LF NETWORKING
Developer & Testing Forum

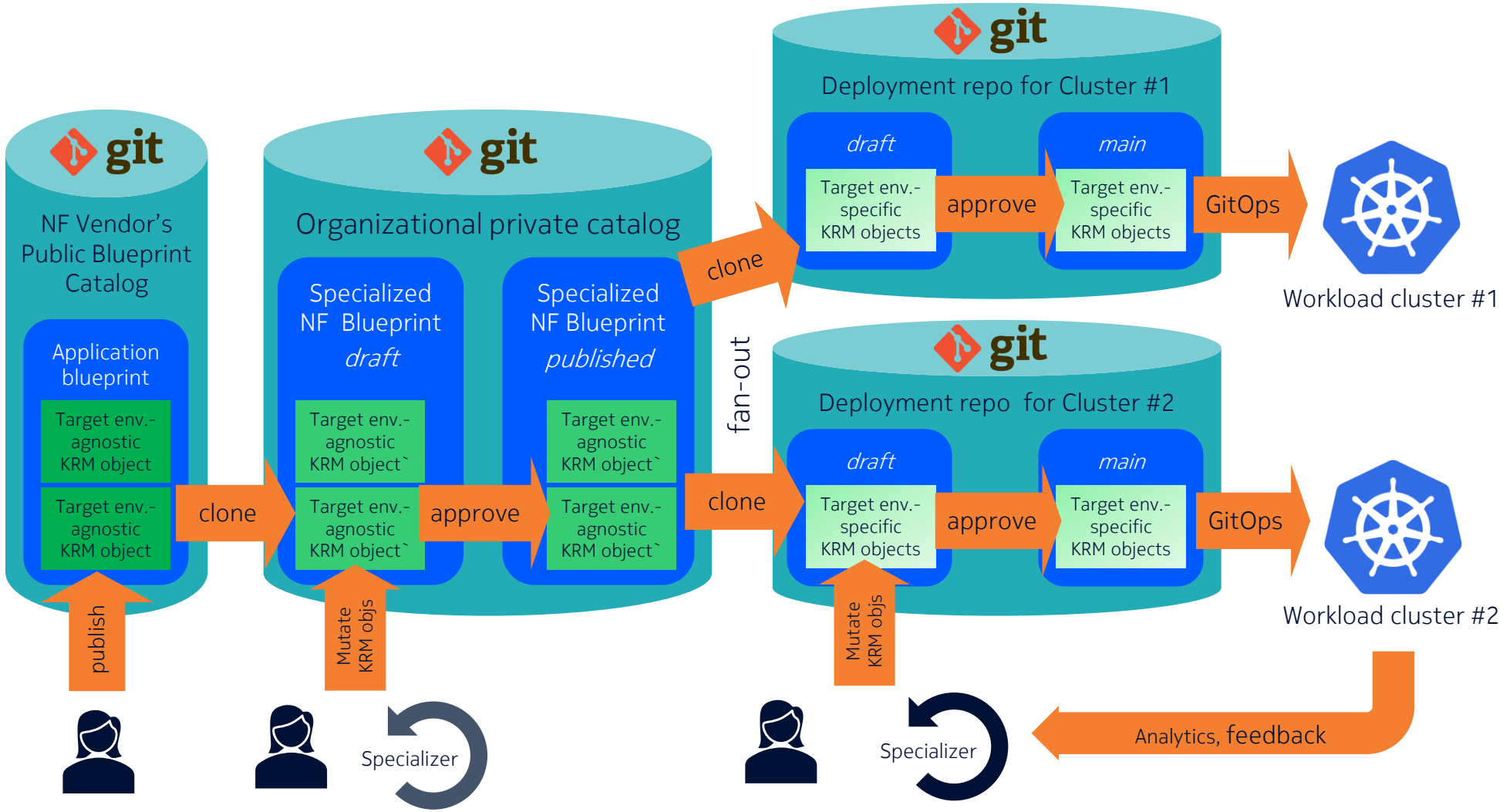
kpt & the dirty details of Nephio

etcd is so 2022… store your Kubernetes resources in Git instead

Istvan Kispal, Nokia Bell Labs

https://lfnetworking.org

Nephio “hydration” process



kpt
a new kid in the (CNCF) sandbox

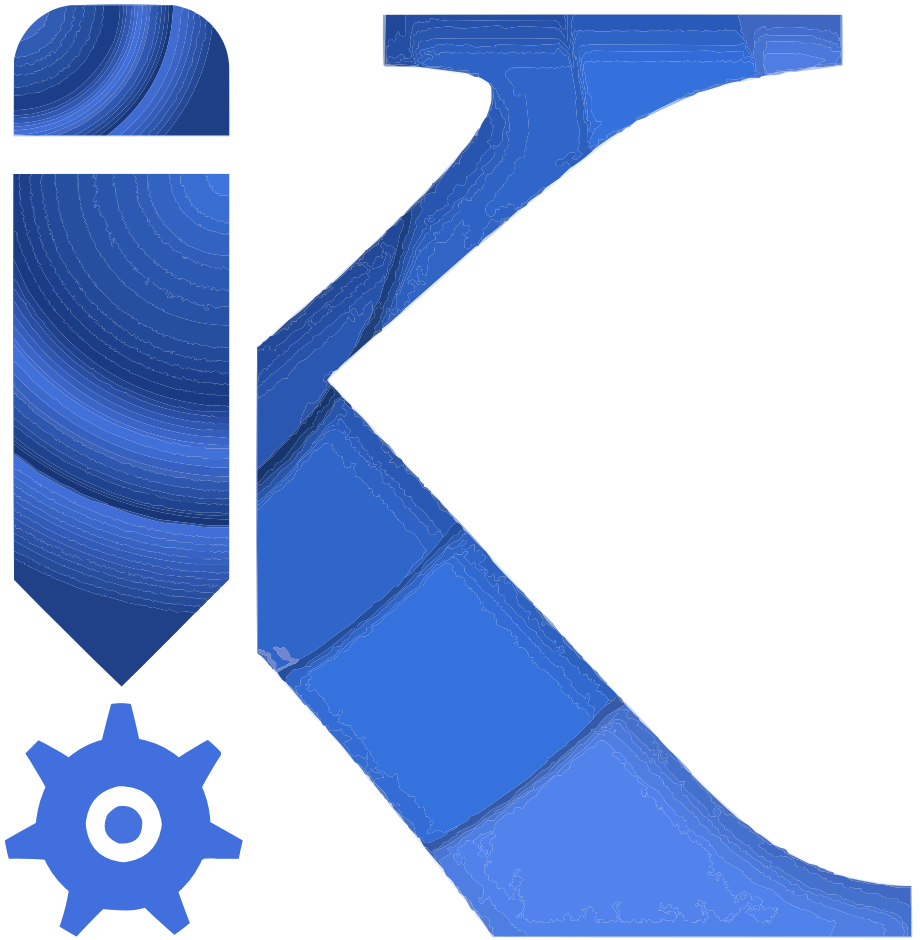
LFN Developer & Testing Forum

https://kpt.dev/

https://www.cncf.io/projects/kpt/

Big umbrella project, with multiple parts:

- kpt package format
- KRM function support:
  - KRM function specification
  - KRM function SDK
  - KRM function catalog
- kpt CLI
- porch: Package ORCHestrator [1] [2] [3]
- ConfigSync: GitOps tool (syncs git repos to K8s clusters)
- Backstage UI plugin: GUI



Declarative configuration management

It is increasingly common to use a declarative approach to manage a growing set of infrastructure and application configurations

Buzzwords

- Infrastructure-as-Code
- Configuration-as-Code
- GitOps
- Configuration-as-Data

All of the above applies the same principles that drives Kubernetes itself to infra-/config management.

