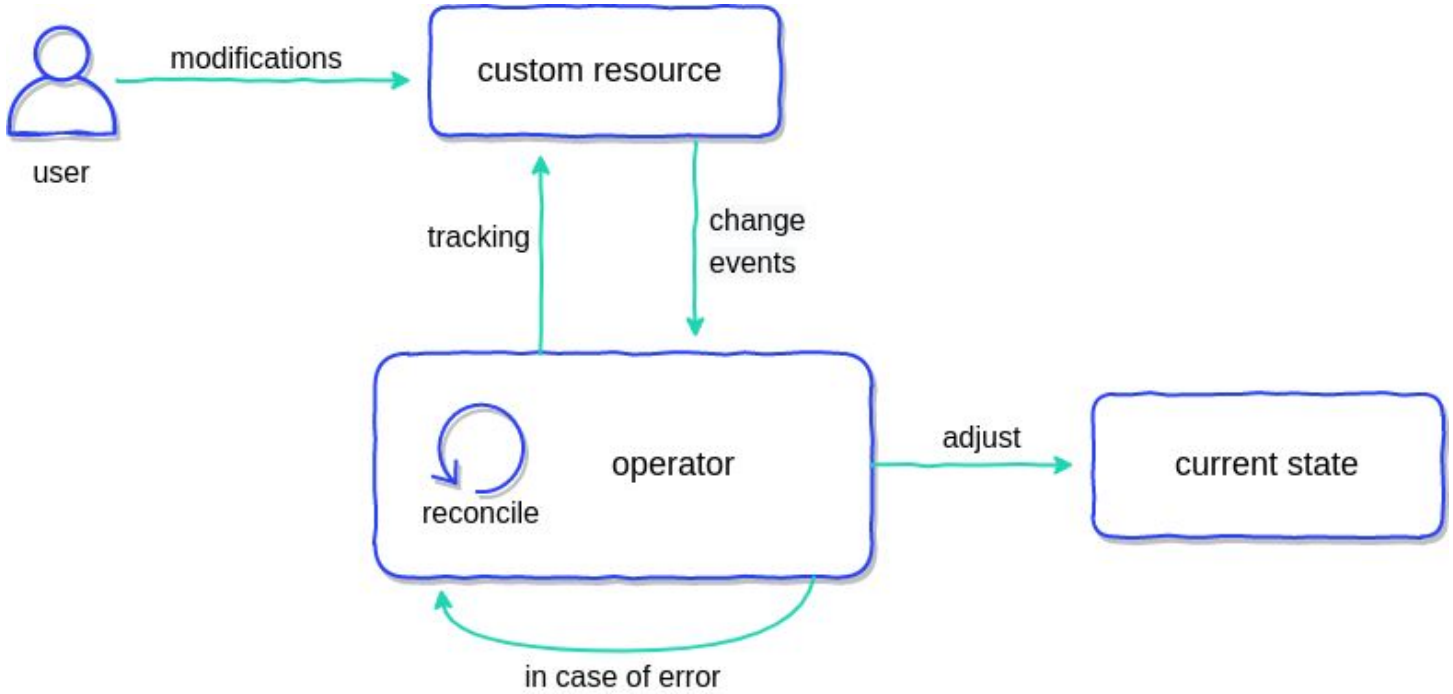
CLOUD NATIVE
COMPUTING FOUNDATION

: A Kubernetes recursion

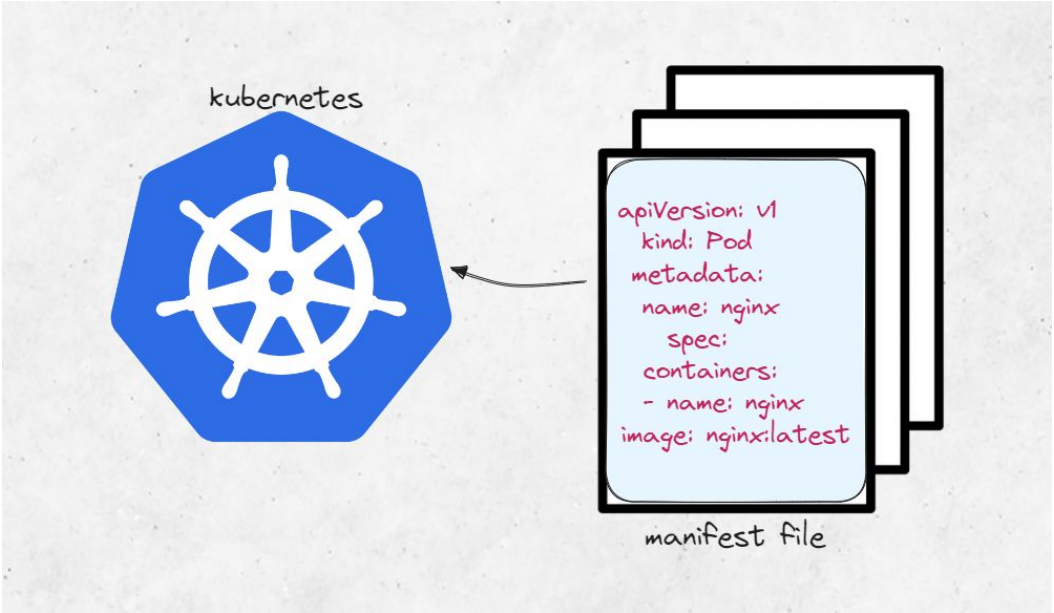
Kubernetes-like control planes for form-factors and use-cases beyond Kubernetes and container workloads.

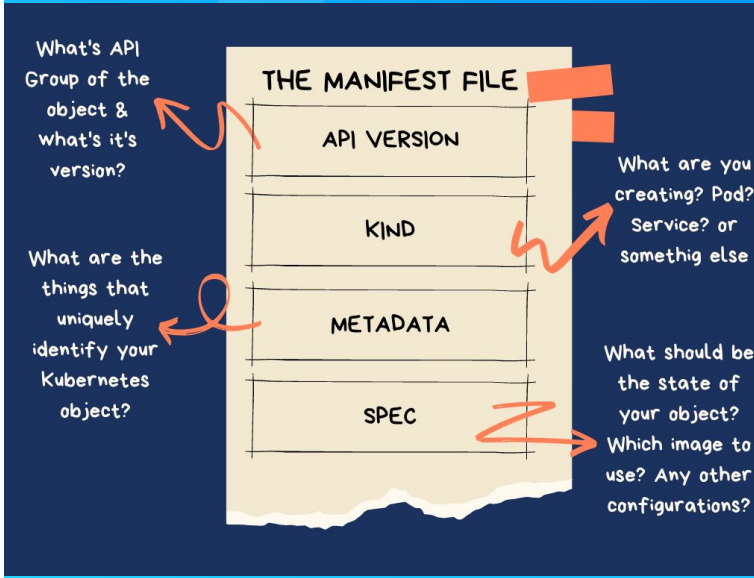
Borislava Bagaliyska, R&D engineer @ VMware Tanzu Platform

