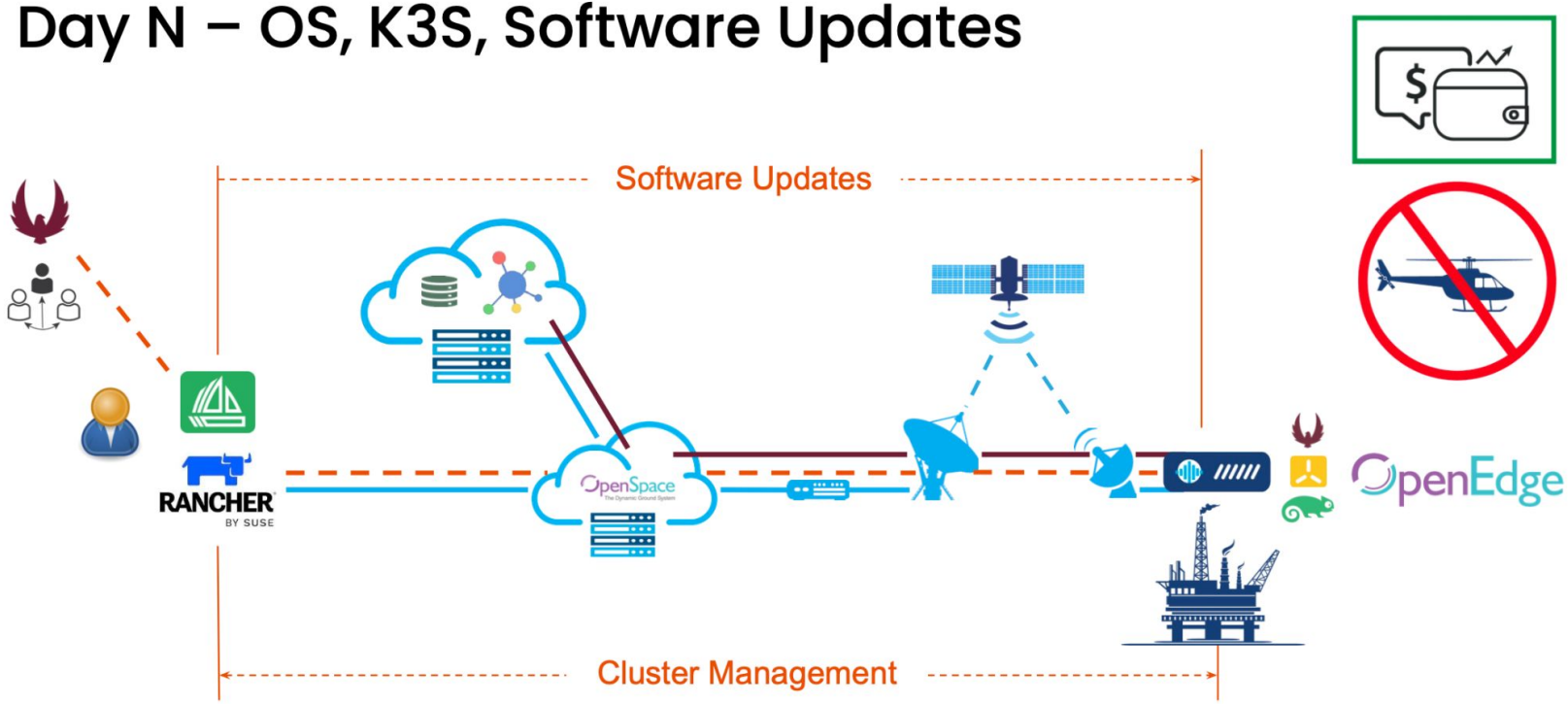
SUSE

Everything ~~is a~~ ~~Computer~~ uses Clusters

Clusters everywhere

- point of sale 
- factories 
- cars 
- planes 
- oil rigs

Day N – OS, K3S, Software Updates



Source: SUSECON24 - Managing the Kratos OpenEdge Software Lifecycle with Rancher

How to manage all those clusters?

Continuous Delivery!