Config-as-Data

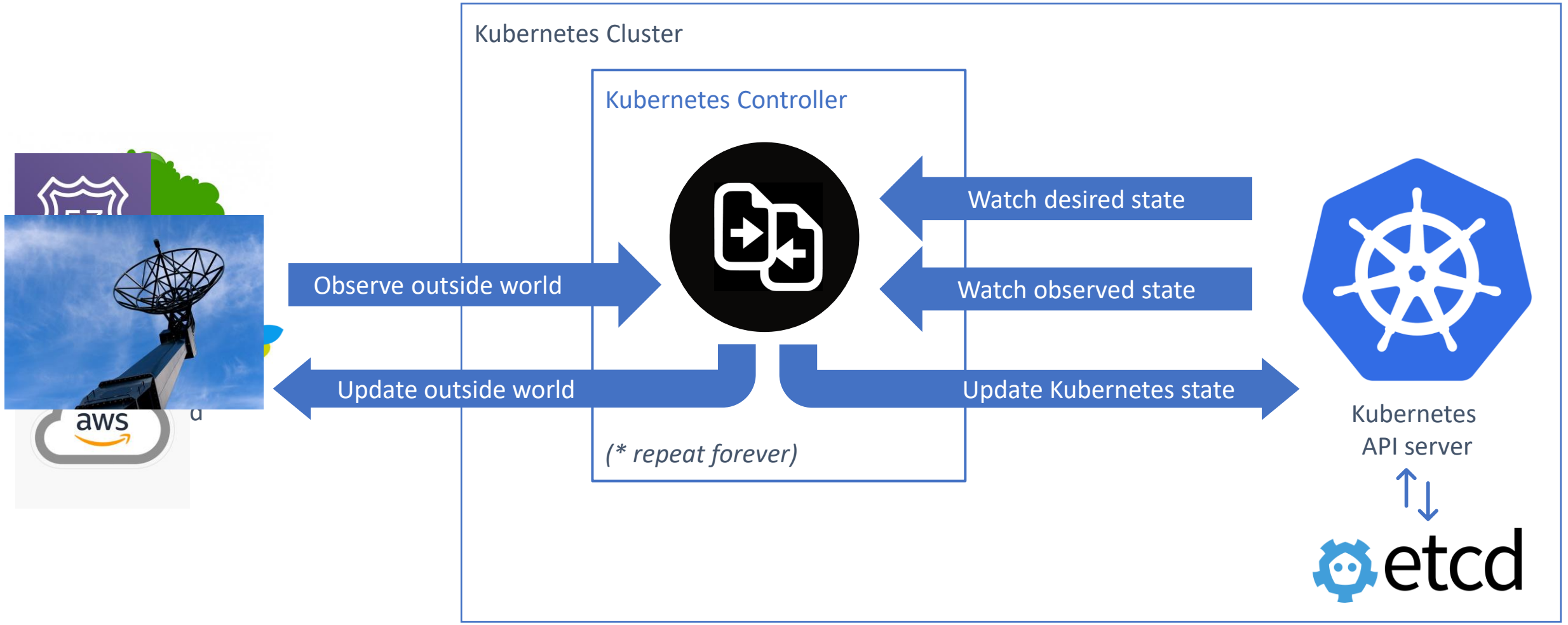
Core principles:

- uses a uniform, serializable data model to represent configuration (KRM)
- makes configuration data (packages) the source of truth, stored separately from the live state (ConfigSync)
- separates code that acts on the configuration (functions) from the configuration data (kpt packages)
- abstracts the storage layer (using the KRM function I/O spec) so that clients manipulating configuration data don’t need to directly interact with it