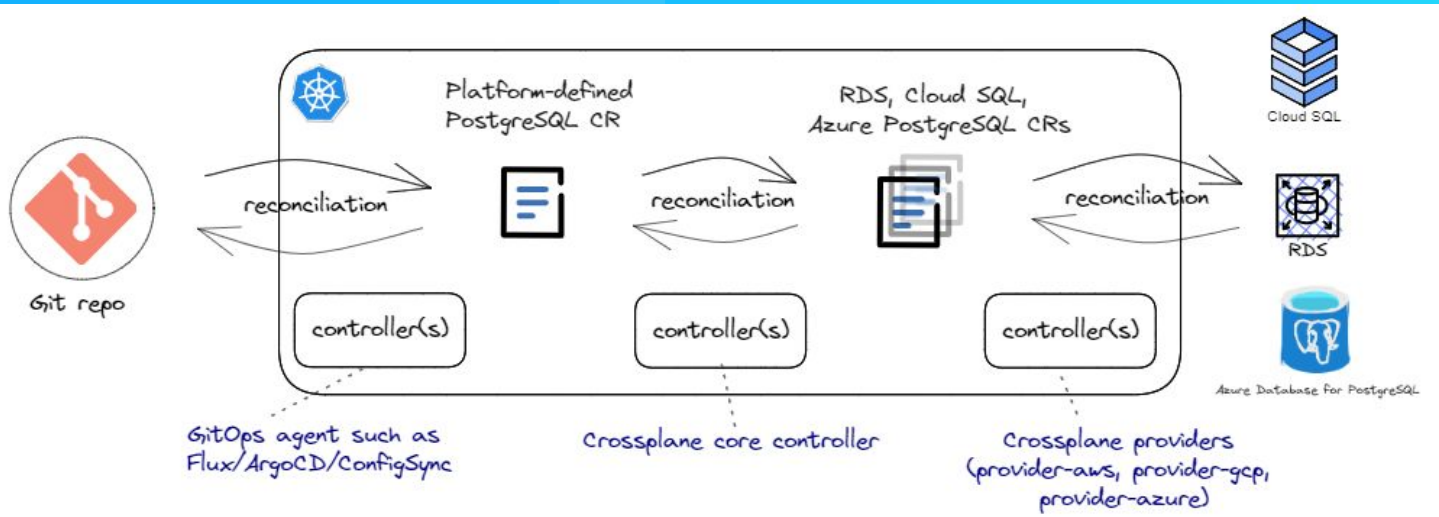
What makes Kubernetes an awesome platform?



Manifest files as APIs







Example: GitOps agent + platform defined abstraction + cloud resource provider

- 1st loop: fetches custom resource yaml files that model a custom platform-defined PostgreSQL resource from git repo and syncs them to the API server
- 2nd loop: implements any transformations from custom PostgreSQL resource down to the cloud provider specific PostgreSQL resource definitions
- 3rd loop: talks to cloud APIs and provisions the actual PostgreSQL managed service that the respective cloud offers

“Kubernetes has demonstrated the power of a well architected control plane with a great API.

The industry is beginning to notice that this control plane can be used to do much more than orchestrate containers,

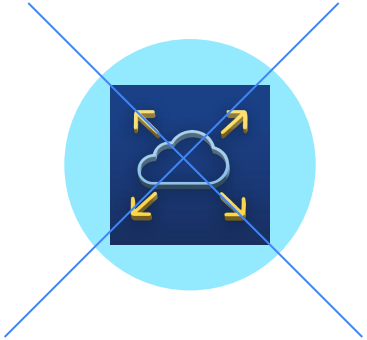
and are increasingly looking to use the Kubernetes control plane to manage all of their infrastructure.”

- Crossplane

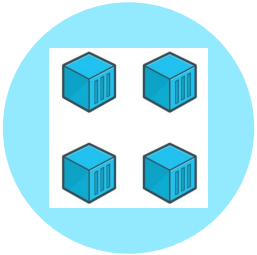

I'm a service provider and love this architecture! Can't I just use Kubernetes as my underlying platform?

Not really.

Multi-tenancy



Scalability

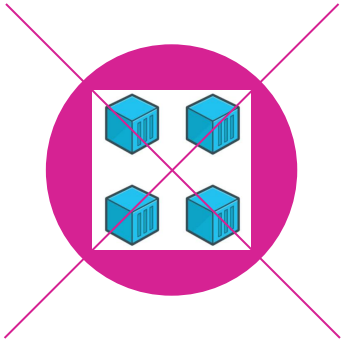


It’s still a container platform!

Enter KCP!

Multi-tenant

Scalable



No pods!

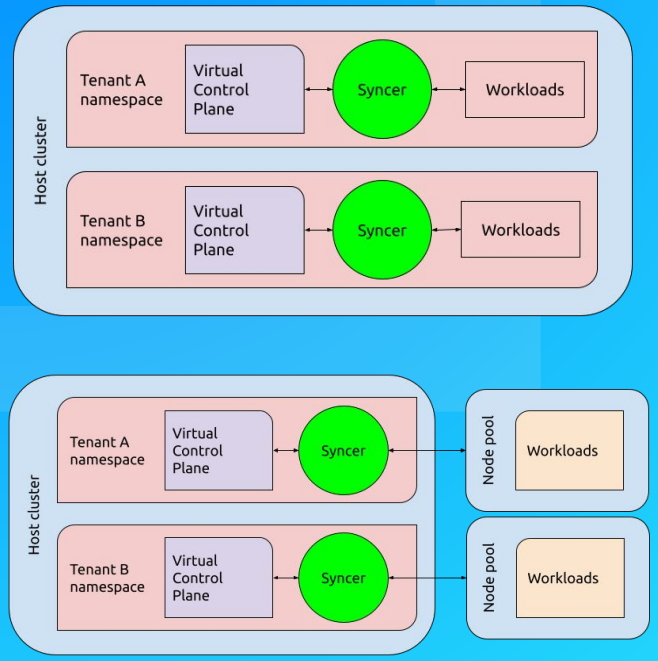
... but still all the Kubernetes

KCP’s goals

- Cloud-native control plane for Kubernetes style APIs
- Infrastructure for platform builders
- Full isolation and security of tenants
- Easily extensible
- Bring SaaS to Kubernetes
- Scalable, distributed and fault-tolerant

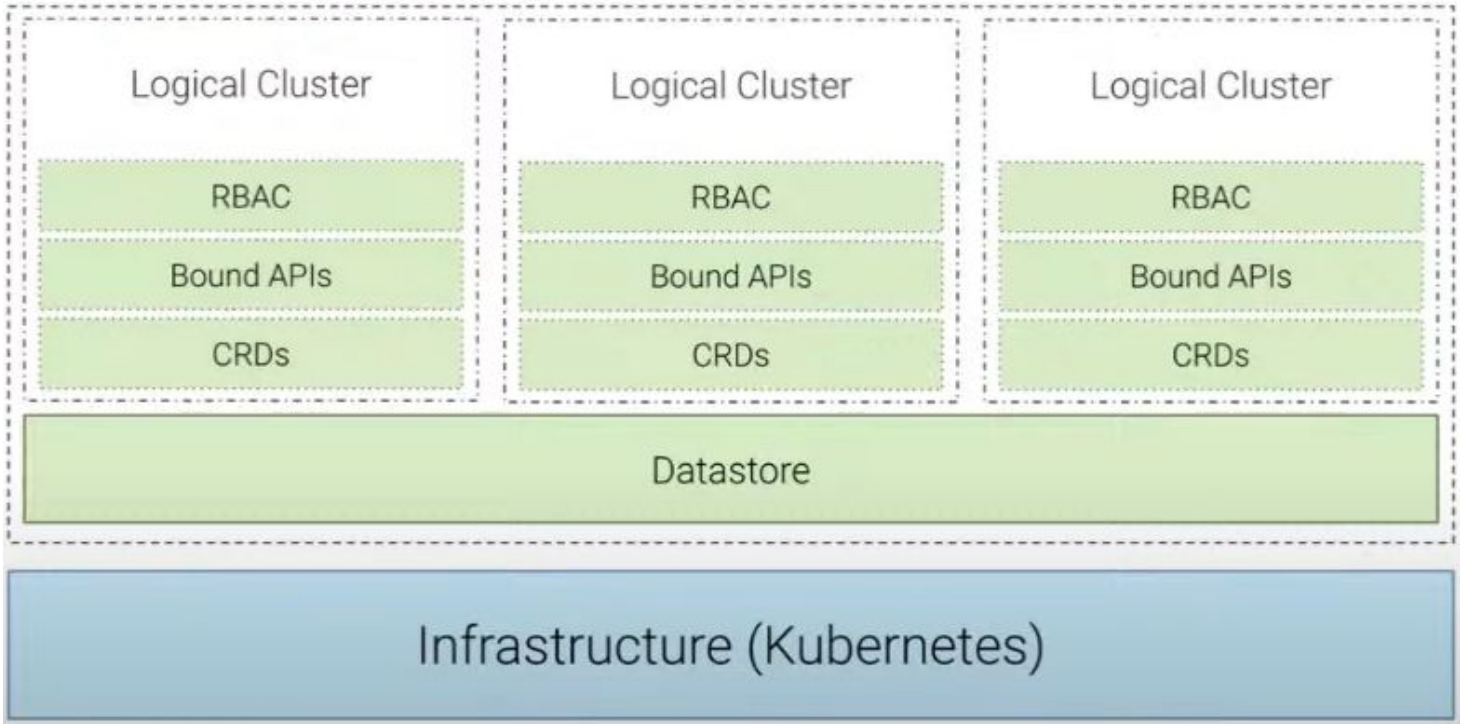
Show me how it does it!