- automated
- fast
- quality feedback

And more specifically: GitOps

Why GitOps

- I need to deploy **monitoring** (Grafana, Prometheus) across North America, EU and Southeast Asia, but data retention policies are different across each geography.
- I am a **Platform Operator**, I want to provision clusters with all components in a scalable and safe way.
- I am an **Application Developer**, I want to get my latest changes automatically into my development environment.

GitOps Principles

- Declarative
  - A system managed by GitOps must have its desired state expressed declaratively.
- Versioned and Immutable
  - Desired state is stored in a way that enforces immutability, versioning and retains a complete version history.
- Pulled Automatically
  - Software agents automatically pull the desired state declarations from the source.
- Continuously Reconciled
  - Software agents continuously observe actual system state and attempt to apply the desired state.

SUSE

Continuous Delivery in Rancher

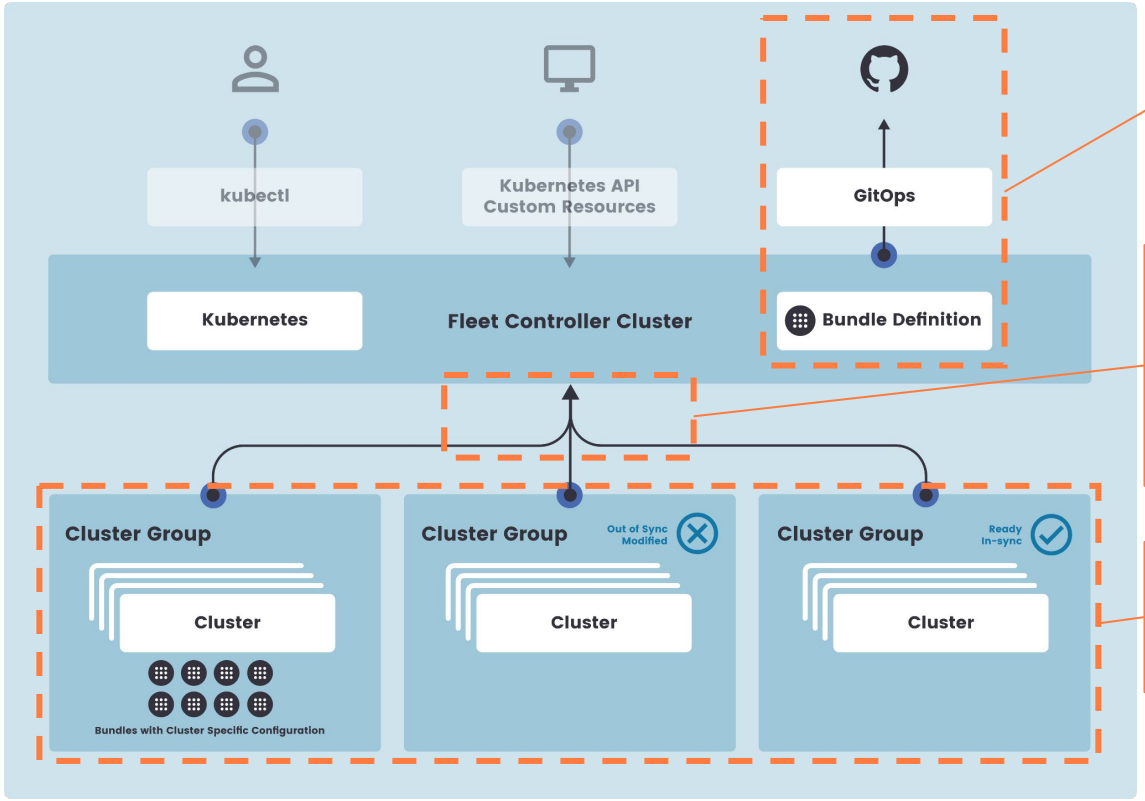
What is Fleet?