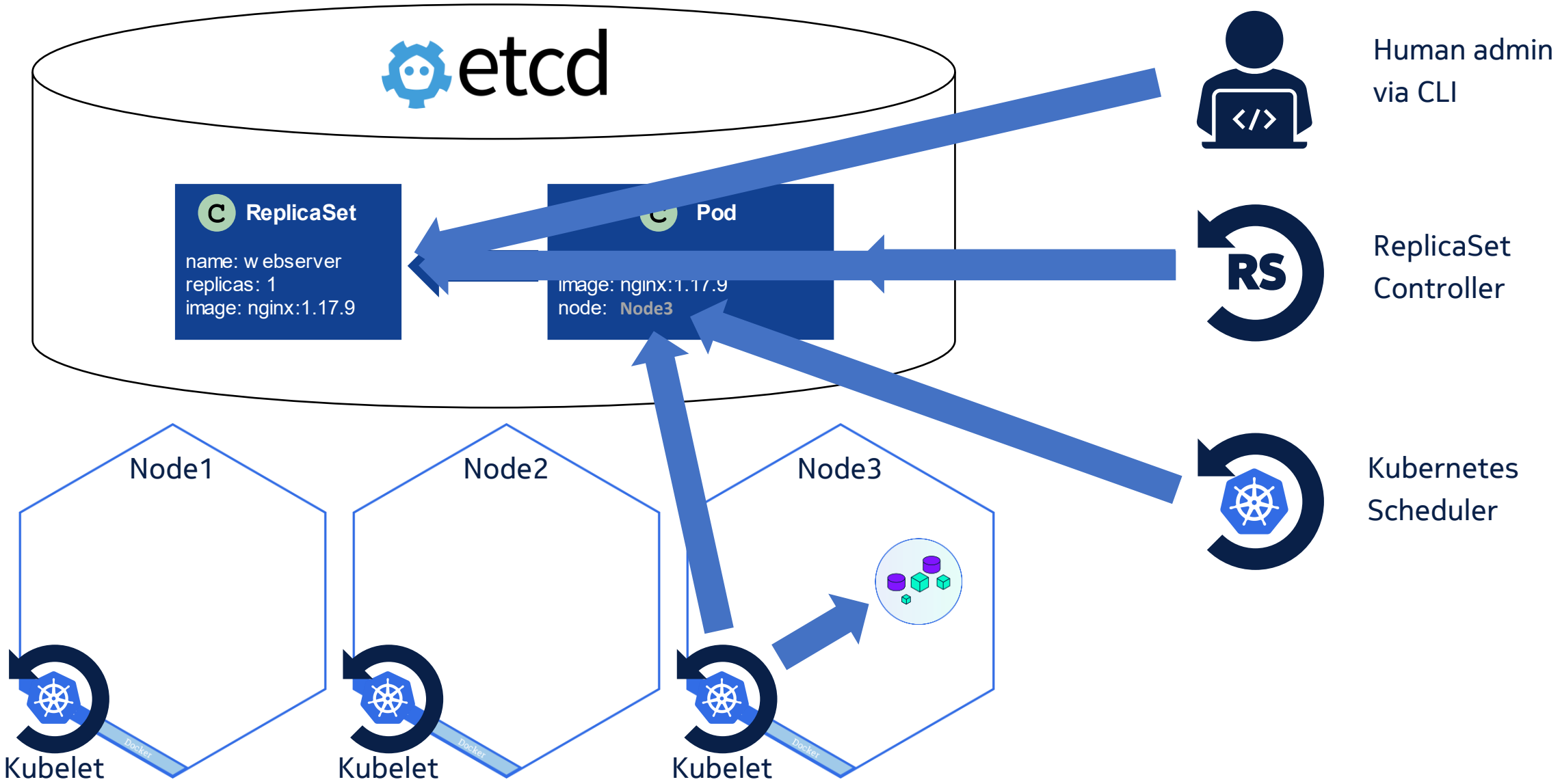
https://github.com/kptdev/kpt/blob/main/docs/design-docs/06-config-as-data.md

Kubernetes controller (simplified)

"Fancy Kubernetes-ism for a client that talks to the API in a loop."