Tenancy options in Kubernetes



- Namespaces
- Physical clusters
- Clusters in VMs
- vclusters - multiple apiservers running in namespaces of host cluster, workloads reusing host nodes
- HyperShift - same but with separate node pools

KCP: Logical clusters sharing apiserver & storage



Logical clusters objects in etcd

(group, version, resource, optional namespace, name)

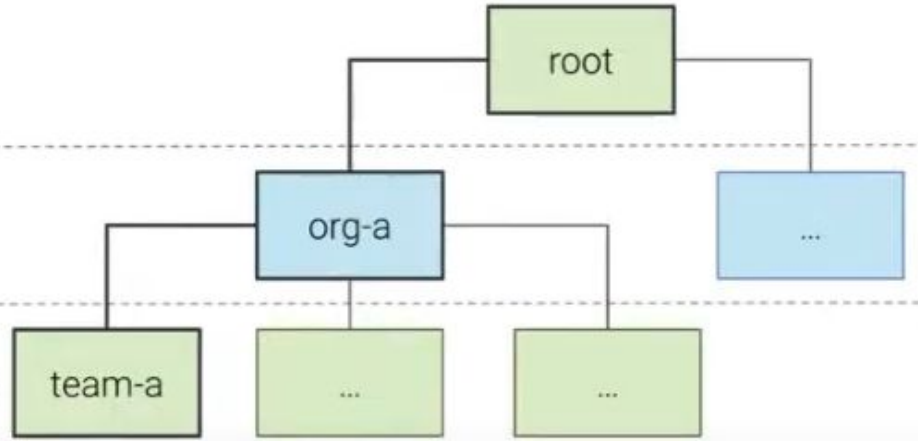
(group, version, resource, **logical cluster name**, optional namespace, name)

KCP's tenancy unit - workspaces

https://kcp:6443/clusters/root

https://kcp:6443/clusters/root:org-a

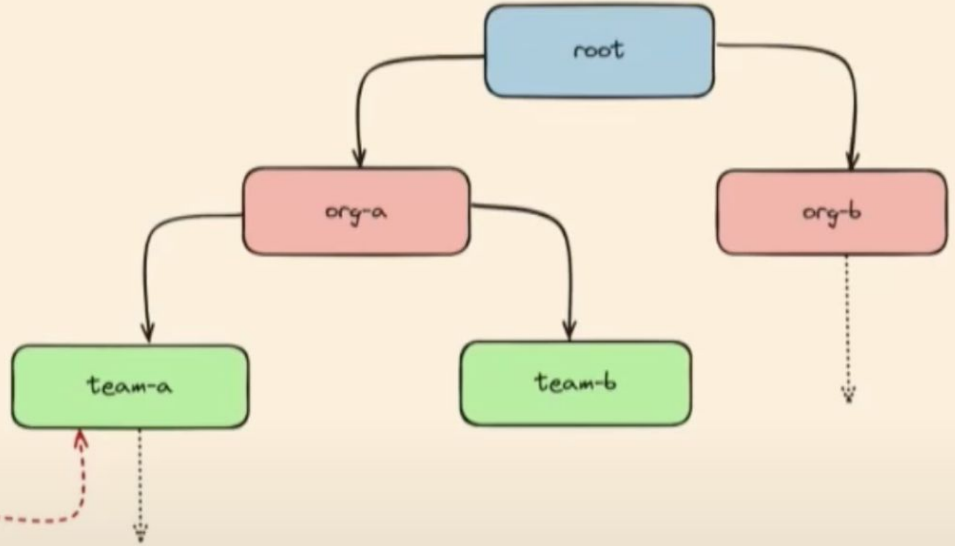
https://kcp:6443/clusters/**root:org-a:team-a**



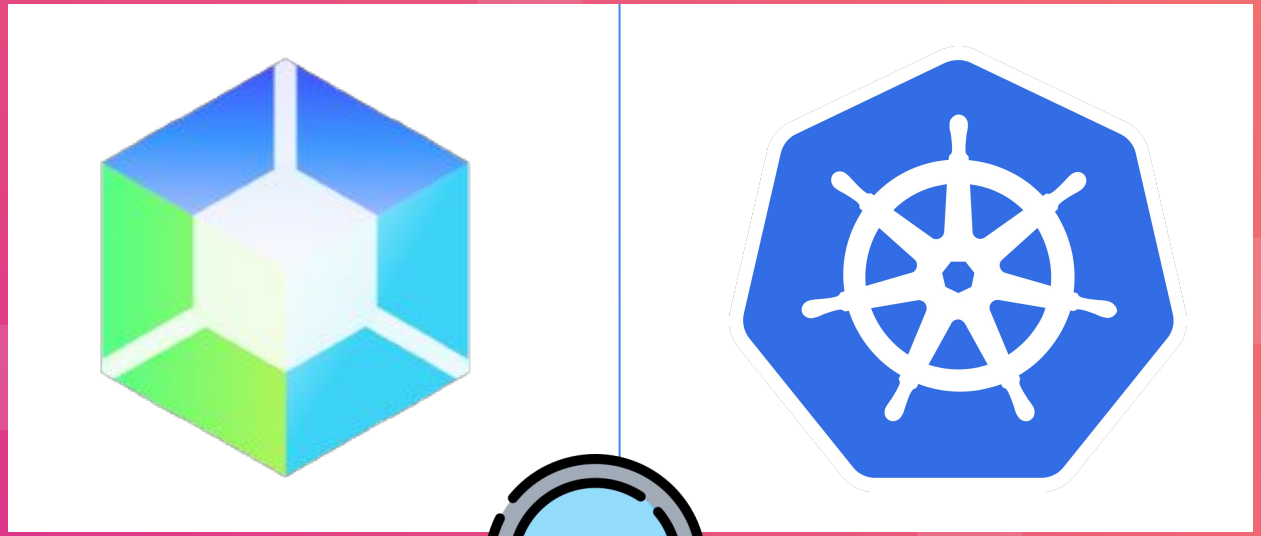
Be your cluster's admin!

Workspaces are created and navigated with kubectl CLI

```
$ kubectl ws .
Current workspace is "root".

$ kubectl get ws
NAME    TYPE           REGION   PHASE   URL          AGE
org-a   organization             Ready   https://…    69d
org-b   organization             Ready   https://…    65d

$ kubectl ws org-a
Current workspace is "root:org-a" (type root:organization).

$ kubectl get ws
NAME     TYPE   REGION   PHASE   URL          AGE
team-a   team            Ready   https://…    3m23s
team-b   team            Ready   https://…    3m18s


$ kubectl ws team-a
Current workspace is "root:org-a:team-a" (type root:team).
```



Try to spot the differences!