- Fleet is a Continuous Delivery/Deployment open-source tool with a GitOps operator.
- It's a Kubernetes-native application so it's API first and relies on CRDs and Operators.
- Created in 2020 by Rancher to offer CD capabilities more closely aligned to the Rancher Kubernetes management philosophy (Kubernetes at **scale**)
- Used internally by Rancher Manager to manage downstream clusters and also available to end users to implement their own CD strategy as Rancher CD
- Can also be deployed in any Kubernetes cluster
- Relevant links:
  - https://fleet.rancher.io/
  - https://github.com/rancher/fleet

Some real world numbers

- QA works on the 1.000 cluster mark
- Some real-world customers using Fleet:
  - Retail: +3000 K3s managed clusters
  - Surveillance cameras: 1 single node K3s cluster per camera. Avg. 50 cameras per venue, designed to scale up to 250
  - Industrial Edge: 120 factories with 2 clusters per factory. ~50 different source repos
  - Telco: network access platform with 900 locations (1x3 node K3s cluster per location)

Multi-Cluster



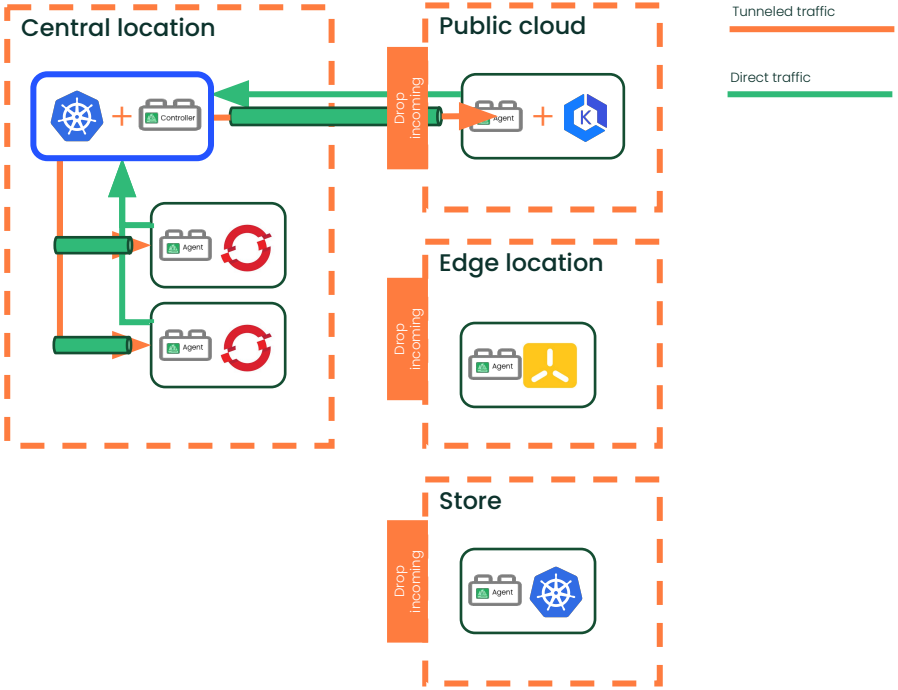
Network traffic optimization

Downstream clusters **PULL** Git repo status and definitions (bundles) from the main Fleet Controller Cluster or Rancher Management Cluster

Advanced targeting and variables: how to target hundreds of repos to thousands of Kubernetes clusters