ReplicaSet example



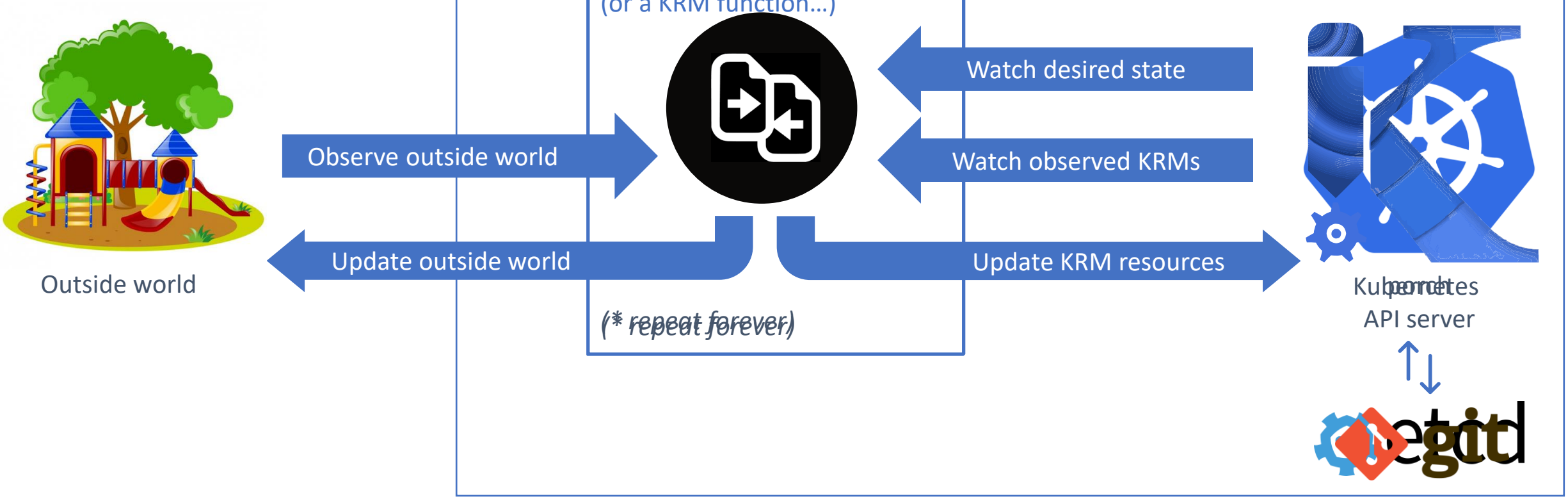
porch API abstractions

Porch enables CRUD operations on KRM resources in Git

- The Porch API is a Kubernetes API itself.
- It defines the following Kubernetes Resources:

Porch API resource	Semantics
Repository	~ Git repo, stores packages It can be a special kind of repository that is watched by a GitOps tool (called a “deployment repo”).
PackageRevisionResources	Contents of a package (YAML files) in Git
PackageRevision	immutable version of a package: ~ Git Tag
PackageVariant	Clone of a package. Usually in another repository. Usually contains modifications to the upstream package (pipeline, config injection, etc.)
PackageVariantSet	A set of PackageVariants ☺

Introducing variance into packages - Package Specializers



Practical takeaways so far

Use Nephio hydration if

- following GitOps principles is a given to you (because it makes sense and/or because your boss forces you)
- you want to enable the manipulation of your configuration data by humans AND machines at the same time
- you want some of the changes to be approved by humans or long-running validation processes

Consider using Kubernetes' Controller pattern to implement almost any automation or control loop tasks that you come across.

Use *declarative tools* for infrastructure-, lifecycle- and configuration management. Consider *GitOps* if you want to avoid configuration drifts (pets vs. cattle), and/or if you want a comprehensive audit log of config changes.

Helm charts and all package formats using *templates* or opaque code *are hostile to automation* (custom abstractions/DSLs, excessive parameterization).

Automate your GitOps setup with kpt by writing custom specializers and custom KRM functions.

KRM functions

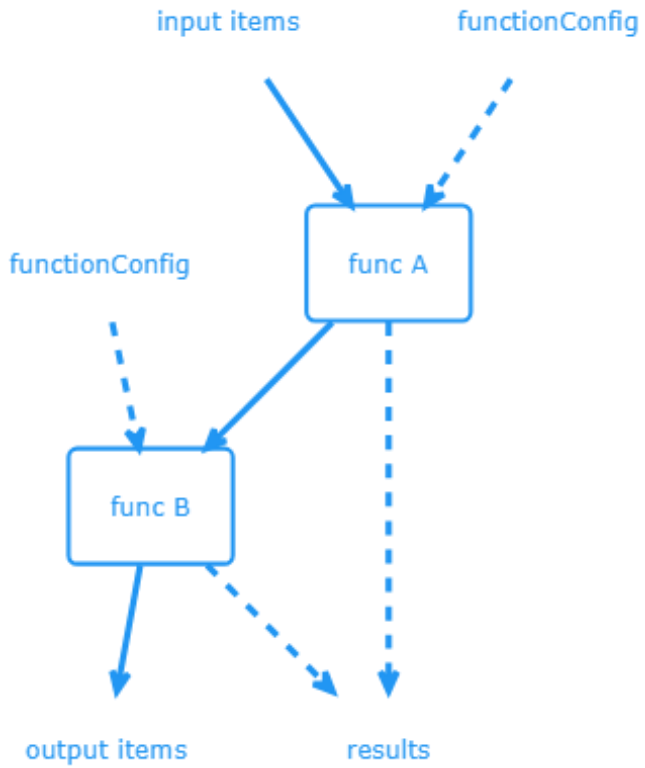
Remember these Config-as-Data principles from before?