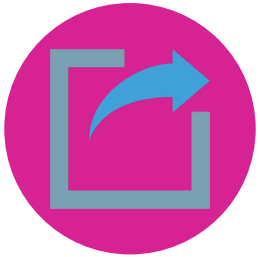
Clusters without workload related resources

https://kcp:6443/clusters/**root:org-a**/api

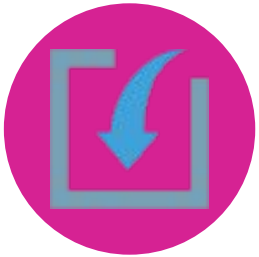
```
$ kubectl api-resources
NAME                  SHORTNAMES   APIVERSION                  NAMESPACED   KIND
configmaps            cm           v1                          true         ConfigMap
events                ev           v1                          true         Event
namespaces            ns           v1                          false        Namespace
resourcequotas        quota        v1                          true         ResourceQuota
secrets                            v1                          true         Secret
serviceaccounts       sa           v1                          true         ServiceAccount
[...]
workspaces            ws           tenancy.kcp.io/v1alpha1     false        Workspace
workspacetypes                     tenancy.kcp.io/v1alpha1     false        WorkspaceType
[...]
```

Adding APIs in workspaces

APIResourceSchema

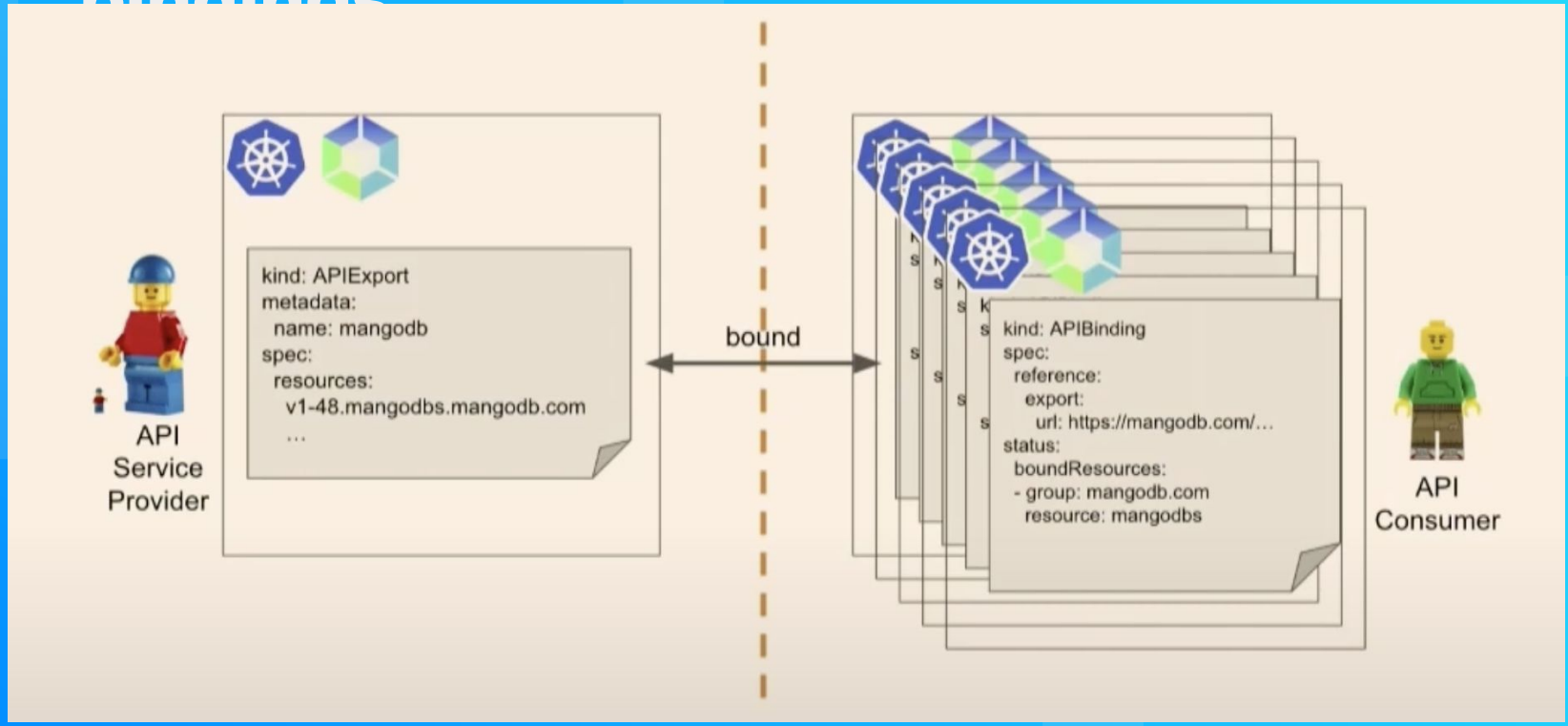


APIExport

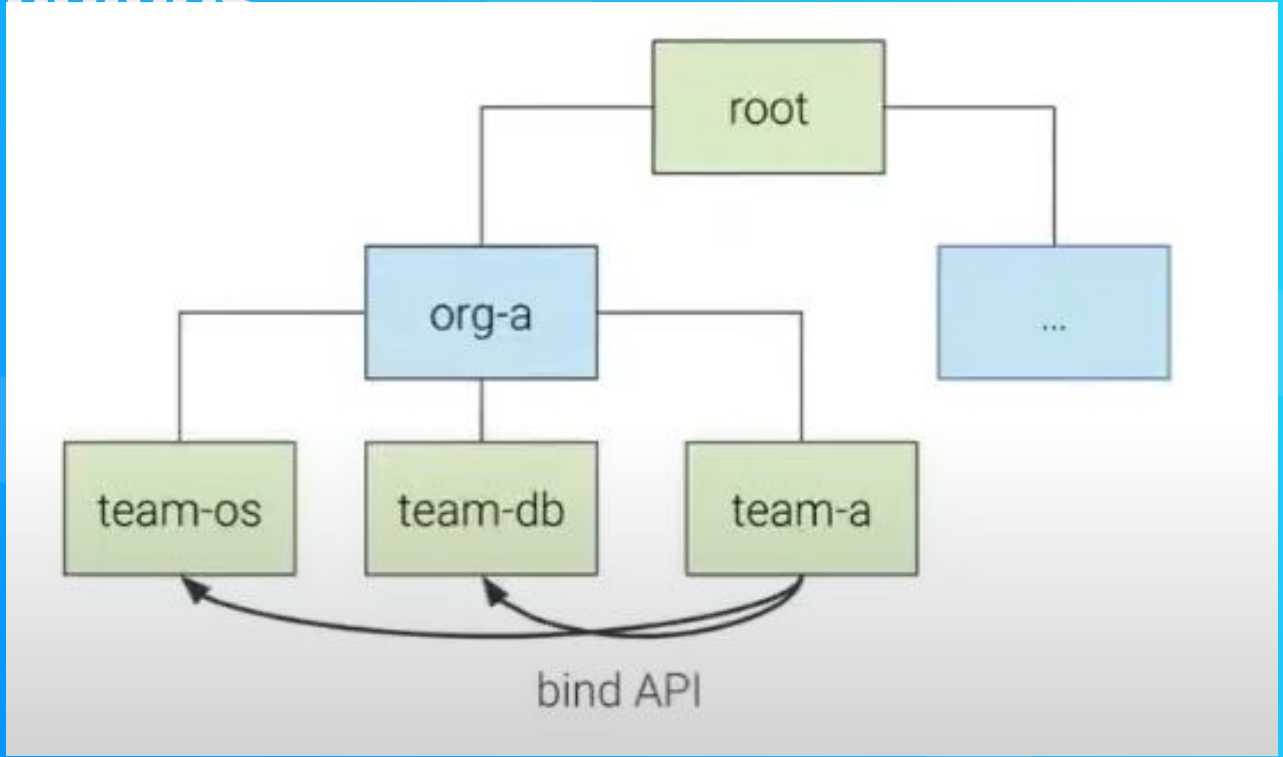


APIBinding

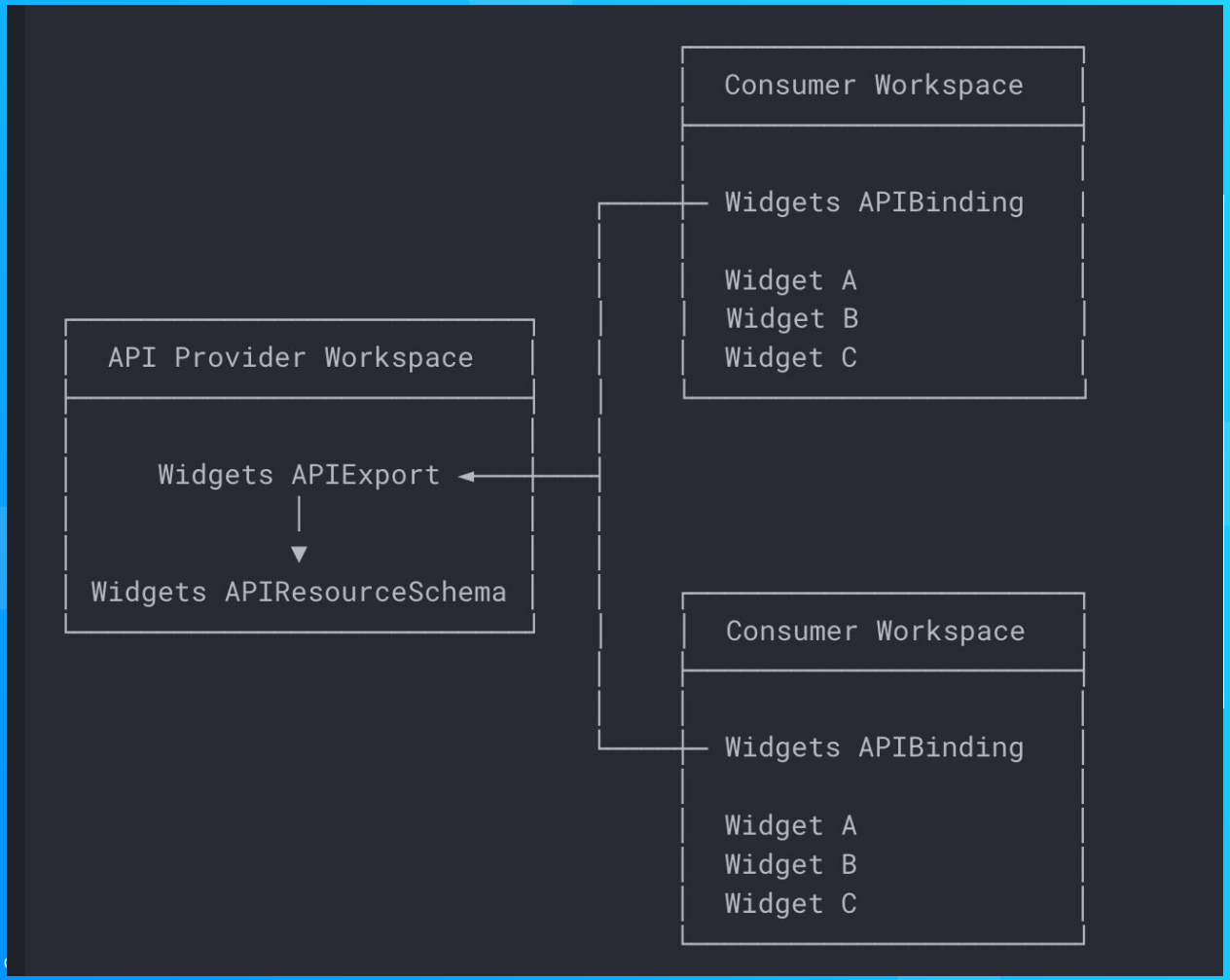
API bindings

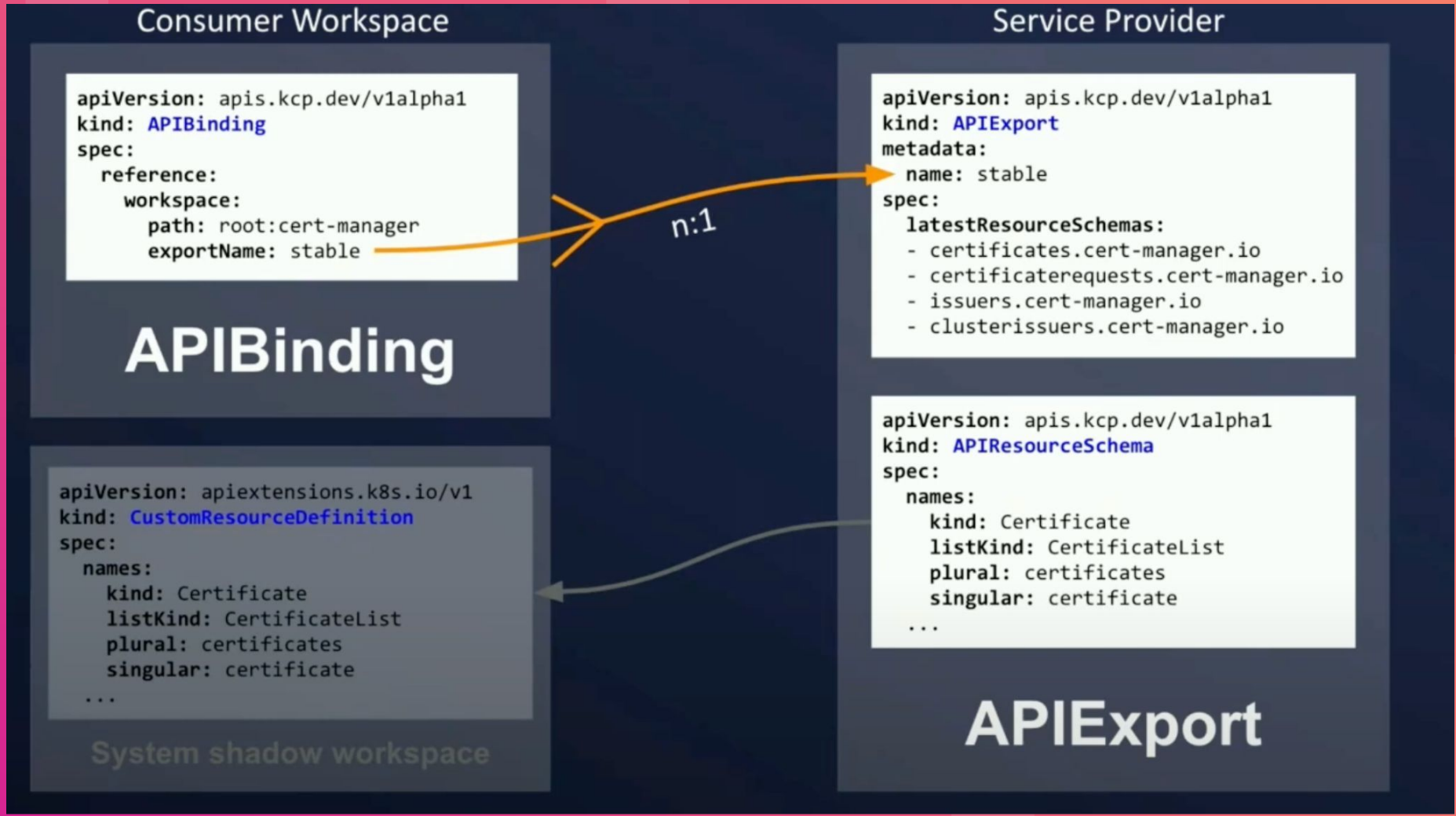


API bindings

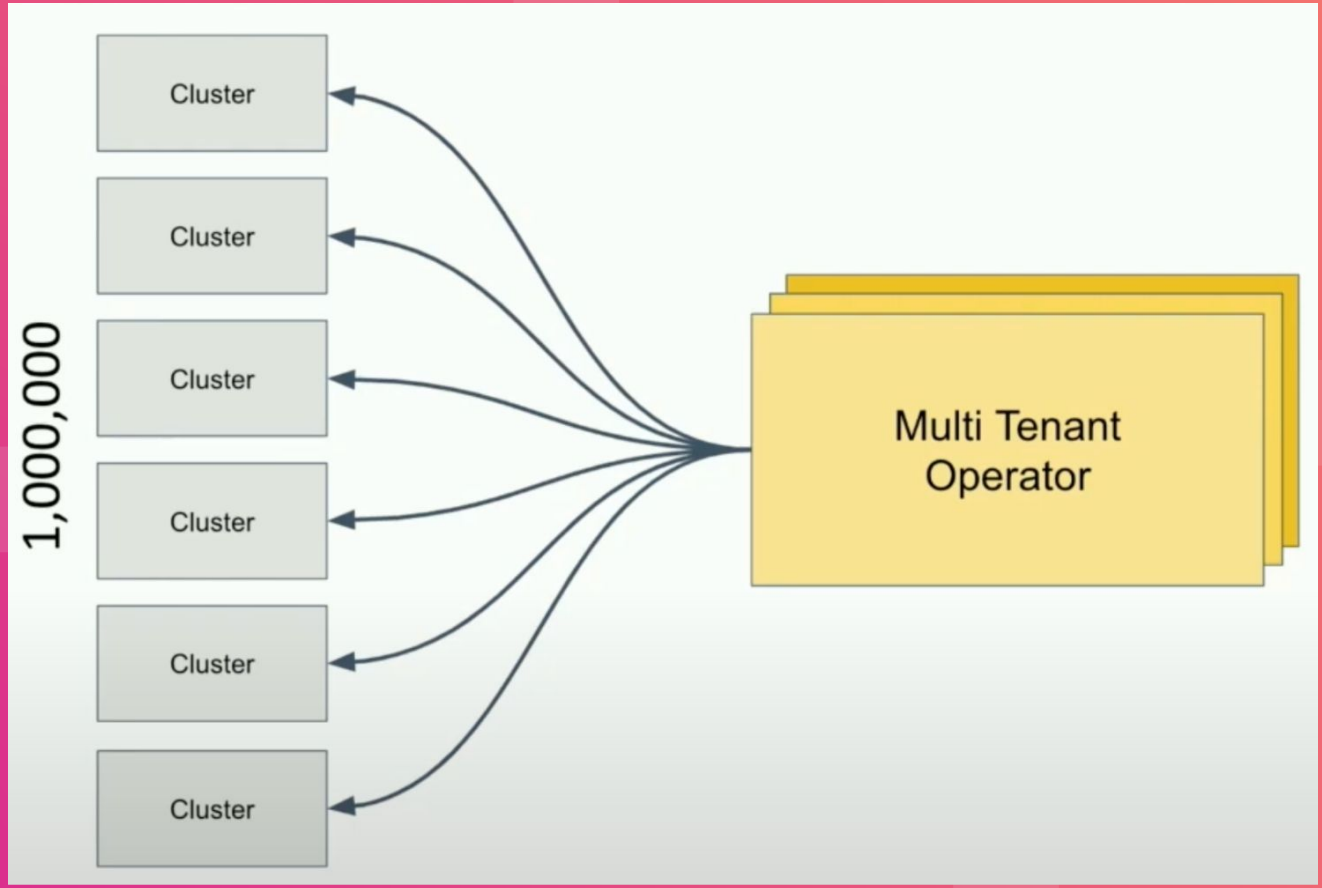


(team-a ws) kubectl bind apiexport root:org-a:team-db:db-export

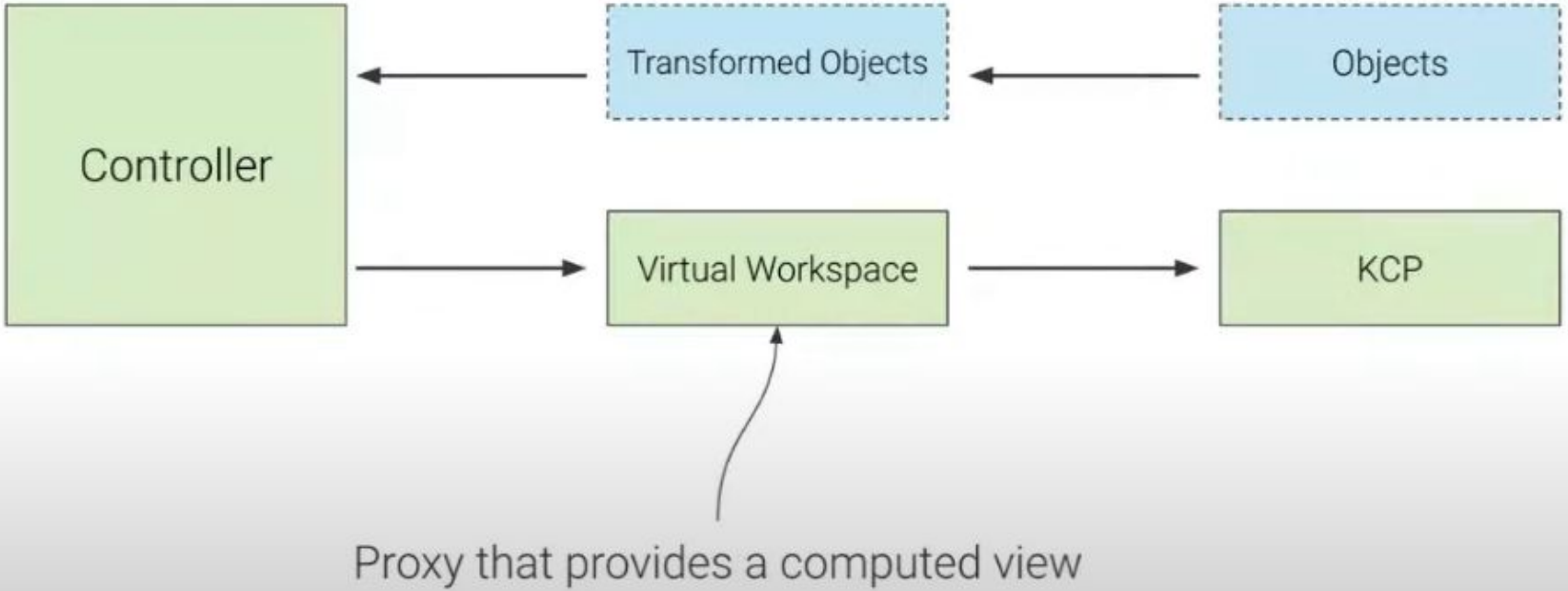



Consumer Workspace
Service Provider
apiVersion: apis.kcp.dev/v1alpha1
kind: APIBinding
spec:
reference:
workspace:
path: root:cert-manager
exportName: stable
n:1
APIBinding
apiVersion: apis.kcp.dev/v1alpha1
kind: APIExport
metadata:
name: stable
spec:
latestResourceSchemas:
- certificates.cert-manager.io
- certificaterequests.cert-manager.io
- issuers.cert-manager.io
- clusterissuers.cert-manager.io
apiVersion: apis.kcp.dev/v1alpha1
kind: APIResourceSchema
spec:
names:
kind: Certificate
listKind: CertificateList
plural: certificates
singular: certificate
...
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
spec:
names:
kind: Certificate
listKind: CertificateList
plural: certificates
singular: certificate
...
System shadow workspace
APIExport