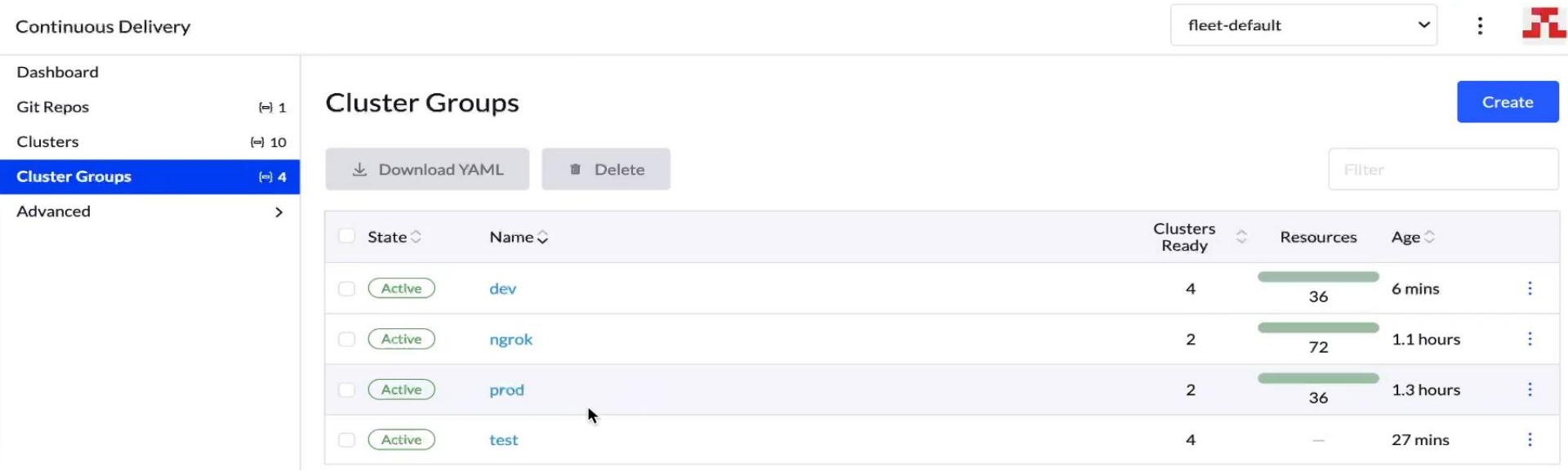
Secure and flexible topologies

Todo: Fix object sizes in the diagram



- The management layer doesn't need direct access to managed clusters. No exposed endpoints.
- Simplified FW and networking.
- Expiring registration tokens and restricted Service Account
- Locations only access their bundles and cluster status
- Direct traffic used for payload: create Bundles, check for repo changes, …
- Supports non persistent links and air gapped scenarios

UI


Continuous Delivery
fleet-default
Dashboard
Git Repos 1
Clusters 10
Cluster Groups 4
Advanced
Cluster Groups
Create
Download YAML
Delete
Filter
State
Name
Clusters Ready
Resources
Age
Active dev 4 36 6 mins
Active ngrok 2 72 1.1 hours
Active prod 2 36 1.3 hours
Active test 4 — 27 mins

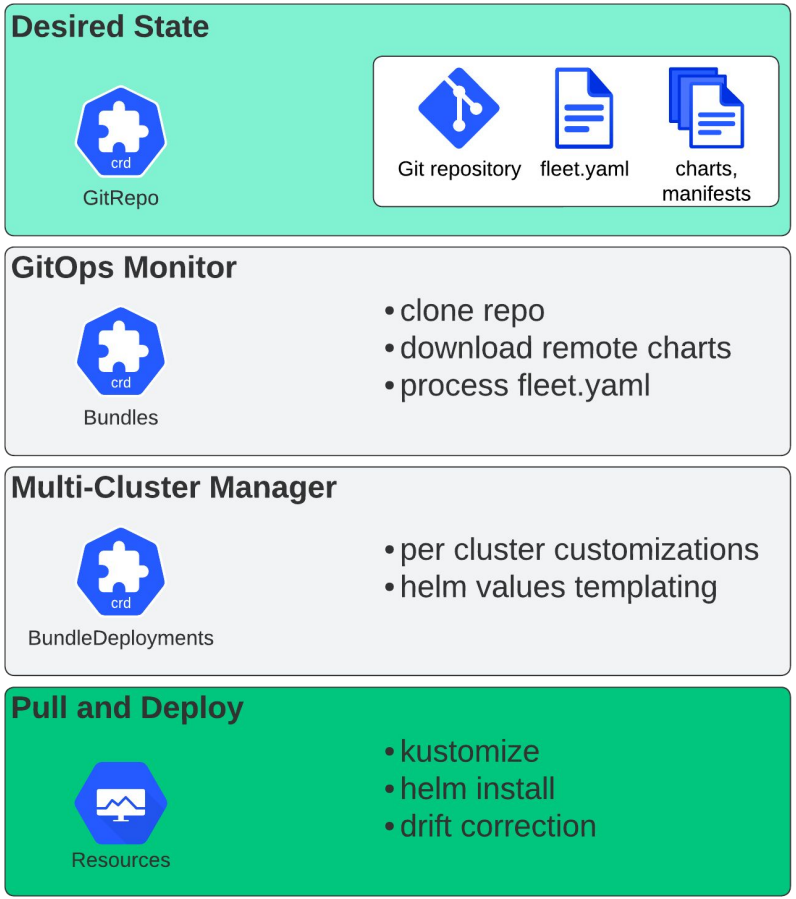
Other features summary

- Full support for Helm and Kustomize
- Git: Multi-repo and multi-branch
- Git: Both polling and web-hook modes
- Supports for dependencies
- Powerful Targeting Semantics
- Built-in RBAC
- Store Fleet configuration (fleet.yaml) in managed Git repos

How does it work?