- separate code that acts on the configuration (*KRM functions*) from the configuration data (KRM resources in kpt packages)
- abstracts the storage layer (using the *KRM function I/O spec*) so that clients manipulating configuration data don't need to directly interact with it

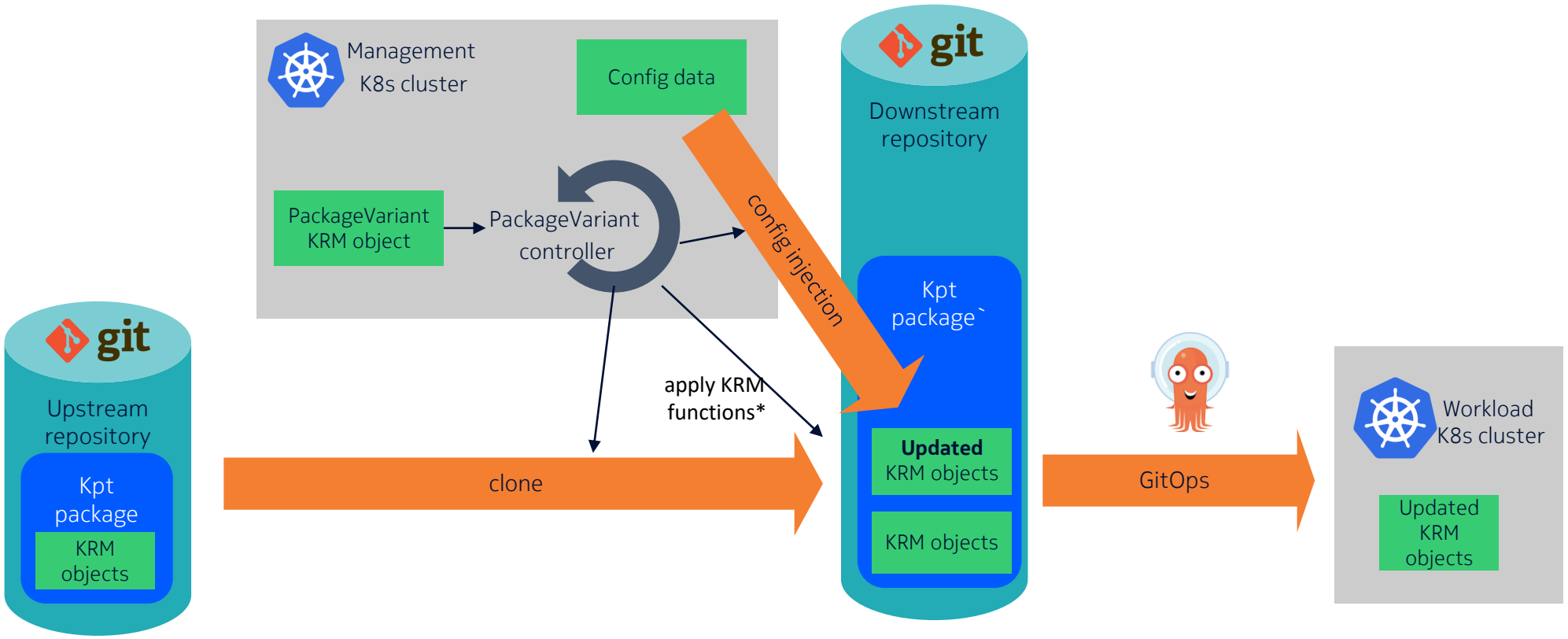
A KRM function is a containerized program that can perform CRUD operations on a set of KRM resources.

Input from stdin, output to stdout → decoupled from storage (Git)

Kpt package metadata contains a pipeline of KRM functions, making it possible to customize the package without using templates, like kustomize.



PackageVariant



PackageVariantSet – PodSet analogy

Package Variance Pattern	Pod Sets
NF Blueprint kpt package	PodSpec (excluding the “node” field)
PackageVariant (target: deployment repo of a cluster)	Pod (target: node)
PackageVariantSet	~DaemonSet
Cluster (actually deployment repo)	Node
(potential future inter-cluster scheduler) Fills downstream repo (“target cluster”) field of PackageVariant if it is empty	Kubernetes scheduler Fills “node” field of Pod if empty

Putting it all back together – hydration revisited