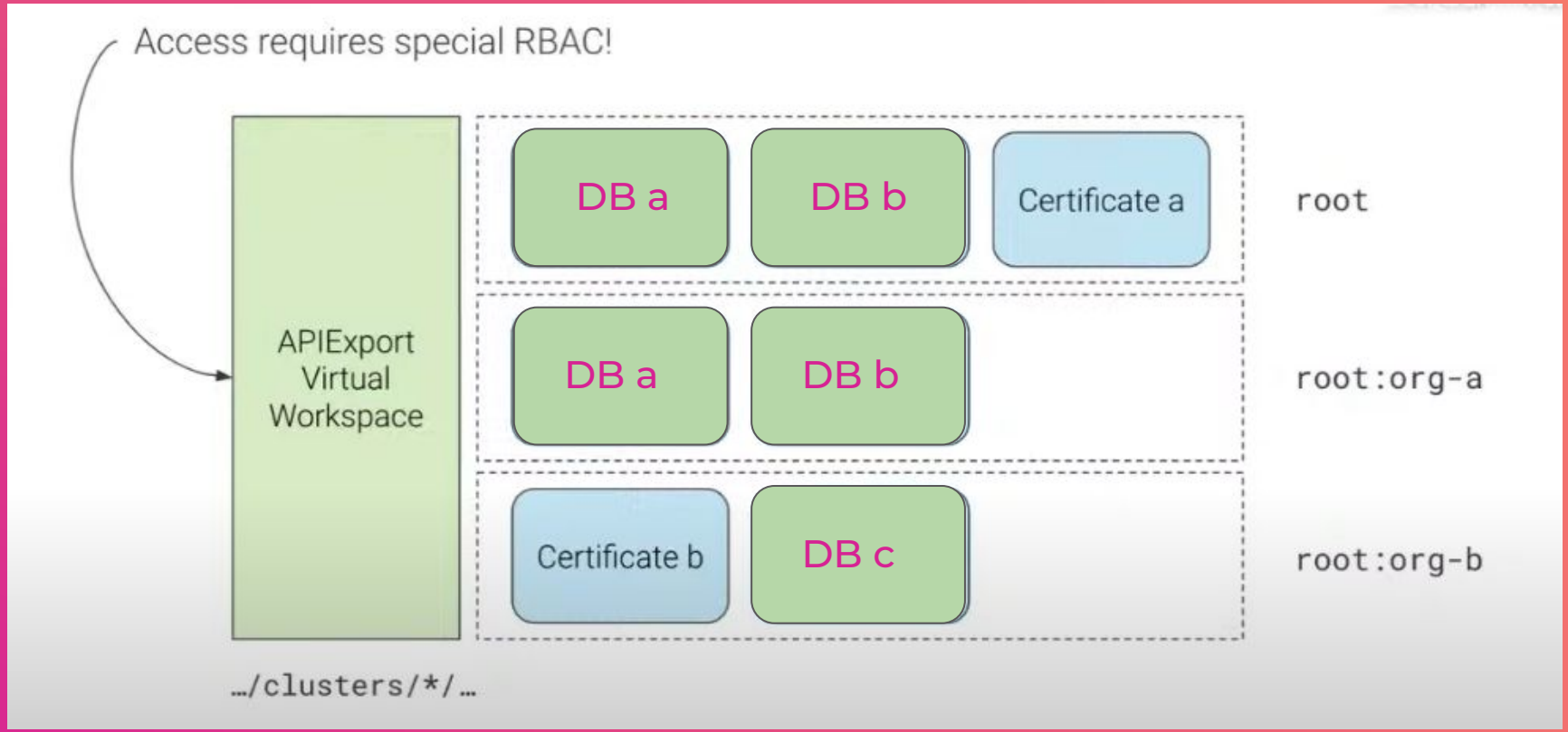
How to reconcile objects in workspaces?



How does the controller work?



How does the controller work?



\+ KCP aware controller-runtime fork

How is this secure?

1. Organizational access authorizer
2. “**access**” verb for granting access to workspace content
3. “**use**” verb for creating workspaces of particular type
4. Workspace level k8s RBAC
5. “**bind**” verb for creating APIBinding to a particular APIExport
6. Permission claims for granting APIExport controller access to workspace resources
7. Maximal permission policies for exported resources
8. Identity hashes of APIExports

Requesting and granting permission

A service provider wanting to access `ConfigMaps` needs to specify such a claim in the `APIExport`:

```
spec:
...
  permissionClaims:
    - group: ""
      resource: "configmaps"
```

Users can then authorize access to this resource type in their workspace by accepting the claim in the `APIBinding`:

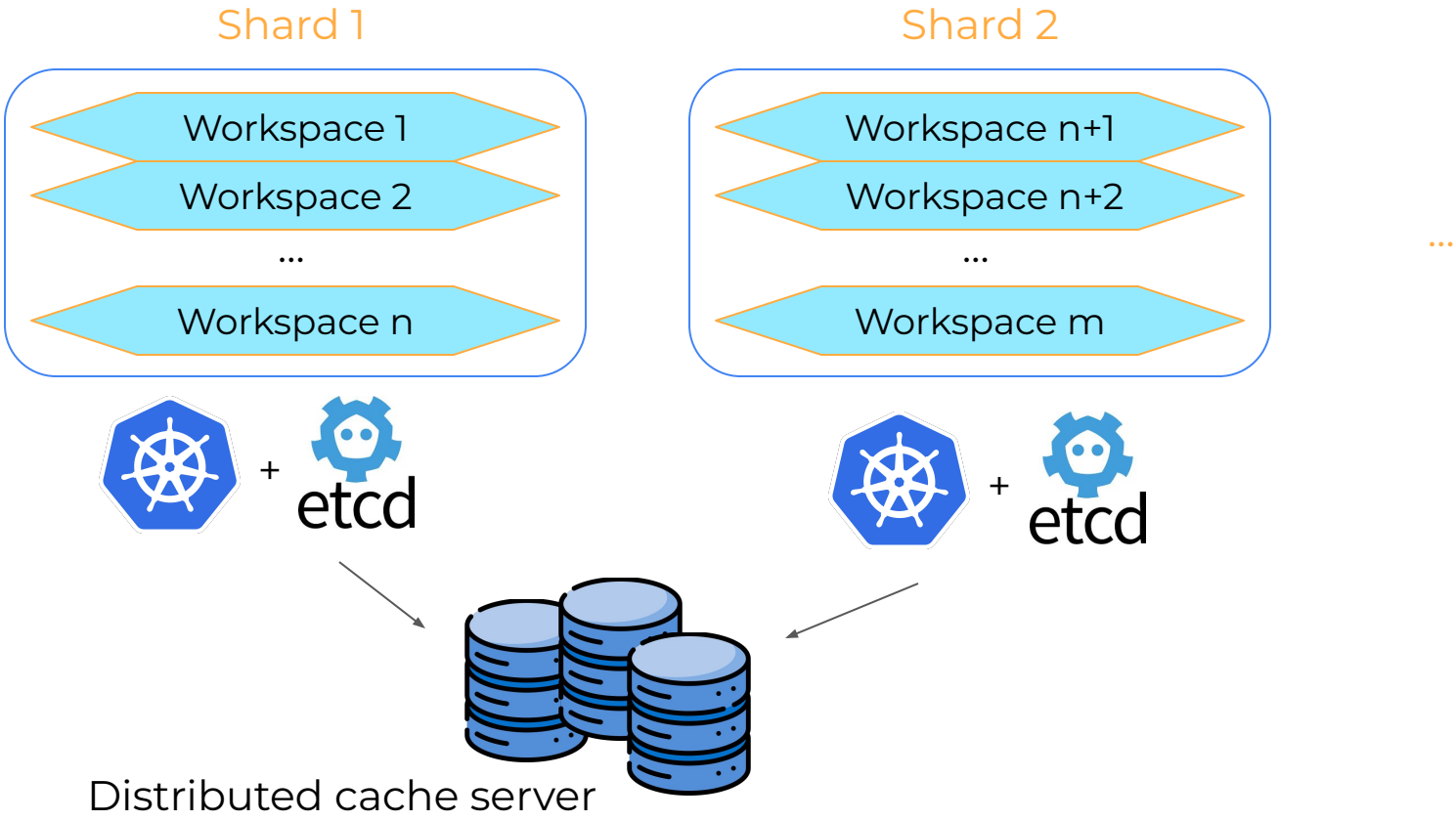
```
spec:
...
  permissionClaims:
    - group: ""
      resource: "configmaps"
      state: Accepted
```