Components:

- GitOps monitor
  - Creates “bundles” from git
- Multi-cluster manager
  - Targets bundles at clusters
- Agent
  - Deploys bundle to cluster



Customization per Cluster

Override deployment settings per cluster, e.g.:

- Namespace
- Helm values
- Keep resources when removing
- Kustomize, overlays
- Drift detection exceptions
- Drift correction

SUSE

Performance

GitOps Performance

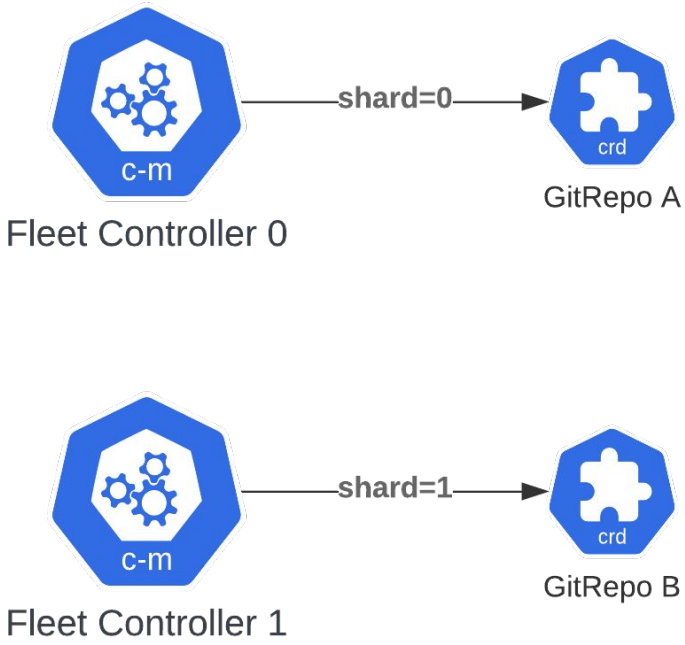
- Polling Git repositories vs. webhooks
- Configure polling interval
- Controller restart sync, periodic re-sync
- Size of Git repository, assets in separate repo?
- Fewer GitRepo resources -> less polling
    - use paths inside one repo

Resource Usage on Control Cluster

- Etcd limits affecting Fleet
  - resource size
  - max number of resources
- How many resources per cluster registration (~14)
- How many resources per deployed git repository (~13)

Horizontal Scaling

- Sharding
- One controller per node
- Labels on resources to assign to controller

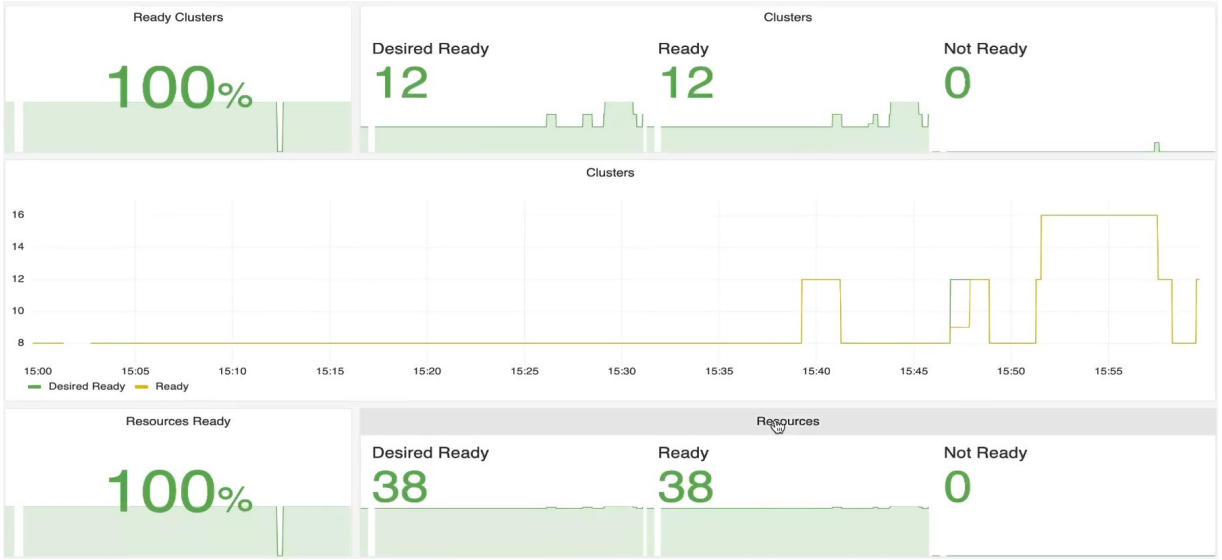


Drift Correction

- works “offline”
- processing on downstream cluster
- oil rig :)

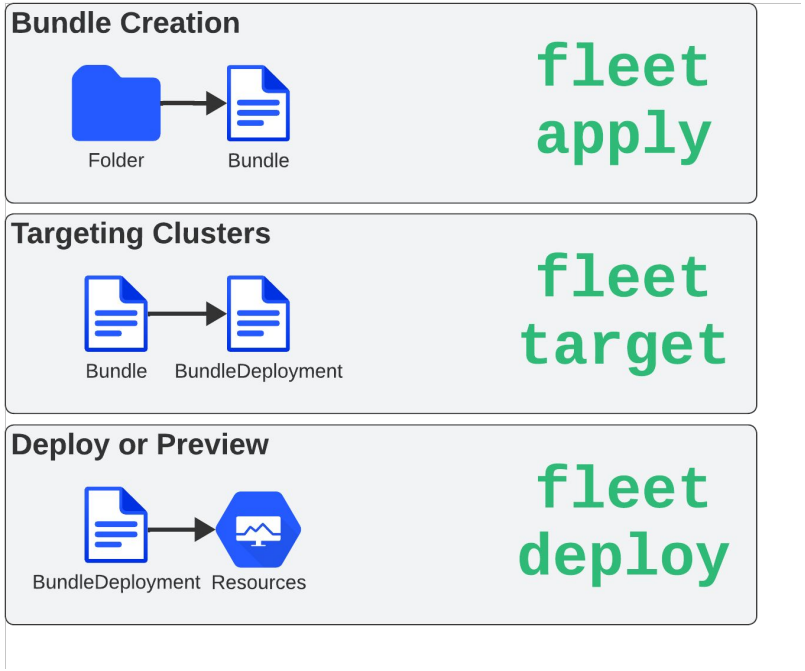
Observe Performance

- resource status
- monitoring / metrics



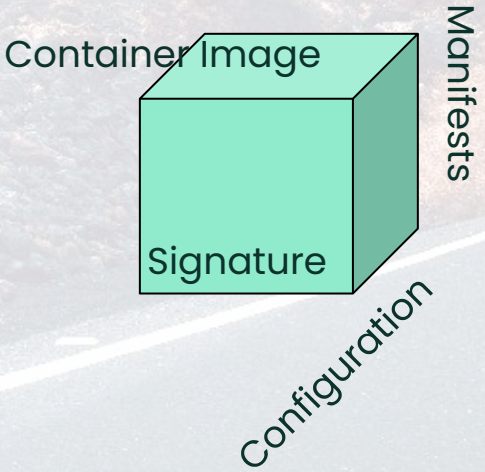
Fleet CLI - Testing

- `apply`
  convert existing folder to a bundle
- `target`
  prepare for clusters
- `deploy`
  deploy/dry-run prepared resource to a cluster



Roadmap 24/25

- Scheduling deployments
- Better CD UX (focused on GitOps up to now)
- OCI-Ops (Helm-Ops)



Created without AI

Thanks!