Advanced permission claims

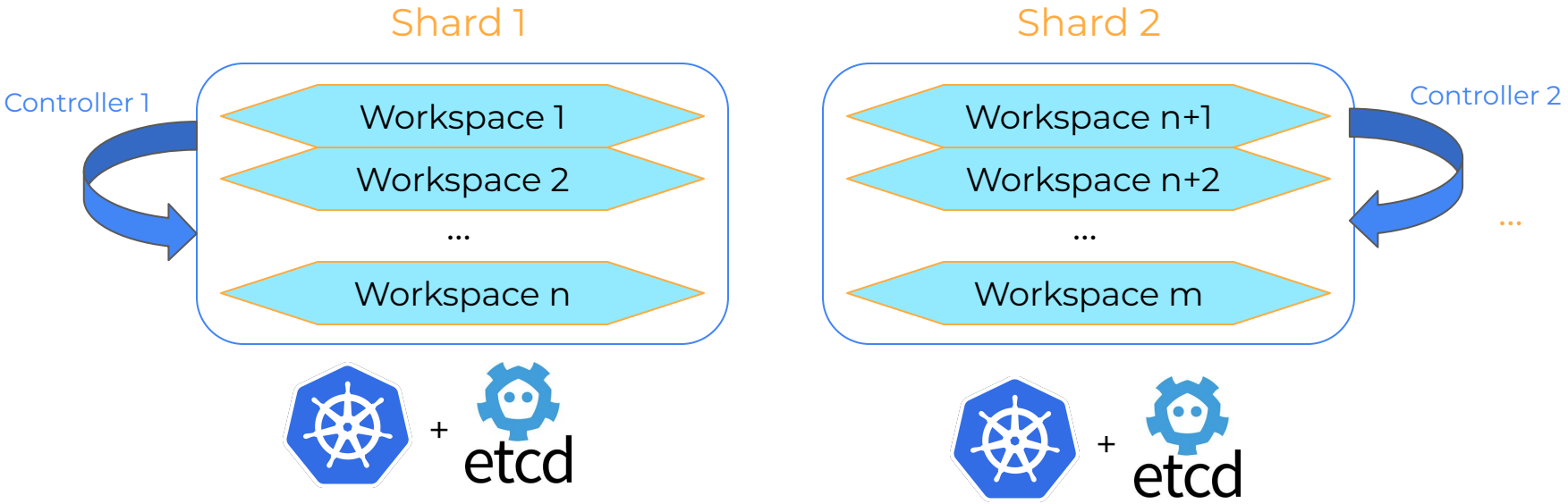
```yaml
apiVersion: apis.kcp.io/v1alpha1
kind: APIExport
metadata:
  name: example.kcp.io
spec:
  latestResourceSchemas:
  - v220801.widgets.example.kcp.io
  permissionClaims:
  - group: "" #
    resource: configmaps
    resourceSelector: #
    - namespace: example-system
      name: my-setup
  - group: somegroup.kcp.io
    resource: things
    identityHash: 5fdf7c7aaf407fd1594566869803f565bb84d22156cef5c445d2ee13ac2cfca6
    all: true #
```

Scalability by sharding

Shards architecture



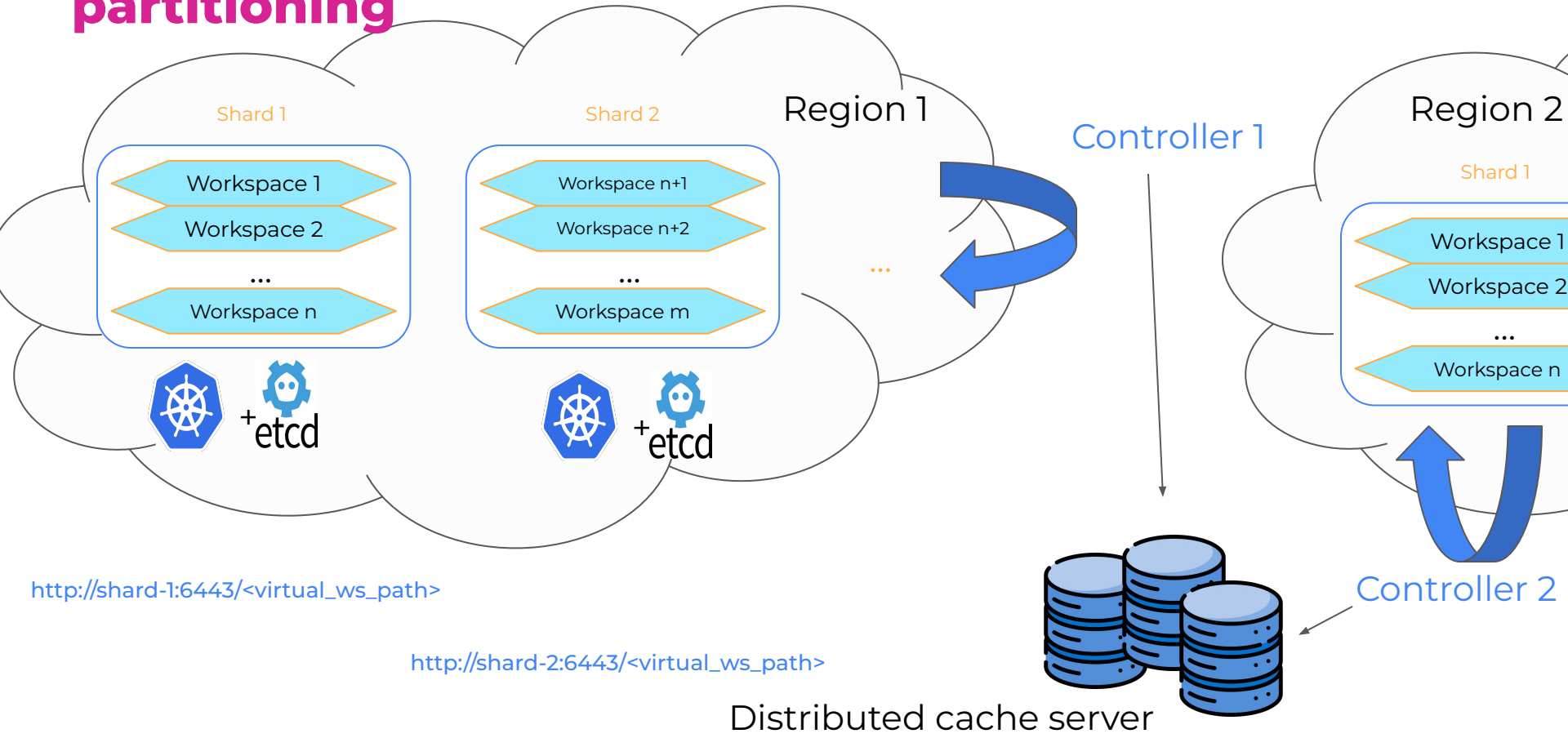
APIExport controllers in shards



http://shard-1:6443/<virtual_ws_path>

http://shard-2:6443/<virtual_ws_path>

APIExport controllers in shards - partitioning



Controller implementation sneak peek

Partitions and APIExportEndpointSlices

```
$ kubectl get apiexportendpointslice example-gcp-europe -o yaml
kind: APIExportEndpointSlice
apiVersion: apis.kcp.io/v1alpha1
metadata:
    name: example-gcp-europe
...
status:
    endpoints
        - url: https://host1:6443/services/apiexport/root/example.kcp.io
        - url: https://host2:6443/services/apiexport/root/example.kcp.io
...
```

Use cases

Service catalogs

IoT & Edge

Internal developer platforms

IaaS platforms

e.g. Crossplane,

Kubernetes LCM

Transparent multi-cluster

Shortcomings

Low maturity

CNCF sandbox project since Sept 2023

Not suitable for large data

Eventuall y consisten t by nature

Any questions? 😊

Resources

- Official docs: https://docs.kcp.io/kcp/main/
- KCP repo: https://github.com/kcp-dev/kcp
- KCP controller-runtime fork: https://github.com/kcp-dev/controller-runtime
- Kcp: Towards 1,000,000 Clusters - Stefan Schimanski, Red Hat - https://www.youtube.com/watch?v=fGv5dpQ8X5I

Thank you!